The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points*

Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs,† and Hui Zhang
Carnegie Mellon University

ABSTRACT

While application end-point architectures have proven to be viable solutions for large-scale distributed applications such as distributed computing and file-sharing, there is little known about its feasibility for more bandwidth-demanding applications such as live streaming. Heterogeneity in bandwidth resources and dynamic group membership, inherent properties of application end-points, may adversely affect the construction of a usable and efficient overlay. At large scales, the problems become even more challenging. In this paper, we study one of the most prominent architectural issues in overlay multicast: the feasibility of supporting large-scale groups using an application end-point architecture. We look at three key requirements for feasibility: (i) are there enough resources to construct an overlay, (ii) can a stable and connected overlay be maintained in the presence of group dynamics, and (iii) can an efficient overlay be constructed? Using traces from a large content delivery network, we characterize the behavior of users watching live audio and video streams. We show that in many common real-world scenarios, all three requirements are satisfied. In addition, we evaluate the performance of several design alternatives and show that simple algorithms have the potential to meet these requirements in practice. Overall, our results argue for the feasibility of supporting large-scale live streaming using an application end-point architecture.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Measurement, Perfomance

*This research was sponsored by DARPA under contract number F30602-99-1-0518, and by NSF under grant numbers Career Award NCR-9624979 ANI-9730105, ITR Award ANI-0085920, ANI-9814929, and ANI-0331653. Additional support was provided by Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

†Bruce Maggs is also with Akamai Technologies.

SIGCOMM'04, Aug. 30–Sept. 3, 2004, Portland, Oregon, USA.

Keywords

Overlay multicast, application-level multicast, peer-to-peer, live streaming

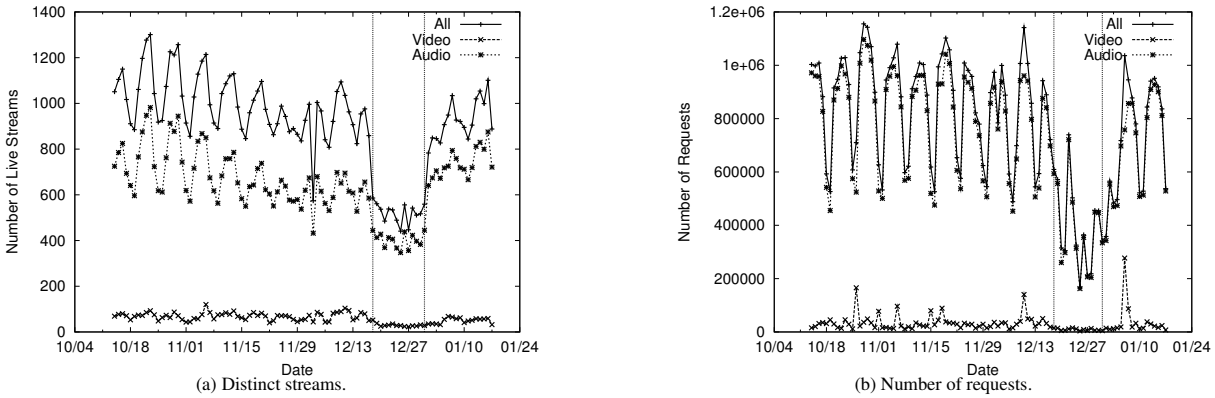
1. INTRODUCTION

Live audio and video streams are now being delivered successfully over the Internet on a large scale. Commercial content delivery networks such as Akamai Technologies [1] and Real Networks [21] have developed and deployed large-scale dedicated infrastructure to deliver both live streams and video-on-demand. These architectures are capable of supporting many simultaneous streams and clients.

In contrast to infrastructure-based content delivery networks, application end-point overlay multicast has recently received attention [9, 8, 12, 14, 20, 25, 5, 2, 18, 24, 13, 4]. In such an architecture, application end-points organize themselves into an overlay structure and data is distributed along the links of the overlay. The responsibility and cost of providing bandwidth is shared amongst the application end-points, reducing the burden at the content publisher. The lack of dependence on infrastructure support makes application end-point architectures easy to deploy, and economically viable. The ability for users to receive content that they would otherwise not have access to provides a natural incentive for them to contribute resources to the system. Application end-point architectures have shown promise for events where the peak group size is small, on the order of 10 to 100 nodes [7]. However, the question remains whether or not such architectures are feasible at larger scales of 1,000s to 100,000s of nodes.

We believe that the first step towards answering the feasibility question is to obtain a better understanding of the application workload and the characteristics of application end-points. We leverage the wealth of data from Akamai Technologies, a large content delivery network that provides live streaming services. Because the system has been in everyday use for several years, the data reflects common "real-world" application end-point characteristics and behavior that may impact the choice of architectures. Inherent properties of application end-point architectures, such as heterogeneity in bandwidth resources and dynamic group membership could adversely affect the feasibility of constructing a usable overlay for data delivery.

We look at three key requirements for feasibility: (i) there must be enough resources to construct an overlay, (ii) a stable and connected overlay must be maintained in the presence of group dynamics, and (iii) the overlay structure must be efficient. These three requirements are fundamental in the sense that an application end-point architecture has little chance of providing good performance if these requirements are not satisfied. We find that in the majority of common scenarios, application end-point architectures have sufficient inherent resources and stability. In addition, efficient overlay



(a) Distinct streams.
(b) Number of requests.

Figure 1: Summary of live streams.

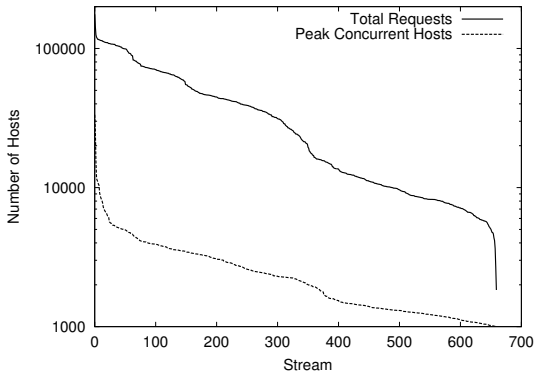


Figure 2: Size of large-scale streams.

structures can be constructed using simple algorithms. Our findings argue for the feasibility of using application end-point architectures for large-scale live streaming applications.

In Section 2, we describe and analyze the streaming media workload that we use in our study. Section 3 studies the feasibility of constructing overlays using the resources available at application end-points. In Section 4, we look at overlay stability and evaluate algorithms to maintaining a stable overlay in the presence of dynamic group membership. Section 5 looks at the construction of efficient overlays. We summarize our findings in Section 6.

2. LIVE STREAMING WORKLOAD

In this section, we analyze the live streaming workload from a large content delivery network to better understand the design requirements for building a live streaming system. We focus our analysis on characteristics that are most likely to impact design, such as group dynamics. In the following sections, we evaluate the impact of the workload on the performance of an application end-point architecture.

2.1 Data Collection and Summary Statistics

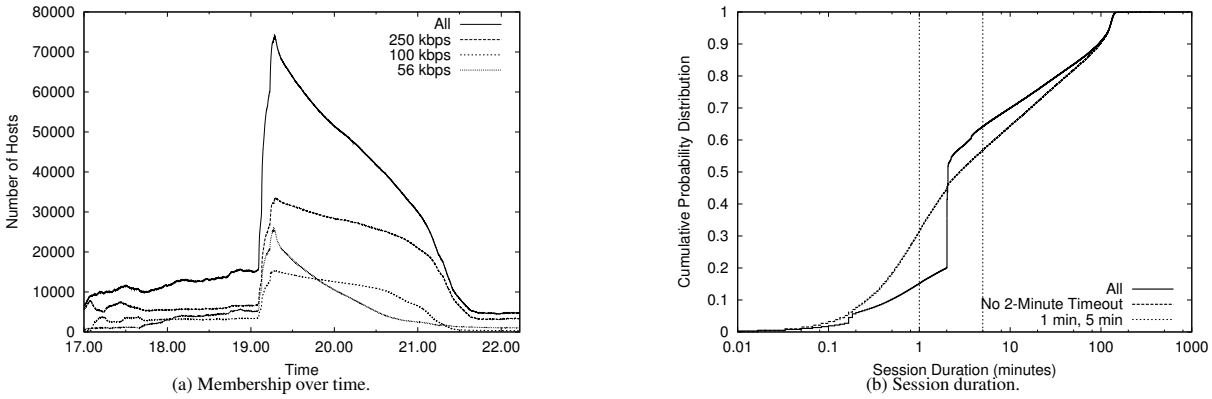
The logs used in our study are collected from the thousands of streaming servers belonging to Akamai Technologies. Akamai's streaming network is a static overlay composed of (i) edge nodes located close to clients, and (ii) intermediate nodes that take streams from the original content publisher and split and replicate them to the edge nodes. The logs that we use in this study are from the edge nodes that directly serve client requests.

The logs were collected over a 3-month period from October 2003 to January 2004. The daily statistics for live streaming traffic during that period is depicted in Figure 1. The traffic consists of three of the most popular streaming media formats, QuickTime, Real, and Windows Media. In Figure 1(a), there were typically 800-1000 distinct streams on most days. However, there was a sharp drop in early December and a drop again from mid-December to January (denoted by the vertical lines). This is because we had a problem with our log collection infrastructure and did not collect logs for one of formats on those days. To classify streams as audio or video streams, we look at the encoding bit rate. If the bit rate is under 80 kbps, then it is classified as audio. Roughly 71% of the streams are audio, and 7% are video streams. We did not classify 22% of the streams because there was insufficient information about their streaming bit rates. Figure 1(b) depicts the number of requests for live streams which varies from 600,000 on the weekends to 1 million on weekdays. Again, the drop in requests from mid-December onwards is due to the missing logs. Note that there are an order of magnitude more requests for audio streams than video streams. In addition, audio streams have extremely regular weekend/weekday usage patterns. On the other hand, video streams are less regular, and are more dominated by "short duration" special events with the sharp peaks corresponding to very large events on various days.

Streaming media events can be classified into two broad categories based on the event duration. The first category, which we call *non-stop events,* are events in which there is live broadcast every day, all hours of the day. This is similar to always being "on-the-air" in radio terminology. The second category, which we call *short duration events,* are events with well-defined durations, typically on the order of a couple of hours. A typical example is a talk show that runs from 9am-10am that is broadcast only during that period, and has no traffic at any other time during the day. For simplicity, for either one of these categories, we break the events into 24-hour chunks, which we call *streams*. For the rest of the paper, we present analysis on the granularity of streams. Note that for *short duration events*, a stream is the same as an event.

In this paper, we limit the discussion to *large-scale streams*. Large-scale streams are defined as the streams in which the peak group size, i.e., the maximum concurrent number of participating hosts, is larger than 1,000 hosts. There were a total of 660 large-scale streams, of which 55 were video streams and 605 were audio streams. Many of the audio streams are *non-stop*, and all of the video streams are *short duration*.

Figure 2 depicts the peak group size and the total number of



(a) Membership over time.
(b) Session duration.

Figure 3: Largest event.

requests for each stream on the y-axis. Each point on the x-axis represents a stream. Note that the same value on the x-axis on the two curves do not necessarily correspond to the same stream. Across all large-scale streams, the peak group size ranges from 1,000 to 80,000 hosts, and the total number of requests ranges from 2,000 to 200,000. In addition to group size, we also summarize the session duration characteristics, which are analyzed in detail in Section 4.

2.2 Workload Processing

Entity vs. incarnation: We define an entity as a unique host, corresponding to an IP address. An entity or host may join the broadcast many times, perhaps to tune in to distinct portions of the broadcast, and as a result have many incarnations.

Log format: Each entry in the log corresponds to an incarnation's session, or a request made by a user to an edge server. The following fields extracted from each entry are used in our study.

- User identification: IP address
- Requested object: stream URI
- Time-stamps: session start time and duration in seconds

2.3 Largest Event

Next, we present more detailed statistics for the largest event in the logs. The event consisted of three encoded streams at bit rates of 20 kbps audio, 100 kbps audio and video, and 250 kbps audio and video. The event duration was 2 hours, from 19:00-21:00, as shown in Figure 3(a). The sharp rise in membership at 19:00 is a flash crowd caused by everyone wanting to tune in to the start of the event. We note that flash crowds are common in our logs – about 40% of large-scale streams have flash crowds. Combining all three streams, the peak group size was 74,000 users. There were roughly 119,000 IP addresses and over 394,000 requests for the entire duration of the event. Note that there are roughly 3.3 requests/IP address. This may be caused by (i) users that join the broadcast many times, perhaps to tune in to distinct portions of the broadcast, or (ii) multiple users that share the same IP addresses in the case of network address translators (NATs), proxies, or DHCP.

Note that there are many people who join the broadcast before the event started, perhaps to test that their setups are working. Also, there are many people who stay after the broadcast is over, for unknown reasons (they may have just left their media players running). The 20 kbps audio stream has a sharper drop-off in membership than the other streams because many users switch from audio to a better quality stream. The most dynamic segment of the stream was between 19:00-19:30, where the peak join rate was over 700 joins/second and the peak leave rate was over 70 leaves/second.

Figure 3(b) depicts the cumulative distribution of session duration time in minutes for the combined streams. There is a sharp rise at 2 minutes caused by a known client-side NAT/firewall problem that forces streaming servers to time out on the connection. The second curve depicts the session duration distribution without the incarnations that experienced 2-minute timeouts. The average session duration is 22 minutes, with 20% of sessions lasting longer than 30 minutes. The average is dominated by the tail of the distribution. Except for the tail, the behavior is fairly dynamic. For example, over 55% of sessions are shorter than 5 minutes. Furthermore, 30% of sessions are shorter than 1 minute. We explore what these numbers indicate about the stability of the system in Section 4.

Unless otherwise stated, in the remaining sections, we combine all three streams of this event into one stream, and refer to this one stream as the "largest stream or event." We assume that the "desired" encoding for all hosts is 250 kbps.

3. ARE THERE ENOUGH RESOURCES?

In an application end-point architecture, there is no dependence on costly pre-provisioned infrastructure, making it favorable as an economical and quickly deployable alternative for streaming applications. On the other hand, the lack of any supporting infrastructure requires that application end-points contribute their outgoing bandwidth resources. The feasibility of such an architecture depends on whether or not there are enough bandwidth resources at application end-points to support all participants at the encoding bit rate.

In this section, we look at the feasibility of supporting a large-scale live streaming application using only the resources available at application end-points. To answer this feasibility question, we run trace-based analysis on the large-scale streams. We first need to estimate the outgoing bandwidth resources at each host. Then, using these estimates, we derive the amount of resources for each stream over time. Note that the amount of resources is dependent on the patterns of joining and leaving of participating hosts.

3.1 Outgoing Bandwidth Estimation

We define resources as the amount of outgoing bandwidth that hosts in the system can contribute (i.e., how much bandwidth each host can send). Next, we describe the methodology that we use to quickly and accurately estimate the outgoing bandwidth of over one million IP addresses of hosts that participated in the large-scale streams.

3.1.1 Bandwidth Data Collection

We use a combination of data mining, inference, and active measurements for the estimation. While the most accurate method-

Access technology	Packet-pair measurement	Outgoing bandwidth estimate
Dial-up modems	0 kbps $\leq$ BW $<$ 100 kbps	30 kbps
DSL, ISDN, Wireless	100 kbps $\leq$ BW $<$ 600 kbps	100 kbps
Cable modems	600 kbps $\leq$ BW $<$ 1 Mbps	250 kbps
Edu, Others	BW $\geq$ 1 Mbps	BW

Table 1: Mapping of access technology to outgoing bandwidth.

Type	Degree-bound	Number of hosts
Free-riders	0	58646 (49.3%)
Contributors	1	22264 (18.7%)
Contributors	2	10033 (8.4%)
Contributors	3-19	6128 (5.2%)
Contributors	20	8115 (6.8%)
Unknown	-	13735 (11.6%)
Total	-	118921 (100%)

Table 2: Assigned degree for the largest event.

ology would be to actively measure the bandwidth of all the IP addresses, it requires significant time and resources, and many hosts do not respond to measurement probes because they filter packets or they are off-line. Next, we describe the techniques we use to collect bandwidth data.

Step 1: As a first order filter, we use "speed test" results from a popular web site, broadbandreports.com. Speed tests are TCP-based bandwidth measurements that are conducted from well-provisioned servers owned by the web site. Users voluntarily go to the web site to test their connection speeds. Results from the tests are aggregated based on DNS domain names or ISPs and are summarized over the past week on a publicly available web page. Approximately 10,000 tests for over 200 ISPs are listed in the summary. The results include bandwidth measurements in both incoming and outgoing directions. We use only the outgoing bandwidth values for bandwidth estimation. We map the IP addresses in our streaming logs to their ISPs using DNS names, and then match the ISPs to the ones listed at broadbandreports.com. For the 72% of hosts that matched, we assign their bandwidth values to the ones reported by broadbandreports.com.

Step 2: Next, we aggregate the remaining IP addresses into /24 prefix blocks and conducted packet-pair measurements to measure the bottleneck bandwidth to several hosts in each block. Of the 13,483 prefix blocks probed, 7,463 responded. We use the measurement results from the prefix blocks to assign bandwidth estimates to an additional 7.6% of the IP addresses, for a total of 79.6% estimated so far. Note that packet-pair measures the average of the incoming and outgoing bandwidth (because it relies on round-trip time measurements). However, many access technologies have asymmetric bandwidth properties. For example, an ADSL host with an incoming link of 400 kbps and an outgoing link of 100 kbps has a packet-pair measurement of 250 kbps. Using the average of the two directions could over-estimate the outgoing link. Taking this into account, we map the bandwidth measurements from packet-pair into the access technology categories listed in Table 1, where BW stands for the measured bandwidth from using packet-pair. We use the values in the last column as the outgoing bandwidth estimates.

Step 3: We use EdgeScape, a commercial product provided by Akamai that maps IP addresses to host information. One of the fields in the host information database is access technology. We compared the overlapping matches from this database to the ones from broadbandreports.com and found them to be consistent. To translate access technology into raw bandwidth, we use the values in the "outgoing bandwidth estimate" column in Table 1. Using this technique, we are able to assign estimates to an additional 7.1% of the IP addresses, for a total of 86.7% estimated so far.

Step 4: Finally, we use the host's DNS name to infer its access technology. Our first heuristic is based on keywords such as *dsl, cable-modem, dial-up, wireless, .edu* and popular cable modem and DSL service providers such as *rr.com and attbi*. Using this technique, we get estimates for an additional 2.2% of IP addresses. As our second heuristic, we manually construct a database of small DSL and cable modem providers, corporations, and international educational institutions that do not use common keywords in their DNS names. This technique provides estimates for an additional 1.2% of IP addresses. Again, we translate access technology into bandwidth using the values in Table 1.

Using all 4 steps provides us with bandwidth estimates for 90% of the IP addresses in the traces. We discuss our treatment of the remaining 10% of hosts with unknown estimates later in this section.

3.1.2 Degree-Bound Assignment

To simplify the presentation, we normalize the bandwidth value by the encoding bit rate. For example, if a host has an outgoing link bandwidth of 300 kbps and the encoding rate of the stream is 250 kbps, then the normalized value is $\lfloor 300/250 \rfloor = 1\ degree$. Assuming a tree structure for the overlay, this host can have an out-degree of 1, i.e., it can support one child at the full encoding bit rate. Throughout this paper, we use "degree" instead of kbps as the outgoing bandwidth unit.

The degree assignment for the largest broadcast is listed in Table 2. Half of the hosts have 0-degree and are labeled as free-riders. Roughly 39% of the hosts are contributors, capable of supporting one or more children. Of these, 6.8% are hosts who are capable of supporting 20 or more children.

The degree assignment derived from the outgoing bandwidth value reflects the *inherent* capacity of a host. However, all that capacity may not be available for use. For example, the bandwidth may be shared by many applications on the same host, or may be shared across many end-hosts in the case of a shared access link. Also, users may not wish to contribute all of their bandwidth to the system. In this paper, we set an absolute maximum bound of 20 for the out-degree (degree cap of 20) such that no host in our simulations can exceed this limit even if they have more resources to contribute. This roughly translates to 5 Mbps for video applications. We also study the effect of more conservative policies such as degree caps of 4, 6, and 10.

3.1.3 Hosts With Unknown Measurements

For the 10% of IP addresses without bandwidth estimates, we assign an estimate to them using 3 assignment algorithms. The *optimistic* estimate assumes that all unknowns can contribute up to the maximum resource allocation (which we set to be degree 20 or less, depending on the degree cap). This provides an upper-bound on the best case resource assignment. The *pessimistic* estimate assumes that all unknowns are free-riders and contribute no resources. This provides a lower bound for the worst-case. The *distribution* algorithm assigns a random value drawn from the same distribution as the known resources. This algorithm provides reasonable estimates assuming that the known and unknown resources follow the same distribution.

3.2 Resource Index

To measure the resource capacity of the system, we use a metric called *Resource Index* [7]. The Resource Index is defined as the ratio of the supply of bandwidth to the demand for bandwidth in the system for a particular encoding bit rate. The supply is computed as the sum of all the degrees that the source and application end-

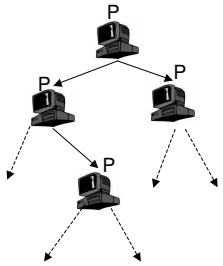


Figure 4: Example of how to compute the Resource Index.

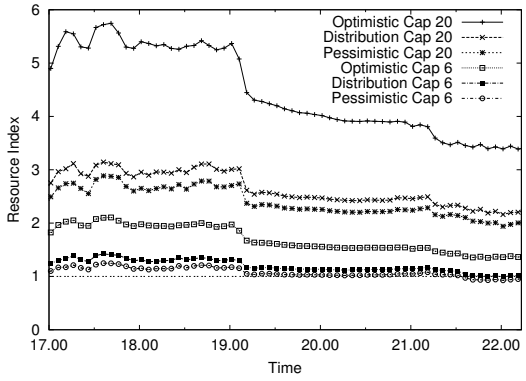


Figure 5: Resource Index for largest event.

points participating in the system contribute. Note that the degree is dependent on the encoding bit rate, and in turn the Resource Index is also dependent on the encoding bit rate. Demand is computed as the number of participating end-points. For example, consider Figure 4, where each host has enough outgoing bandwidth to sustain 2 children. The number of unused slots is 5, and the Resource Index is $(5+3)/3 = 8/3$. A Resource Index of 1 indicates that the system is fully saturated, and a ratio less than 1 indicates that not all the participating hosts in the broadcast can receive the full encoding rate. As the Resource Index gets higher, the environment becomes less constrained and it becomes more feasible to construct a good overlay tree. A Resource Index of 2 indicates that there are enough resources to support two times the current number of participants.

3.3 Trace Replay: Single-Tree Protocol

In this section, we use the Resource Index to measure the amount of resources across a set of 81 streams (all video streams and 5% of randomly selected audio streams) out of the 660 large-scale streams. To ensure some confidence in our results, at least 70% of the IP addresses in a trace must have bandwidth estimates in order for it to be analyzed.

For each stream, we replay the trace using the group participation dynamics (joins and leaves) and compute the Resource Index for each second in the trace. First, we discuss the results for the largest event, and then we present a summary of the results for the other large-scale events.

Figure 5 depicts the Resource Index as a function of time, with degree caps of 6 (bottom 3 lines) and 20 (top 3 lines) children. The time interval of interest is between 19:00 - 21:00 when the event was taking place. Again, note that a Resource Index above 1 means that there are sufficient resources to support the stream using an application end-point architecture. The highest and lowest curves for each degree cap policy are computed using optimistic and pessimistic bandwidth estimates for unknowns, respectively. Regardless of the treatment of hosts with unknown estimates and the degree cap policy, the Resource Index is always above 1 during 19:00 - 21:00. However, a degree cap of 6 places more constraints on the resources in the system and could potentially make it more difficult to construct a tree with good performance.

Figure 6 depicts a summary of the other large-scale streams. The Resource Index for audio streams is depicted in Figure 6(a). Each point on the x-axis represents an audio stream. The y-axis is the Resource Index for that stream averaged over the stream duration. The lowest curve in the figure is the Resource Index computed using the pessimistic bandwidth estimate for unknowns. For audio streams, even the pessimistic estimate is always between 2-3 when using a degree cap of 4. This is expected because audio is not a bandwidth-demanding application. The typical encoding rate for audio is 20 kbps which is low enough for most hosts on the Internet to support, including dial-up modems. Thus, application end-points participating in audio streaming applications can provide more than enough resources to support live streaming.

Figures 6 (b), (c), and (d) depict the average Resource Index for video streams with degree caps of 6, 10, and 20 children. As the degree cap increases, the Resource Index increases. The top most curve in all 3 figures represents the optimistic estimate for unknowns. In the most optimistic view, across all degree cap policies (6,10, and 20), only one stream had a Resource Index below one. In the worst case scenario, where the degree cap is 6 and the unknown assignment policy is pessimistic, roughly a third of video streams had Resource Index below 1.

In order to determine feasibility, we look at the *inherent* amount of resources in the system. Using a degree cap of 20 and the distribution-based degree assignment for unknowns as depicted in Figure 6(d), we find that only 1 stream has a Resource Index of less than 1. The stream with the worst Resource Index (labelled 40 on the x-axis) had an encoding rate of 300 kbps, but was composed almost exclusively (96%) of home broadband users (DSL and cable modem). Many home broadband connections can only support 100-250 kbps of outgoing bandwidth, which is less than the encoding bit rate. Therefore, such hosts did not contribute any resources to the system at all.

To better understand whether or not this composition of hosts is common, we look at the nature of the event. This is a short duration stream, starting on Sunday night at 11pm and ending at 2am in local time, where local time is determined based on the geographic location of the largest group of hosts. Section 5 gives an overview of how geographic location is determined. Most participants are home users because the event took place on a weekend night when people are most likely to be at home. About 5 of the 55 large-scale video streams have this behavior. Their Resource Index is close to 1 for the distribution-based degree assignment in Figure 6(d). In contrast, most of the other streams take place during the day, and are often accessed from more heterogeneous locations with potentially more bandwidth resources, such as from school or the workplace.

To summarize the results in this section, we find that there are more than sufficient bandwidth resources amongst application end-points to support audio streaming. In addition, there are enough *inherent resources* in the system at scales of 1000 or more simultaneous hosts to support video streaming in over 90% of the common scenarios. This indicates that using application end-point architectures for live streaming is feasible. While we have shown that there are *inherent resources*, we wish to point out that designing policies and mechanisms to encourage participants to contribute their resources is beyond the scope of this paper. We have looked at sim-

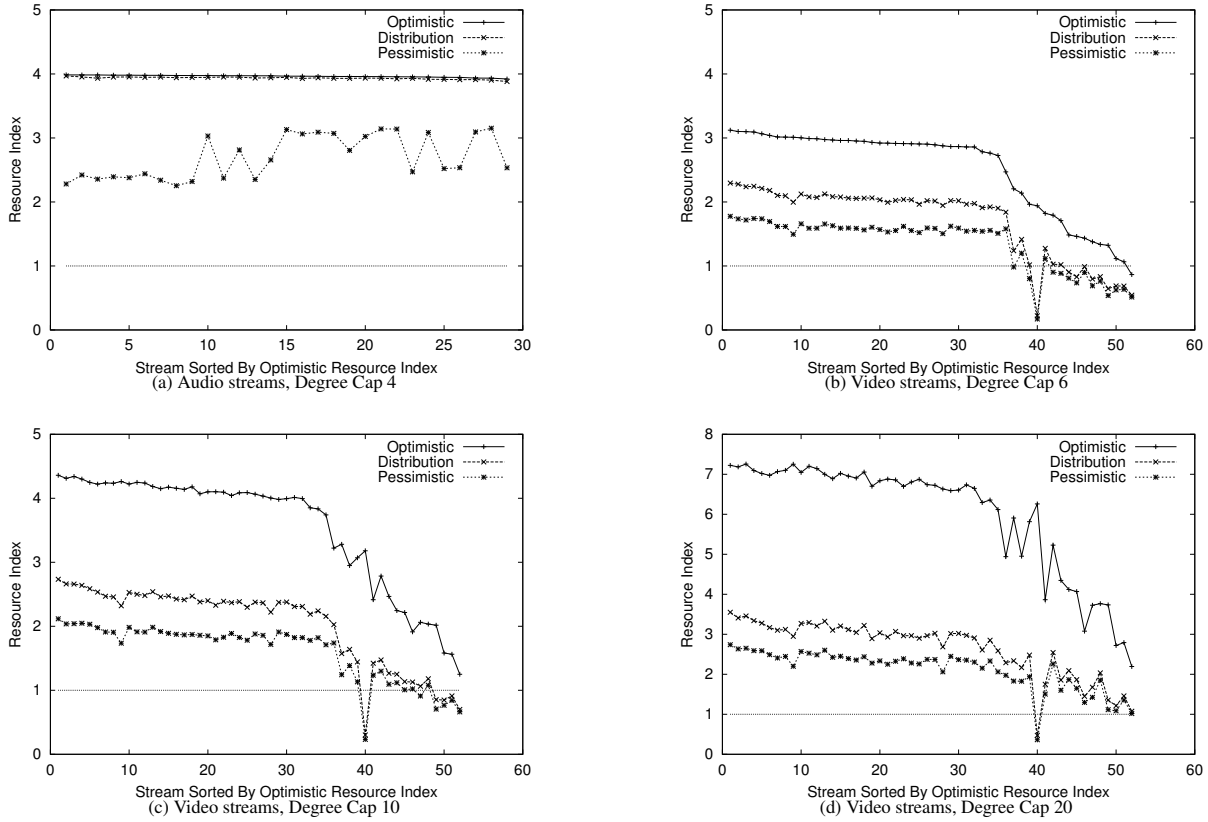


Figure 6: Average Resource Index for each stream.

ple policies, such as capping the degree bound to a static amount to make sure that no person contributes more than what is considered to be "reasonable." There could be more complex policies to allow users to individually determine how much they are willing to contribute in return for some level of performance [6].

While there are resources in the common scenarios, in 10% of the cases there were not enough resources (or close to not enough) to allow all participants to receive at the full encoding rate. In such scenarios, there are three generic classes of alternatives to consider. The first alternative is to enforce admission control and reject incoming free-riders when the Resource Index dips below one. A second alternative is to dynamically adapt the streaming bit-rate, either in the form of scalable coding [15], MDC [11], or multiple encoding rates [7]. This has the advantage that the system can still be fully supported using an application end-point architecture with a reduction in the perceived quality of the stream. Lastly, a third alternative is to add resources into the system. These resources can be statically allocated from infrastructure services such as content delivery networks (CDNs) with the advantage that the resources only need to *complement* the already existing resources provided by the application end-points. Another solution is to allocate resources in a more *on-demand* nature using a dynamic pool of resources, for example, using a waypoint architecture [7].

3.4 Impact of NATs and Firewalls

One factor that can negatively impact our results is the presence of participating hosts that have *connectivity restrictions* behind Network Address Translators (NATs) and firewalls. Such hosts cannot communicate with other connectivity restricted hosts and thus reduce the number of usable pair-wise overlay links. Recently, several

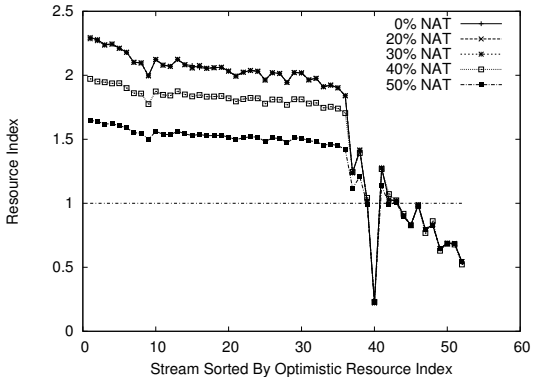


Figure 7: Resource Index for video streams when considering NATs.

solutions have been developed to allow hosts behind NATs and firewalls to participate in overlay multicast [10]. Such solutions enable all hosts except for hosts behind symmetric NATs [23] and certain firewalls to have universal connectivity. For the purpose of this paper, we assume that these solutions are implemented. Thus, our concern is with symmetric NATs and firewalls that still are connectivity restricted. Note that the connectivity semantics of a symmetric NAT is that it cannot communicate with other symmetric NATs. Firewalls, on the other hand, can have arbitrary connectivity semantics. For this paper, we assume that firewalls have the most restrictive semantics – the same as symmetric NATs in they cannot communicate with other symmetric NATs or firewalls. Thus, links between two

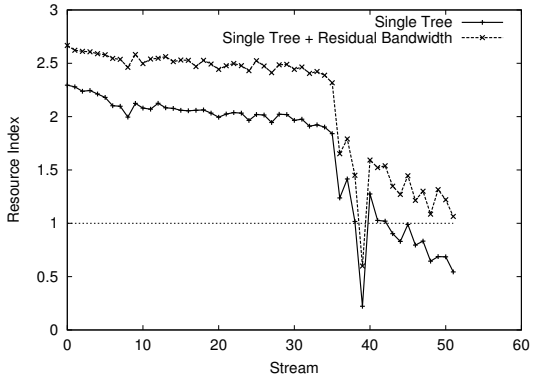


Figure 8: Resource Index for multiple trees.

different symmetric NATs or firewalls cannot be used in the overlay. To simplify the discussion, we will refer to both symmetric NATs and firewalls that have the same connectivity semantics as symmetric NATs for the rest of this section.

To understand how our results change in the presence of symmetric NATs, we consider the Resource Index as a function of the percentage of symmetric NATs in the system. We refer the readers to [10] for the details on how to implement the optimizations and compute the Resource Index with connectivity restrictions.

Figure 7 depicts the Resource Index for the same video streams as those depicted in Figure 6(b) with a degree cap of 6 and a distribution-based estimate of unknowns. Each line represents a scenario where there are 0% to 50% of restricted hosts (symmetric NATs) in the system. There is little or no difference between the 0%, 20% and 30% cases (the curves overlap). We start to see a drop in the Resource Index when more than 40-50% of the hosts are connectivity restricted. However, the drop is not that significant as the Resource Index is still above 1 for 67% of the streams even when half of the hosts are connectivity restricted. This is similar to when there are no connectivity restricted hosts in the system (Figure 6(b)). Furthermore, from operational experience with End System Multicast [10], the percentage of symmetric NATs is usually much lower (10%-30%). In such regimes, the Resource Index is the same as when there are no symmetric NATs given that all NAT-based optimizations are implemented in the protocol.

To summarize, for the large-scale streams in our traces, the presence of 40% or more connectivity restricted hosts in the system reduces the Resource Index. However, such reductions are not significant enough to make the Resource Index drop below one unless it was already below one without connectivity restricted hosts in the system. In addition, for realistic percentages of NATs (10%-30%), the Resource Index is unchanged compared to when there are no NATs in the system.

3.5 Multiple-Tree Protocols

In the previous section, we analyzed the amount of resources for single-tree protocols. More recently, multiple-tree protocols [4, 18, 13] have been proposed to increase the overall resilience of the system. Such protocols are tightly coupled with specialized video encodings, such as multiple description coding (MDC). The video stream is encoded into k independent descriptions (or sub-streams) and distributed across k independent trees.

The implication of multiple trees and MDC on resources is that the amount of resources in the system may increase as the residual bandwidth that was previously unused in the single-tree protocol may now be used. For example, if a host has an outgoing bandwidth of 300 kbps, and the stream is encoded at 250 kbps for a single tree, then the host has a residual bandwidth of 50 kbps that is unused. On the other hand, if the stream is encoded using MDC into many descriptions each at 50 kbps, then the host can contribute all of its outgoing bandwidth to the system to transmit up to 6 descriptions.

Overall, the use of MDC and multiple trees should always result in an increase in the supply of resources compared to a single tree. To quantify the increase, we modify the Resource Index computation as follows. We allow fractional supply (where the fraction corresponds to the residual bandwidth) to be used. For example, the supply for the host in the previous example is computed as $300/250 = 1.2$. We assume the demand remains the same as in the single-tree case – this is simplistic in that we are assuming no overhead and no redundancy in the encoding. A host needs to collect at least 5 descriptions in this example ($5x50\ kbps = 250\ kbps$), to have good quality video. The intuition behind this is that a stream that is originally encoded at 250 kbps, say a tennis match, is jerky and not watch-able at 50 kbps, or even at 200 kbps. If it were perfectly watch-able, then the stream would have already been encoded at the lower rate for the single-tree protocol.

Figure 8 depicts the Resource Index for the multiple-tree protocol for the same video streams presented earlier in Figure 6(b). Also depicted is the Resource Index for the single-tree protocol. The configuration shown here is for degree cap 6 (or the equivalent in kbps) and the distribution-based assignment for unknowns. To have sufficient resources, a Resource Index higher than 1 is needed for both the single-tree and multiple-tree protocol. We find that for the streams that had sufficient resources using a single-tree protocol, using a multiple-tree protocol can increase the bandwidth resources up to 20-25%. More interestingly, for the remaining streams that did not have sufficient resources using a single-tree protocol, the Resource Index increases from below 1 to above 1 for all but one stream when using a multiple-tree protocol. The value of the Resource Index determines how much encoding overhead the system can support. For example, a Resource Index of 1.1 means that 10% of overhead may be added.

To summarize, using multiple trees and MDC can increase the amount of resources in the system. With a degree cap of 6, 1/3 of the streams had a Resource Index of below 1 using a single-tree protocol. Using a multiple tree protocol, in all but one case, the Resource Index is above 1. Thus, multiple-tree protocols increase the feasibility of overlay multicast especially for those streams that do not have abundant resources.

3.6 Resources Summary

Our results indicate promise for application end-point architectures. Using a single-tree protocol and a single encoding rate, all audio streams have abundant resources and most video streams have enough inherent resources. With realistic percentages of NATs and firewalls in the system, the resource characteristics is the same as if there were no NATs and firewalls. Lastly, in resource constrained environments, using multiple-tree protocols can increase the supply of resources in the system and improve the situation.

4. IS THERE ANY STABILITY?

In this section, we look at feasibility of maintaining a stable and connected tree in the presence of group dynamics. In addition, we evaluate mechanisms that can be used to increase the stability of the overlay.

4.1 Extreme Group Dynamics

Figure 9 depicts the session duration characteristics for the 660 large-scale streams. The x-axis is the session duration in minutes.

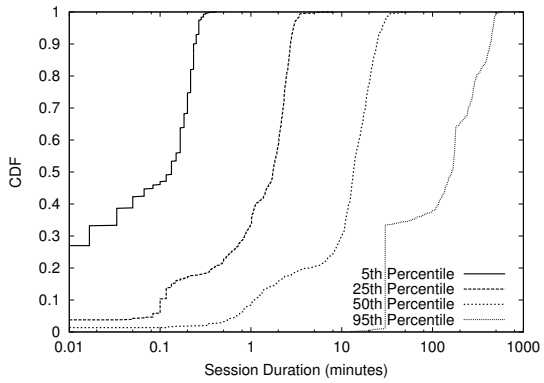


Figure 9: Incarnation session duration in minutes.

The y-axis is the cumulative distribution of the number of streams. The first curve on the left depicts the cumulative distribution of the observed 5th percentile session duration in all 660 streams. For example, 5% of incarnations in 30% of the streams had durations of shorter than 1 second. The next 3 curves are for the 25th, 50th and 95th percentile. Note that the same value on the y-axis does not necessarily correspond to the same stream on all the curves.

Based on this figure, we make two observations. First, there is a significant number of very short sessions. Looking at the 25th percentile curve, we find that for most streams, 25% of sessions are under 2 minutes. Furthermore, the most disastrous is that in the 50th percentile curve, 20% of the streams have extremely dynamic group membership with half of the sessions shorter than 5 minutes. With such short session durations, it seems very unlikely that there could be any stability.

Our second observation is that there is a heavy tail, where a small number of incarnations have very long sessions. The 95th percentile curve in Figure 9 shows that for most streams, the 95th percentile session duration is longer than 30 minutes. Perhaps the tail can help add some stability to the tree. Note that these observations are consistent with the session duration analysis of the largest event in Section 2.

4.2 Stability Metrics

When an incarnation leaves, it causes all of its descendants to become disconnected from the overlay and stop receiving data. Disconnects are perceived as glitches in the stream, resulting in poor performance. Disconnected descendants will need to find new parents and reconnect to continue receiving the stream. Therefore, a desirable tree is one in which when an incarnation leaves, no one is affected. Incarnations that will stay for a long time should be at the top of the tree, and incarnations that will stay for short durations should be leaf nodes at the bottom of the tree. To capture stability of the overlay we look at two metrics:

•**Mean interval between ancestor change for each incarnation.** This metric captures the typical performance of each incarnation. An ancestor change is caused by an ancestor leaving the group, typically resulting in a glitch in the stream. Frequent glitches may be annoying. Therefore, the longer the interval, the better the performance. If a host sees only one ancestor change during its session, the time between ancestor change is computed as its session duration. If a host sees no ancestor changes at all, the time between ancestor change is infinite.

•**Number of descendants of a departing incarnation.** This metric captures overall stability of the system. If many hosts are affected by one host leaving, then the overall stability of the system is poor. However, assuming a balanced tree, most hosts will be leaf nodes and will not have children. Therefore, we hope to see that a large percentage of hosts will not have children when they leave.

4.3 Overlay Protocol

We simulate the effect of group dynamics on the overlay protocol using a trace-driven event-based simulator. The simulator takes the group dynamics trace from the real event and the degree assignments based on the techniques in the previous section, and simulates the overlay tree at each instant in time. Hosts in the simulator run a fully distributed self-organizing protocol to build a single connected tree rooted at the source. The protocol is a simplified version of the one used in the End System Multicast project [7]. Note that we do not simulate any network dynamics or adaptation to network dynamics. The following protocol functions of the simplified protocol are also common across many of the existing overlay multicast protocols.

Host Join: When a host joins, it contacts the source to get a random list of m current group members. In our simulations, m is set to 100. It then picks one of these members as its parent using the parent selection algorithm described below.

Host Leave: When a host leaves, all of its descendants are disconnected from the overlay tree. For each of its descendants, this is counted as an ancestor change. Descendants then connect back to the tree by independently finding a new parent using the parent selection algorithm. Note that we prioritize reconnections by allowing descendants that contribute resources to connect back first, before free-riders. This prevents free-riders from saturating the tree before all descendants are able to reconnect. This is implemented by having hosts that contribute fewer resources wait longer before trying to reconnect.

Parent Selection: When a host needs to find a parent, it selects a set of m random hosts that are currently in the system, probes them to see if they are currently connected to the tree and have enough resources to support a new incoming child, and then ranks them based on the parent selection criteria described in the next section. In our simulations, m is set to 100. We do not simulate the mechanisms for learning about hosts currently participating in the system, but assume that such mechanisms provide random knowledge. In a real implementation, Gossip-based mechanisms [22] may be used.

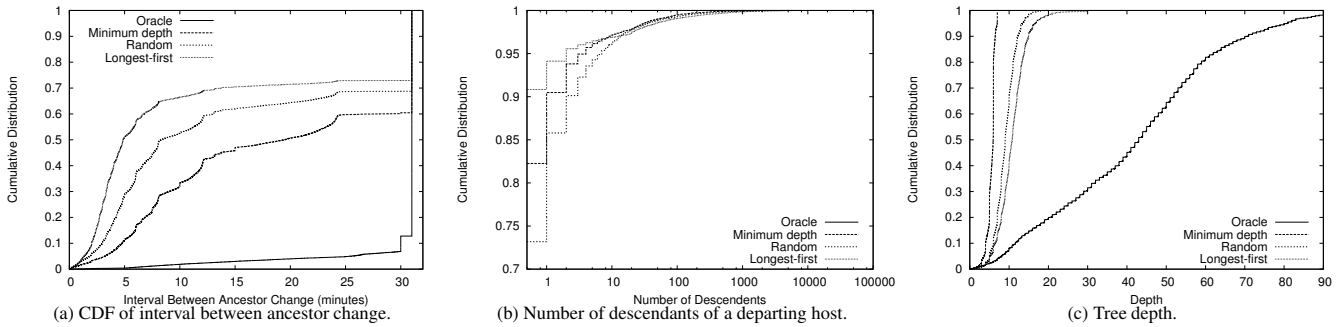
4.4 Parent Selection Algorithms

The parent selection algorithm determines the stability of the overlay. If hosts have stable parents, as opposed to unstable parents, then the tree is likely to be more stable. We ran simulations on 4 parent selection algorithms. Note that the chosen parent needs to be connected to the tree and have enough resources to support an incoming child (has not saturated its degree-bound), in addition to satisfying the parent selection criteria.

•*Oracle:* A host chooses the parent who will stay in the system longer than itself. If no host will stay longer, it chooses the host that will stay the longest. This algorithm requires future knowledge and cannot be implemented in practice. However, it provides a good baseline comparison for the other algorithms.

•*Longest-first:* This algorithm attempts to predict the future and guess which nodes are stable by using the heuristic that if a host has stayed in the system for a long time, it will continue to stay for a long time. The intuition is that if the session duration distributions are heavy-tailed, then this heuristic should be a reasonable predictor for identifying the stable nodes.

•*Minimum depth:* A host chooses the parent with the minimum depth. If there is a tie, a random parent is selected from the ones with the minimum depth. The intuition is that balanced shallow trees minimize the number of affected descendants when an ances-



(a) CDF of interval between ancestor change. (b) Number of descendants of a departing host. (c) Tree depth.

Figure 10: Stability performance of largest event.

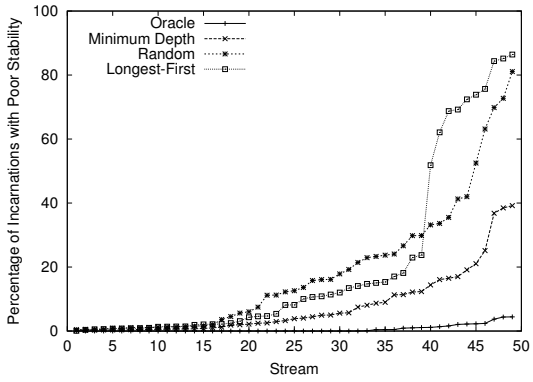


Figure 11: Stability performance for 50 large-scale streams.

tor higher up at the top of the tree leaves.

•*Random:* A host chooses a random parent. This algorithm provide a baseline for how a "stability-agnostic" algorithm would perform. Intuitively, random should perform the worst compared to the above algorithms.

We used the degree assignment from the previous section, with a degree cap of 4 for audio streams and 20 for video streams, and the distribution-based assignment for the hosts with unknown measurements. Unless otherwise stated we use this same set up for all subsequent simulations.

4.5 Results: Single-Tree Protocol

We simulated the performance of the 4 parent selection algorithms for the largest event over the most dynamic 30-minute segment from 19:00-19:30. We did not use the sessions with NAT/firewall timeout problems discussed in Section 2 in the simulations because their 2-minute session durations are artificial. Each parent selection algorithm is simulated 10 times, each time using a different random seed.

The cumulative distribution (CDF) of the mean time interval between ancestor change is depicted in Figure 10(a). The x-axis is time in minutes. A larger interval is more desirable. Because we are simulating a 30-minute trace, the maximum time interval is 30 minutes, if a host sees one ancestor change. Hosts that do not see any ancestor changes have an infinite interval. For presentation purposes, we represent infinity as 31 on the x-axis. The bottom most line in the figure represents the CDF when using the oracle algorithm. Roughly 10% of the incarnations saw only one ancestor change in 30 minutes. Furthermore, 87% of incarnations did not see any changes at all. In fact, there were only one or two events that caused ancestor changes across all the runs. It is surprising that there is stability in the system during the busiest 30 minutes in the trace. In addition, the overlay built by the oracle algorithm can exploit that *inherent stability.*

The second-best algorithm is minimum depth. Over 90% of the incarnations saw either no changes or 5 or more minutes between changes. This should be tolerable to humans as they will see a glitch every 5 minutes or so. The random algorithm and the longest-first algorithm performed poorly in this metric. For random, only 70% of the incarnations saw no changes or 5 or more minutes between changes. To our surprise, the longest-first algorithm performed the worst, with 50% of incarnations seeing decent performance. The reason that it did not perform well stems from several factors. While it correctly predicted stable nodes in 91% of the cases, it was wrong for the remaining 9% as depicted in Figure 10(b). The number of descendants of a departing host is on the x-axis, in log scale. The y-axis is the cumulative percentage of departing hosts. If longest-first were always correct, it would overlap with the y-axis, like oracle where almost all departing hosts had no descendants. One of the difficulties in getting accurate predictions is that at the start of the event, almost all hosts will appear to have been in the group for the same amount of time making stable hosts indistinguishable from dynamic hosts. Note that longest-first is predicting correctly for more cases than random and minimum depth, which had 72% and 82% of incarnations with no descendants when they left the system. To explain the poor performance, we look at the second factor.

The second factor is that the consequence of its mistake is severe as depicted in Figure 10(c). The x-axis is the average depth of each node in the tree. The longest-first algorithm has taller trees than the random and minimum depth algorithms. Therefore, when it guesses incorrectly, a large number of descendants are affected. We examined the tail end of Figure 10(b) more closely and confirmed that this was the case.

One interesting observation is that the oracle algorithm builds the tallest tree. The intuition here is that nodes that are stable will cluster together and "stable" branches will emerge. More nodes will cluster under these branches, making the tree taller. Although the height does not affect the stability results in our simulations, in practice a tall tree is more likely to suffer from problems with network dynamics.

We find that minimum depth is the most effective and robust algorithm to enforce stability. Its property of minimizing damage seems to be the key to its good performance. The fact that it does not attempt to predict node stability makes it more robust to a variety of scenarios, as depicted in Figure 11. We ran the same set of simulations using 4 parent selection algorithms for 50 of the large-scale streams. These are the same streams as the ones presented in Section 3, but with only half of the video streams present. Again, we used the distribution-based assignment for hosts with unknown

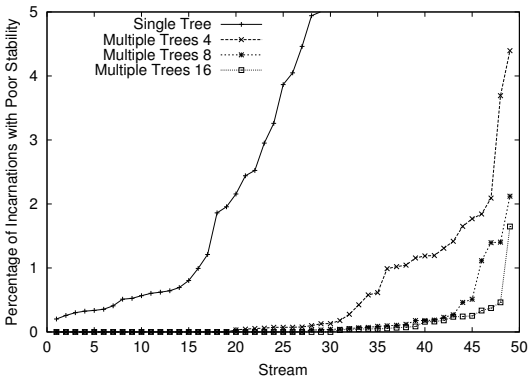


Figure 12: Stability performance for multiple trees.

measurements, a degree cap of 20 for video streams, and a degree cap of 4 for audio streams. The simulations were run over the most dynamic 1-hour segments in each trace. We assume a conservative cut-off for poor performance: an incarnation that sees an ancestor change more frequently than once in 5 minutes is seeing poor performance. The x-axis is the stream and the y-axis is the percentage of incarnations that see poor stability performance in that stream. Again, the oracle algorithm performed the best with most streams having no incarnations with poor performance. Minimum depth performed the second best with 45 out of the 50 streams having 20% or less incarnations with poor stability performance. Random and longest-first both performed poorly with 5-10 streams where 50% or more incarnations see poor performance.

While we present results based on 4 parent selection algorithms, we also explored many design alternatives. For example, we looked at prioritizing contributors and combining multiple algorithms. However, we do not present them in this paper due to space limitations. We note that alternate algorithms did not perform as well as the ones listed above. For example, when the parent selection algorithm prioritized contributors such that they are higher up in the tree, the performance was as poor as random. This is explained by the observation that there is no correlation between being a contributor and being stable.

We also looked at the impact of resource on stability. In particular, we looked at whether there is more stability if there are more high degree nodes (i.e., more resources). We ran simulations on the audio streams with degree caps of 6 and 10, and found that there was only a slight improvement compared to when the degree cap was 4.

To summarize, we find that there is *inherent stability* in application end-point architectures. Without future knowledge, there exists practical and simple algorithms such as minimum depth that can provide good stability performance. While we have looked at how to reduce the number of ancestor changes in the system, another important direction is to reduce the perceived impact of an ancestor change. In particular, mechanisms such as maintaining application-level buffers to smooth out interruptions could help while the affected descendants are looking for new parents. Multiple-tree protocols, which we discuss next, may also help reduce the impact of ancestor changes.

4.6 Impact of Multiple-Tree Protocols

In multiple-tree protocols, the stream is split and distributed across k independent trees. The probability of many trees seeing simultaneous disruptions is small. In addition, with sufficient redundancy in the encoding, the impact of a disruption in one tree may be negligible. On the other hand, because a host now has k times more ancestors, it is likely that it would see a larger number of ancestor changes overall. Although frequent ancestor changes may not always affect perceived quality, it creates more protocol overhead and activity on the network because hosts need to find new parents more frequently.

To explore the effect of multiple-tree protocols on stability, we simulate the same 50 streams as those depicted in Figure 11 using a multiple-tree protocol. There are three modifications to the single-tree protocol in Section 4.3. Except for the changes below, each tree is independently constructed using the single-tree protocol.

Independent trees: To maintain independence between trees, each host is an interior node (contributor of resources) in only one tree [4]. A host selects the tree that it will contribute its resources, and joins that tree as a contributor. It joins the remaining trees as leaf nodes. Thus, when it leaves the broadcast, it will only affect the stability of one tree because it has descendants in only one tree.

Load balancing: We implement load balancing of resources among trees such that all trees have roughly the same amount of resources. The load balancing algorithm is run at join time, where a host will become a contributor (interior node) for the tree which currently has the lowest Resource Index. The source keeps track of the Resource Index by maintaining a count of the amount of resources in each tree and the current number of incarnations in the system.

Preemption: There may be cases where a tree may be saturated and not have enough resources. If a new host were to join the tree, it would not be able to. To allow new contributors to join, they may preempt existing "free-riders" in the tree. Preemption involves disconnecting the free-rider from the tree to open up a position for the new incoming contributor. The contributor takes the free position and may accept the free-rider that was preempted to join under it as its child. We implement a limited form of preemption where a new contributor only preempts free-riders at depth 1 (i.e., children of the source) and found this to be sufficient for the workloads in our study. Overall, preemption rarely take place. Even for the stream with the most preemption, preemption caused only 4% of disconnects compared to the 96% caused by departing hosts.

Generally, MDC encoding adds redundancy and overhead compared to the original stream. In our simulations, we assume that the overhead is 25%. In practice, this overhead depends on the specific video stream and the MDC optimization. Setting this number too low could result in poor resilience; setting this number too high wastes bandwidth resources. We run the simulations using three configurations: 4, 8, and 16 trees. Each tree carries a fraction of the source rate. For example, in the 4-tree configuration, each tree carries 1/4 of the original source rate with 25% redundancy and overhead. With 25% redundancy, receiving 3 out of 4 descriptions is sufficient. Only results for the minimum depth parent selection algorithm are presented. Note that minimum depth performed the best amongst all the practical algorithms evaluated for the single-tree protocol.

First, we look at the percentage of incarnations that see frequent ancestor changes (have an average interval between ancestor change shorter than 5 minutes, similar to the single-tree case). Because an incarnation has multiple simultaneous parents, one in each tree, it is likely to see more frequent ancestor changes as more trees are used. Simulation results confirm this intuition. We find that with 16 trees, across the 50 streams, on average 75% of incarnations in the streams see too frequent ancestor changes. With 8, 4, and single-tree, the percentage drops to 55%, 32%, and 8% respectively. While this indicates that protocol overhead increases significantly with multiple trees, it does not indicate the perceived quality of the streams.

Next, to understand perceived quality, we look at the average interval between *too many* simultaneous disconnects. As previously

mentioned, being disconnected from one tree does not impact the quality of the stream. However, being disconnected from *too many* trees simultaneously, or over 25% of the trees in our configuration, indicates poor perceived quality. We assume that when a host is disconnected, it takes one second for it to find a new parent and connect back to the tree. The y-axis in Figure 12 depicts the percentage of incarnations that see too many disconnects, defined as more often than once in 5 minutes. The x-axis is the 50 large-scale streams – the same as in the single-tree analysis in the previous section. The y-axis is truncated at 5% to better illustrate the differences between the different configurations. In addition, the previous results from the single-tree minimum depth protocol are also depicted. The percentage of incarnations with poor stability is higher for the single-tree protocol. For the multiple-tree protocol, using 4 trees, all streams have less than 5% of incarnations with poor performance. Fewer incarnations see poor performance as more trees are used. For example, using 16 trees, all streams have less than 2% of incarnations with poor stability performance.

In this section, we see that multiple trees can increase the perceived quality of the streams. However, the improved performance comes at a cost of more frequent disconnects, more protocol overhead, and more complex protocols.

5. CAN EFFICIENT OVERLAYS BE CONSTRUCTED?

In this section, we look at the feasibility of constructing efficient large-scale overlays. An efficient overlay is one in which the overlay structure closely reflects the underlying IP network. The challenge is to enable hosts to discover other *nearby* hosts that may be used as parents. When there are as many as 70,000 other hosts simultaneously participating, it is not possible for a host to know *everyone else* because it would require significant protocol overhead to maintain such knowledge. As a result, each host will only know a subset of the current membership. In order to construct efficient overlays, that subset must contain hosts that are nearby.

We develop and analyze techniques for partitioning application end-points into clusters. One member of each cluster is designated as the cluster head (also called *membership server*). Hosts in the same cluster maintain knowledge about one another. Clustering policies that leverage network proximity have the potential to increase the efficiency of the overlay structure.

5.1 Membership Management

Next, we describe the clustering-based membership management protocol hosts use to maintain and distribute group membership information. We wish to highlight that the simplicity of the protocol allows for simple recovery given the dynamic arriving and departing nature of the membership servers.

Handling host join: A new host joining the system contacts a rendezvous point, often the source of the broadcast who is responsible for knowing the current membership servers participating in the broadcast. The rendezvous point responds with a list of current membership servers. The new joining host then selects one of the membership servers to contact, either randomly or by using clustering techniques discussed in the next section. The selected membership server replies with a fresh list of current members that it knows (mostly inside the same cluster). The joining host then uses the list in the tree construction protocol to connect itself to the tree via a member in the list.

Creating membership servers: The rendezvous point is responsible for ensuring that there are enough membership servers in the system. Membership servers are created *on-demand* based on the needs of the system. For example, when a new host arrives and there are not enough membership servers in the system, the rendezvous point will immediately assign the new host to function as a membership server (assuming the new host has enough resources to support the control traffic).

Recovering from membership server dynamics: Because we are using application end-points as membership servers, we must cope with membership servers leaving. Just before leaving, a membership server looks to see if it can promote one of the hosts inside its own cluster to become the new membership server. It is possible that a promotion may not be possible because of resource constraints. In such cases, the rendezvous server will notice that the number of membership servers has decreased and will create a new membership server from the newly arriving hosts. Note that when a membership server leaves, it does not affect data delivery except for the hosts that are its own direct descendants. The existing hosts that were part of the departing membership server's cluster need to find a new membership server. If a promotion was successful, the newly promoted host becomes their replacement membership server. The membership state can be quickly refreshed as hosts can send explicit liveness updates to the replacement membership server. If a promotion was not successful, hosts will move to different membership servers.

State maintenance: In order to recover from membership servers departing the broadcast dynamically, all membership servers explicitly exchange state about their liveness with the rendezvous point. Membership servers also maintain explicit state, liveness, and information about other membership servers and a random subset of members outside their cluster. In addition, all hosts inside a cluster exchange explicit state and maintain keep-alives with their membership server. When keep-alive messages are received at the membership server, the membership server will respond with a list of a subset of other live membership servers in the system and other members outside its cluster (learned from exchanges with other membership servers). Knowing hosts outside one's own cluster helps with recovery. Hosts also exchange their group membership knowledge with other hosts that they know. Gossip-like protocols [22] may be used, with a stronger bias towards exchanging information with hosts inside their own cluster.

Interplay between membership management and tree construction: The data delivery structure and the membership management clusters are loosely coupled. The membership information from the clusters implicitly influences the data delivery tree. We do not enforce strict clustering on the data delivery tree. In fact overlay nodes are free to select nodes outside their own cluster as parents if those nodes provide better performance. This simplifies performance optimizations and recovery from node failures.

5.2 Clustering Policies

In this section, we discuss the design of clustering policies. We consider three different clustering policies. Our first policy is *random*, where the clusters are agnostic of network proximity. Our second policy is network *delay-based clustering*. Short delays are reasonably correlated with good bandwidth performance [17], which is also important for streaming applications. And lastly, we look at *geographic clustering*, which roughly approximates network distance. For example, the network delay between hosts in the same continent is likely to be shorter than hosts in two different continents.

We implement the clustering policies by having hosts join the cluster belonging to the membership server "closest" to them. For random clustering, hosts pick a cluster to join at random. We call these three policies *naive clustering*.

In addition to considering proximity, we need to consider two critical requirements: ensuring that cluster sizes are not too large and ensuring that each cluster has enough resources. Bounding the

cluster size helps to prevent membership servers from being overloaded. Being aware of resources helps to ensure that there are enough resources within a cluster such that hosts can use other hosts inside their own cluster as parents. While hosts may still use hosts outside their cluster as parents, this degrades the efficiency of the overlay. Hosts in different clusters are likely to be farther away than hosts inside the same cluster.

Ignoring these two requirements results in poor clusters. For example, we analyze the largest event and find that using naive random clustering, all clusters have sizes close to 200 hosts. However, naive delay-based and geographic clustering both produce clusters with a wide range of sizes (from 10's to 1000's). In addition, the Resource Index for 7% of the random clusters, and 20% of the delay-based and geographic clusters are below 1. For example, delay-based clustering produces a few huge clusters with low Resource Index, each comprising almost exclusively of DSL and cable modem hosts that belong to the same ISP.

We use the following algorithms to meet the two additional requirements of maintaining cluster sizes and resources.

Cluster Size Maintenance: Two possibilities for bounding the cluster size and handling overflows are: (i) *redirection*, where new hosts are redirected to the next best cluster until an available one is found and (ii) *new cluster creation*, where a new contributor host that is supposed to join a full cluster creates a new cluster.

Resource Maintenance: We redirect free-riders joining a cluster with Resource Index at or below 1 to other clusters, but we allow contributors to join because they either increase or maintain the Resource Index.

5.3 Clustering Quality

In this section, we evaluate the quality of the clustering produced by the various policies and design choices. First, we discuss how we obtained the proximity data used in the evaluation.

5.3.1 Proximity Data

Network delay: In order to evaluate efficiency, we need to know the pair-wise delay between all participating hosts. This is infeasible without access to the hosts themselves. Instead, we approximate pair-wise delay values using Global Network Positioning (GNP) [16]. We assign coordinates to each of the hosts and "compute" the delay based on the geometric distance. To assign coordinates, we use 13 landmarks (PlanetLab [19] machines) located around the world. Landmarks measure the round-trip time between themselves and the IP addresses in our streaming workload, and then compute coordinates based on the measurement data (using 8 dimensions). Due to the overhead of probing and the low response rate, we probed only the IP addresses in the largest stream in our traces. Of the 118,921 IP addresses, only 27,305 responded. Hosts that did not respond are not used in our simulations.

Geographic distance: We obtain location information from Akamai's EdgeScape service, the same service that provided access technology information in Section 3. Using EdgeScape, we map an IP address to its latitude and longitude. Manual verification of mapping results with known geographic locations showed that the information was accurate for our purpose, which is coarse-grain geographic clustering.

For all of the following analysis and simulations, we use a degree cap of 20 and the distribution algorithm for assignment of unknowns, similar to the setup in Section 4. We assume that each membership server contributes one "degree" out of its existing resources for the join protocol overhead.

5.3.2 Clustering Quality Metric

To capture clustering quality, we use the average and maximum intra-cluster distance in milliseconds as the metric. Average intra-cluster distance measures the overall "tightness" of the clustering. The smaller the value, the closer all hosts in the cluster are to each other. Maximum intra-cluster distance measures the worst-case "spread" of the cluster. Again, we would like to see a small distance. The distance metric we use here is the network distance (approximated using GNP) and the following analysis is conducted for the largest event only for the hosts with GNP coordinates.

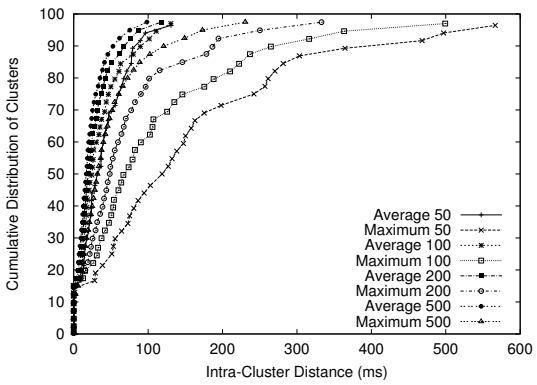


Figure 13: Clustering quality when varying number of clusters.

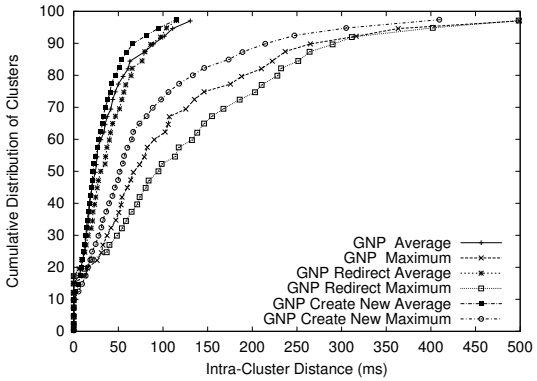


Figure 14: Clustering quality when bounding cluster sizes.

5.3.3 Sensitivity to the Number of Clusters

The rendezvous point needs to maintain a minimum number of clusters. To understand what is a good number, we look at the intra-cluster distance as a function of the number of clusters when using naive delay-based clustering. Figure 13 plots the cumulative distribution of the intra-cluster distances for all clusters created in the simulation, where the minimum number of clusters is varied between 50 and 500, and the maximum cluster size is maintained at 200. Using more clusters results in smaller intra-cluster distance for each cluster. The average (the lines towards the left) improves only slightly, however the maximum improves significantly from close to 600 ms for 50 clusters to about 250 ms for 500 clusters. All remaining simulations use a minimum of 100 clusters and a maximum cluster size of 200 hosts.

5.3.4 Sensitivity to Cluster Size and Resource Maintenance

Figure 14 depicts the cumulative distribution of intra-cluster distances for naive delay-based clustering (which we also refer to as GNP clustering), and the two techniques used to bound cluster sizes: GNP with redirection, and GNP with new cluster creation.

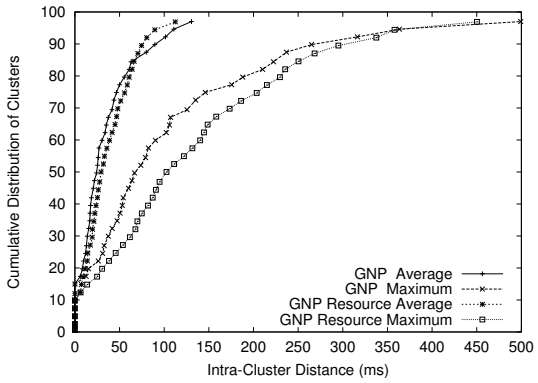


Figure 15: Clustering quality when bounding Resource Index.

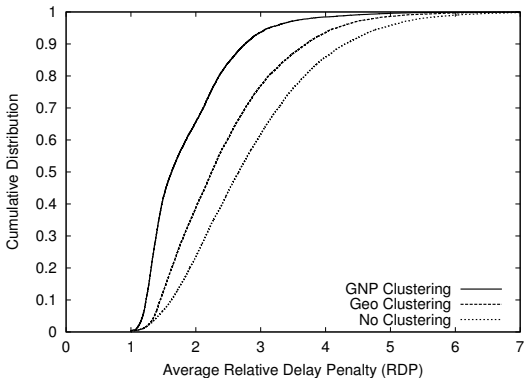


Figure 16: Overlay efficiency.

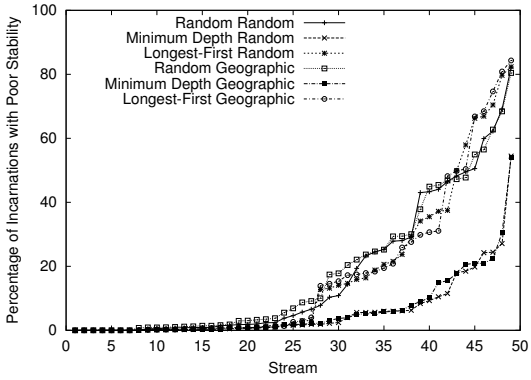


Figure 17: Stability performance with clustering.

For 90% of the clusters the average cluster distance is within 100 ms and the maximum reaches 300 ms. This figure shows that bounding the cluster size does not significantly affect the intra-cluster distances. Redirecting hosts from large clusters does not affect clustering quality because if the cluster is large, that indicates that there are other clusters nearby (created from previous redirections). Similarly, using a new contributor to create a new cluster also works well because more hosts that are nearby will subsequently join the new cluster.

Figure 15 depicts the cumulative distribution of intra-cluster distances for naive GNP and GNP with Resource Index maintenance. Again, average and maximum intra-cluster distances are not significantly affected.

5.3.5 Sensitivity to Cluster Head Choice

We compared the intra-cluster distances from our results above to distances resulting from clustering using the k-means algorithm [3]. The key difference is that k-means will choose an optimal cluster center based on neighboring coordinates while our proposed mechanism chooses cluster heads randomly. Note that the results for k-means clustering assume that all hosts are present in the system at the same time. Although a direct comparison between the two is not possible, it it is still useful to know whether or not the quality of the clusterings are similar. Using k-means, 90% of the clusters had an average intra-cluster distance less than 150 ms, indicating that the clustering algorithms that we use have similar quality to the more theoretically motivated clustering using k-means. Choosing optimal cluster heads is not a critical problem.

To summarize, clustering quality is not sensitive to the optimizations to maintain cluster sizes and available resources inside a cluster. In addition, choosing the optimal cluster head is not critical to the clustering quality.

5.4 Overlay Efficiency and Performance

In this section, we evaluate the efficiency of the overlay structure when using the join protocol, enhanced with clustering as described in the previous sections. To measure efficiency, we use the relative delay penalty (RDP) [8]. RDP is defined as the ratio of the delay between the source to the host along the overlay to the direct unicast distance between the source and the host. If the RDP is close to 1, then the overlay is very efficient and closely reflects the underlying unicast paths. The larger the RDP is, the more inefficient the overlay structure. Note that the location of the source is not provided in the logs. Typically, a content provider is generating live streams on one of their own servers and forwarding the streams into the Akamai network. However, the logs collected at Akamai's edge nodes do not reflect where the content provider's server is located. For the evaluation, we place the source at a location that coincides with a randomly chosen host participating in the broadcast. Again, we use the same configuration as in the last section with a minimum of 100 clusters and a maximum cluster size of 200 hosts.

5.4.1 Efficiency of Large-Scale Overlays

First, we present efficiency results for the largest event in our trace, using only the hosts for which we had GNP coordinates. Figure 16 depicts the cumulative distribution of the average RDP for each incarnation for GNP and random clustering (while maintaining the Resource Index). We use minimum depth as the parent selection algorithm for tree construction. GNP clustering produces more efficient trees, as the RDP is less than 2 for 65% of the hosts–the penalty for using the overlay is only twice that of the direct unicast path from the source. Our RDP values for large-scale groups are similar to previously reported values for much smaller-scale Internet testbed studies (13 hosts) using synthetic workloads [8].

In comparison to GNP clustering, only 35% and 25% of the incarnations have an RDP of less than 2 for geographic and random clustering. Geographic clustering does not perform as well as delay-based clustering because geographic distance may not always correlate with network distance.

5.4.2 Impact of Clustering on Stability

Next, we ask whether or not efficient overlay structures are stable. Clustering may affect tree stability if hosts within different clusters have drastically different stability properties. We evaluate the stability of the same set of streams from Section 4, using random and geographic clustering with redirection to maintain the cluster size and resource availability. We did not evaluate delay-based clus-

tering because we did not have GNP coordinates for these streams.

Figure 17 plots the percentage of incarnations with poor stability performance for each stream using three of the parent selection algorithms previously presented in Section 4. Compared to the performance without any clustering, as presented in Figure 11, the stability performance remains roughly the same. To verify this result, we analyzed the session duration distribution for all clusters of a stream and found that the session duration distributions were similar across all clusters. Thus, clustering does not impact the stability properties of the overlay.

Our results strongly suggest that it is feasible to construct efficient and scalable overlays by leveraging delay-based clustering. In addition, the overlays constructed using various clustering policies have similar stability performance to the overlays constructed without clustering in Section 4.

6. SUMMARY

In this paper, we study one of the most prominent architectural issues in overlay multicast–the feasibility of supporting large-scale groups using an application end-point architecture. Using a large set of live streaming media traces from a commercial content delivery network, we demonstrate that in most of the common scenarios, application end-point architectures (i) have enough resources, (ii) have inherent stability, and (iii) can efficiently support large-scale groups. Our findings show promise for using such architectures for real-world applications.

In addition, we explore and evaluate a range of designs that can help increase the feasibility in practice. We find that minimizing depth in single-tree protocols provides good stability performance. In addition, the use of multiple-tree protocols can significantly improve the perceived quality of streams at the expense of an increase in protocol activity, overhead, and complexity. We also find that simple clustering techniques improve the efficiency of the overlay structure. The strength of our work is perhaps the insight and analysis methodology, more than the designs as such – most of the designs we study are simple.

While our results are encouraging, there are several open issues that are candidates for future work. First, designing policies and mechanisms to encourage application end-points to contribute their resources is an important direction. Second, our results are dependent on application workloads. While we believe that the workloads used in this study represents common behavior for live streaming applications, studying how the findings would change under different or arbitrary application workloads may expose new insight to help us better understand the feasibility of the architecture. Third, there are several unexplored design issues that could also improve performance of application end-point architectures, such as the use of application-level buffers to reduce the impact of ancestor changes in the tree, the (minimum) use of infrastructure to complement the resources and stability in application end-point architectures, and the design of new parent selection algorithms to bridge the gap between minimum depth and oracle. Finally, real large-scale deployment of application end-point architectures will validate and strengthen our findings.

Acknowledgements

We wish to thank Roberto De Prisco of Akamai Technologies, for assistance with collecting log data from the Akamai streaming servers. We also thank the anonymous reviewers for their valuable feedback.

7. REFERENCES

[1] Akamai. http://www.akamai.com/.

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

[3] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[4] 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.

[5] 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.

[6] Y. Chu, J. Chuang, and H. Zhang. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *ACM SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS)*, 2004.

[7] 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*, 2004.

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

[9] P. Francis. Yoid: Your Own Internet Distribution, http://www.aciri.org/yoid/. April 2000.

[10] A. Ganjam and H. Zhang. Connectivity Restrictions in Overlay Multicast. In *Proceedings of NOSSDAV*, 2004.

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

[12] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

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

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

[15] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM*, August 1996.

[16] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of INFOCOM*, June 2002.

[17] T.S.E. Ng, Y. Chu, S.G. Rao, K. Sripanidkulchai, and H. Zhang. Measurement-Based Optimization Techniques for Bandwidth-Demanding Peer-to-Peer Systems. In *Proceedings of IEEE Infocom*, 2003.

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

[19] Planetlab. http://www.planet-lab.org/.

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

[21] Real broadcast network. http://www.real.com/.

[22] R. Renesse, Y. Minsky, and M. Hayden. A Gossip-Style Failure Detection Service. Technical Report TR98-1687, Cornell University Computer Science, 1998.

[23] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of UDP Through Network Address Translators. IETF-Draft, December 2002.

[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)*, October 2002.

[25] 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*, April 2001.