

Dagster: Contributor-Aware End-Host Multicast for Media Streaming in Heterogeneous Environment

Wei Tsang Ooi

Department of Computer Science,
National University of Singapore,
Singapore.
ooiwt@comp.nus.edu.sg

ABSTRACT

We present Dagster, an end-host multicast scheme for delivering multimedia streams. Unlike previous schemes, Dagster does not constrain the amount of bandwidth a node must donate. Instead, it relies on a novel incentive scheme to encourage nodes to contribute more bandwidth to improve the total capacity of the system. The key idea behind the incentive is that Dagster allows a node with more donated bandwidth to preempt another node in the system. Dagster also allows a host to receive from multiple parents at the same time, thus is more resilient to node failures. Our simulation results show that Dagster's design leads to low rejection rate, high resilience to failure and smaller diameter. Furthermore, nodes that donate more bandwidth have lower rejection rate and are positioned fewer overlay hops away from the source, providing incentives for nodes to increase their contribution.

1. INTRODUCTION

Networked multimedia applications such as video conferencing often require one-to-many communication that is not widely supported by existing Internet infrastructure. Even though IP multicast was proposed 15 years ago, it is still not universally deployed due to scalability and economical reasons. Frustrated by lack of deployment of IP multicast, many researchers have proposed application-level techniques to provide multicast. These efforts can be classified into *proxy-assisted approach*, where packets are replicated and forwarded by application-level proxies in network, and *end-host approach*, where replication and forwarding are done by receivers in the multicast session. The end-host approach is particularly attractive because it does not require any additional support from network infrastructures and therefore, can be more easily deployed. The end-host approach is sometimes known as *peer-to-peer multicast* in the literature. End-host multicast can be viewed as an extension to the current peer-to-peer systems, from sharing of disk space and computing power to sharing of network bandwidth as well.

The success of end-host multicast hinges on the assumption that peers are willing to cooperate and share their up-link bandwidth and contribute to the system. Unfortunately, selfish users exist. Studies on Gnutella, a peer-to-peer file sharing system have shown that *free loaders* form a large percentage of its users.¹ The existence of peers that consume bandwidth but contribute little or none in return can severely limit the scalability of end-host multicast. This issue must be taken into consideration in the design of any peer-to-peer systems.

One approach to enforce contribution is to require a node to donate the amount of bandwidth at least as much as they consume. However, studies of host characteristics in peer-to-peer system Napster and Gnutella have revealed a wide range of up-link and down-link bottleneck bandwidth.² A peer that participates in end-host multicast is likely to have the same characteristics. The amount of up-link bandwidth a host donates to serve other peers may be vastly different than the desired receiving bit-rate. Thus, end-host multicast schemes that deter selfish users by enforcing equal upload and download bandwidth will not work well in practice.

The other approach is to provide incentive for peers to donate their resources. A particular successful example is the *tit-for-tat* policy used in peer-to-peer file sharing tool BitTorrent,³ where peers who have uploaded larger amount of data are given higher priority when downloading files. Similar incentive should be provided by end-host multicast schemes in heterogeneous environment to encourage peers to share their up-link bandwidth.

The second issue that end-host multicast schemes must address is the transience of peers.⁴ Unlike proxy-based approach where proxies can be assumed to be reasonably reliable, end-hosts may abruptly leave a session

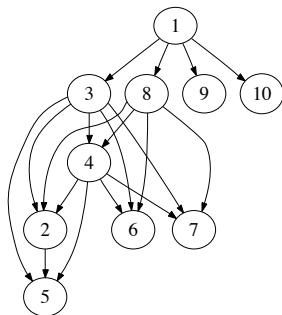


Figure 1. An example of Dagster’s graph

and disrupt service to other peers. Thus, any practical end-host multicast must be designed with robustness in mind to handle transience of peers.

Another issue that arises in a heterogeneous environment is the heterogeneity of receivers. While this issue has been studied extensively in IP Multicast, it was rarely considered in the case of end-host multicast. Receivers with different capability or network capacity might be interested in receiving from the same media source. An end-host multicast scheme should support delivery of media streams with different fidelity or bit-rate to different receivers.

This paper introduces Dagster, a new end-host multicast scheme for delivering non-interactive media streams that addresses the issues mentioned above. There are three key ideas behind Dagster that differentiate it from previous work. First, to improve robustness, Dagster organizes the receivers into an overlay directed acyclic graph, instead of a tree. Second, to allow a receiver to receive from multiple parents and to allow different receivers to receive media streams at different bit-rates, receivers in Dagster transcode the received media stream before forwarding it to another receivers. Finally, Dagster encourages peers to share their up-link bandwidth by providing them incentives to donate higher bandwidth. The incentives come in the form of (a) lower service rejection probability and (b) lower disruption probability. This is done by allowing a peer to preempt another peer with smaller donation. We show that this simple idea leads to several desirable properties of the constructed overlay graph, which can provide the above mentioned incentives to the receivers. A major design issue of Dagster is how to construct the overlay directed acyclic graph. This is the focus of this paper.

Before we proceed to describe the details of the construction algorithm in Dagster, we present a simple example in Figure 1 to give readers a flavor of the overlay structure constructed by Dagster. Figure 1 shows in the topology of a 10-nodes overlay network constructed by Dagster. Node 1 is the source of the media stream, and nodes 2 to 10 are the receivers. Receivers contribute by streaming the content it receives to other receivers and a receiver can receive packets from multiple parents. For example, node 8 is sending to node 2, 4, 6 and 7; and node 4 is receiving from both node 3 and 8 at the same time.

The rest of this paper is organized as follows. Section 2 describes the related work in media streaming using end-host multicast. Section 3 gives the model and an overview of Dagster. We present our graph construction algorithm in Section 4. Experimental results are shown in Section 5 and we conclude in Section 6.

2. RELATED WORK

2.1. End-host Multicast for Media Streaming

Many application-level multicast schemes have been proposed in recent years. In this section, we focus on related work in end-host multicast that is designed for delivering media streams, and highlight the design differences between Dagster and these work.

End System Multicast (ESM)⁵ is the most mature and widely deployed end-host multicast system. ESM organizes end hosts into a single tree for delivery. In contrast, Dagster organizes end hosts into a directed acyclic

graph and allows a node to receive from multiple parents. ESM allows a peer which contributes bandwidth to either preempt a free loader or reduce the rate to the free-loader.⁶ Dagster uses a more general scheme, by allowing a peer i to preempt another peer j as long as the contributed bandwidth of i is larger than j .

Other end-host multicast systems proposed include Yoid, P2Cast, ZIGZAG and PeerCast. Yoid⁷ uses a shared overlay tree for distribution of content, and construct a mesh overlay for control purposes. Yoid tries to make use of native IP multicast when it is available. P2Cast⁸ is an end-host multicast scheme designed for video-on-demand with patching. Besides forwarding streams to other peers, a peer donates storage space by caching the initial part of a video stream so that it can be sent later to newly admitted receivers. Another approach, ZIGZAG,⁹ organizes the peers into a hierarchy of clusters, and construct an overlay tree based on the clusters. By bounding the size of the clusters, the authors bound the degree of each peer and the height of the tree. PeerCast^{4,10} proposes a peering layer in participating nodes, and a redirect primitives for constructing application-level multicast tree.

Like ESM, all approaches above organize end-hosts into a single tree. These work did not address the issue of receiver heterogeneity, and has no mechanism to encourage cooperation among peers. The authors assume that a peer can serve multiple peers at the same bit-rate as the receiving rate. While this assumption is valid in a LAN environment, it is not reasonable when we consider home users with asymmetric network links.

CoopNet¹¹ and SplitStream¹² both organize peers into multiple trees. In these two schemes, the source video is divided into k stripes, and each stripe is delivered through a single multicast tree. The peers are organized such that each peer serves as internal node in one of the tree and as leaf nodes in other trees. Thus, failure of a peer only affects one tree. A node can receive as many stripes as its down-link allows, by subscribing to the appropriate number of trees. Both schemes do not address the issues of free loaders. SplitStream suggested enforcement of the condition that every peer must contribute at least as much up-link bandwidth as the bit-rate of received stream.¹² However, such condition is unnecessarily strong and causes users with asymmetric links to under utilize their down-link.

Bullet¹³ organizes peers into an overlay mesh for dissemination of high-bandwidth data. Bullet allows transmission of disjoint data set to different points in the network, and proposes distributed algorithm for a peer to locate and retrieve data that it needs. Bullet is similar to Dagster in that both deliver data over an overlay mesh instead of a tree. However, Bullet's focus is on making data disjoint and locating data, not on providing incentives. Making data disjoint is a non-issue in Dagster since a receiver always receives disjoint streams from its parent.

Most recently, Chu and Zhang¹⁴ have independently studied the problem of end-host multicast in heterogeneous environment. They showed that altruism is important in improving the performance of end-host multicast, but they did not address the issue of how to provide incentive for peers to be altruistic.

2.2. Incentives in Peer-to-Peer Systems

The issue of providing incentives to peers to contribute their CPU, network, and storage resources in a peer-to-peer systems has garnered a lot of attention lately. The proposed incentive schemes can generally be organized into two categories: (i) *Reputation-based* systems where peers build good reputation over time by contributing resources. Peers with good reputations will receive better services in return.¹⁵ (ii) *Payment-based* approach where peers get rewards in virtual currency for contributing resources, and have to pay in virtual currency to use resources from others.¹⁶ These incentive schemes often allow a peer's rewards to evolve over time and hence require maintenance of historic information. We feel that history maintenance is not desirable as it has been shown by Lai et. al.¹⁷ that storing private history is not scalable, and storing shared history requires support of a common infrastructure.

On the other hand, Dagster's incentive scheme is based on per-session, statically configured resource needs and contributions, and hence does not evolve over time. This avoids the need for maintenance of history information across sessions.

Ngan et. al¹⁸ recently proposed a method to detect selfish nodes in a peer-to-peer multicast systems. Their approach is designed to ensure fairness in the multicast tree, i.e., each node must transmit as much as they receive, and is discussed in terms of SplitStream. The key idea is to periodically rebuild the multicast trees and

let a peer observes the behavior of the parents. Selfish nodes who refuse to forward packets will eventually be detected and dealt with.

Since Dagster is designed for heterogeneous environments with asymmetric links, nodes are unfair to begin with. Thus, fairness is not an issue in Dagster. Dagster depends on altruism to improve the performance on the system. Therefore, our incentive scheme focuses on how to encourage nodes to contribute more up-link bandwidth, rather than ensuring that each node contributes their fair share.

2.3. Transcoding for Heterogeneous Receivers

As far as we know, Dagster is the first end-host multicast system that considers transcoding by peers. Transcoding and adaptation have been proposed as a mechanism to handle receiver heterogeneities in proxy servers in the context of IP multicast,^{19, 20} but until recently, CPU capacity at end-host is not powerful enough to handle the computational requirements of these tasks. Thus, none of the end-host multicast scheme previously proposed has integrated media transcoding by peers into their systems. However, the concept of transcoding received media streams to serve other peers is not new. The VLC media player²¹ already allows a user to transcode and forward the received media stream to other users.

The only work we are aware of that performs peer-to-peer media adaptations is by Kouvelas et. al. They proposed using Self-organizing Transcoder (SOT)²² to perform local repairs and congestion controls in IP multicast. Upon detecting link congestion, a group of peers can coordinate and select an upstream peer to adapt the stream for downstream receivers. In this paper, we only consider transcoding in Dagster that is performed based on peer's requirement. Dagster does not preclude adapting streams based on network conditions, but this issue is outside the scope of this paper.

3. MODEL AND ASSUMPTIONS

Dagster is an end-host multicast scheme, where each receiving end-host contributes some specified amount of up-link bandwidth to serve other peers. A receiver may encode and send a segment of the received video stream to some other receivers, and may receive data from more than one other receivers. Receivers in Dagster are organized into a directed acyclic graph, or DAG. The DAG is source-specific – that is, a DAG is constructed for all participants that wish to view a stream originated from a source.

We borrow terminologies from graph theory and use the term node inter-changeably with end-hosts in this paper. We say that a node i is a *parent* of j and j is a *child* of i , if i is sending data to j . The terms *ancestor* and *descendant* are defined in the usual way.

Unlike previous models, Dagster places no restriction on the bit-rate of the stream received by a node and the up-link bandwidth donated by a node. Every node in Dagster specifies their required bit-rate and amount of up-link bandwidth they are willing to donate (called *donation* for short) when joining a multicast session. Nodes in Dagster may also place a bound on their in-degrees (i.e., number of parents) and out-degrees (i.e., the number of children).

3.1. Centralized State Maintenance

States of each peer and the graph representing the current overlay topology are maintained centrally at the source of the multicast session. Decisions related to construction and maintenance of the graph in Dagster are done at the source as well. A common argument against centralized state maintenance and decision making is that it constitutes a single point of failure. However, this argument does not apply here. If the source fails, there will be no data to receive anyway. The same argument is made by Chou et. al.¹¹

A motivation for us to use centralized state management is to have an efficient mechanism to avoid loops in the construction of trees. One source of complication in the design of distributed tree maintenance protocol is avoidance of loops and network partitions (For example, see Yoid⁷). By adopting a centralized solution, Dagster avoids the complexity of distributed tree maintenance protocols.

The other reason why we decided to use a centralized solution is to avoid cheating. Since Dagster allows nodes to preempt each other, there must exist a centralized authority that can issue commands for such preemption to

happen. A fully distributed solution would allow nodes to cheat by sending fake preemption messages to others. In Dagster, all such messages must come from the source. While this restriction does not prevent the source from implementing policies that are biased toward certain group of receivers, we view this as acceptable since the source should have the right to give higher priority to certain receivers.

3.2. Assumptions

Dagster is designed for non-interactive streaming. Thus, we do not consider reducing end-to-end latency in organizing peers in the overlay network.

In this paper, we assume that the core networks have abundance bandwidth, and the bottlenecks occur at the network edges. Like many popular peer-to-peer applications, we let the users configure what is the bit-rate of the stream they want to receive and how much up-link bandwidth they are willing to donate. We assume that users will not specify values that are larger than the capability of their network connections.

An issue that we omit in this paper is network efficiency. Dagster is not topology-aware. We fully realize the importance of considering network topology in constructing the overlay graph for disseminating media streams. However, to allow us to focus on the issue of providing incentives, we do not consider network topology in this paper and leave it as future work.

In a heterogeneous environment where nodes may contribute less than they consume, the scalability of any end-host multicast scheme will be limited, as the advantage of self-scaling no longer applies. Limited by this fact, Dagster is not designed to be scalable to thousands of nodes, and is more suitable for smaller scale multicast with tens to hundreds of members. Our decision to use centralized state management also limits the scalability of Dagster.

4. DESIGN OF DAGSTER

We now present the design of Dagster. We first elaborate on the novel features of Dagster. We then introduce and motivate four rules that govern the Dagster algorithm for admitting and organizing peers. Finally, we describe the algorithm itself.

4.1. Features

Dagster is designed for heterogeneous environment, where peers may have different down-link bandwidth and asymmetric links. This has lead to different design decisions compared to previously proposed scheme. The main features of Dagster that differentiate it from previous work are, (i) it provides incentives for peers to donate higher up-link bandwidth to serve other peers, (ii) it allows a node to receive from multiple peers using distributed streaming, and (iii) it requires a parent to transcode incoming streams for its children. We elaborate on these features below.

Incentive for Larger Donations Since Dagster allows participating nodes to have different desired bit-rate and donated bandwidth, it is important for Dagster to provide incentives for the participating hosts to donate their up-link bandwidth. Dagster achieves this by taking node donation into consideration when organizing nodes into the overlay DAG. A new node is allowed to preempt an existing child c if its donation is larger than c 's. Nodes with small donation will eventually be pushed further away from the source. Furthermore, a node that asks for high input bit-rate but makes small donation will have a higher chance of being rejected, as it is more likely not to be able to find a place in the graph to be inserted into.

Receiving from Multiple Parents Dagster allows a participating node to receive a stream from multiple parents. This is motivated by two reasons. The major reason is that, since peers are transient, receiving from multiple parents improves the robustness of the system – in the case when a parent fails and the child cannot find a replacement, the failure only reduces the quality of the received stream, but the child still receives something. Another, minor reason for having multiple parents is to fully utilize donated bandwidth of nodes. Since Dagster is designed for nodes in a heterogeneous environment, the bit-rate requested by a node may be higher than any other nodes is willing to provide. Having multiple parents allow Dagster to aggregate the donated bandwidth from multiple nodes to serve a single node.

Transcoding by Parents Since nodes may want to view the same media content at different bit-rate, Dagster requires a node to re-encode the received stream for transmission to other peers. Recent increase in CPU capacity has allowed this to be practical*. This decision also ensures that a node can receive disjoint data from different parents (see Section 4.5). Nodes in Dagster are required to donate both CPU time for transcoding and up-link bandwidth for forwarding. For simplicity, our subsequent discussion of Dagster will focus on bandwidth constraint and donations only. CPU time considerations can be easily added into the framework.

4.2. Rules

We now introduce some rules that govern peer organization in Dagster. But before we list down the rules, we first introduce some notations and terms.

For a node i , we denote the required input bit-rate as R_i , the donated bandwidth as B_i , the maximum in-degree (i.e., number of parents) as P_{max} and the maximum out-degree (i.e., maximum number of children) as C_{max} . The number of parents required by node i is denoted as P_i , and number of available out-degree slots is denoted C_i . Furthermore, we denote $b_{i,j}$ as the data-rate being sent from node i to node j . We define A_i as the available donated bandwidth, i.e.,

$$A_i = B_i - \sum_{j \text{ child of } i} b_{i,j}$$

It is possible for a parent p to stop sending to its child i , in favor of serving another newly arrival node j . We say that parent p *abandons* i , and node j *preempts* i . The abandoned node i has to look for a new parent to replace p . If a new parent q is found, we say that q *adopts* i .

We now present the four rules for constructing the overlay DAG in Dagster and the motivation behind each of them.

RULE 1 (MONOTONICALLY DECREASING BIT RATE). *If i is a parent of j , then $R_i \geq R_j$.* This rule is needed because Dagster allows receivers to receive stream at different bit-rate. A node can only receive a stream with quality as good as its parent. Hence, when a node i is inserted into the system, it must locate parents whose input stream is of higher quality than its requirement. A consequence of this rule is that, the requested bit-rate along any path from the source to a node is monotonically decreasing.

RULE 2 (EQUAL BIT-RATE FROM PARENTS). *For a node i and any two of its parents j and k , $b_{j,i} = b_{k,i}$.* This rule says that a node divides its required bit-rate equally among all of its parents. Since the probability of node failure is unknown in practice, Dagster assumes that each parent is equally likely to quit or fail, and ignore common ancestors of the parents. Under this simplifying assumption, we divide the sending rate equally among the parents to maximize the received bit-rate if one of the parent has failed. Doing so also allows us to employ multiple state encoding to code the data sent from parents to a child (see Section 4.5).

RULE 3 (RULE OF PREEMPTION). *If $B_i > B_j$, then i can preempt j .* The rule of preemption is the key to encouraging nodes to donate larger amount of up-link bandwidth to serve other peers. By having larger B_i , a node i is allowed to preempt more nodes, thus increasing its chance of being served, and being inserted closer to the source in the DAG. Note that i can preempt j only if its donation is *strictly larger* than j . Borrowing the term from Chu et. al.,⁶ we say that Dagster is *contributor-aware* since the rule of preemption is biased toward nodes with larger contribution.

RULE 4 (RULE OF CONTINUOUS SERVICE). *Once a node i is admitted, it will not be removed from the session unless all of its parents have failed or i leaves voluntarily.* This rule ensures that a node will receive continuous service as much as possible. In particular, it ensures that a preempted node will always find a new parent. This rule is important since a node with large number of descendants can still be preempted by another node with larger donation. Disrupting service to such a node would affect a large number of peers in the system.

*A quick benchmark shows `ffmpeg-0.4.823` can encode a 352kbps, CIF, MPEG-1 Foreman sequence at the rate of 330 frames per second on a Xeon 2.8GHz Linux machine using command line `ffmpeg -benchmark -s 352x288 -i foreman.yuv -g 10 -bf 6 foreman.mpg`.

4.3. Algorithm

Based on the above rules, we are now ready to describe how a node is inserted by Dagster into the DAG when it requests to join a session.

We first define the term *eligible*. We say that a node i is *eligible* to serve another node j , if all of the following conditions are true:

- $R_i \geq R_j$. This follows from Rule 1.
- $A_i \geq R_j/P_j$, i.e., i has enough remaining donation to serve j . From Rule 2, the required sending rate from i to j is R_j/P_j . Note that if i is the source, then $P_j = 1$.
- $C_i > 0$, i.e., i has not reached its out-degree limit.
- j is not an ancestor of i . This is to prevent cycles in the overlay DAG.

A node i that wishes to join a multicast session sends a join message to the source, indicating its required bit-rate and the amount of bandwidth it is willing to donate. The source will check if the system can admit i as a receiver, by trying to insert i into the DAG constrained by the rules given in Section 4.2. We divide this process into five steps.

Step 1. The source s will first check if s is eligible to serve i . Note that since s is the source of the video stream, we define R_s to be the original bit-rate of the media stream. Any node which requested higher R_i than R_s will be rejected immediately. If s is eligible to serve i , s will create a unicast connection directly to i to avoid routing through other peers.

Step 2. If s is not eligible, then either s has reached its out-degree limit or has exhausted its donation. s tries to free up some of its out-degree slots and donated bandwidth for i . To do this, s checks if it can abandon a subset of its children to make space for i . This process must abide by the rule of preemption and the rule of continuous service. To comply with the rule of preemption, the source s only considers abandoning a child c if $B_i > B_c$. To ensure continuous service to child c , s makes sure that each of the abandoned child c can find P_c eligible adoptive parents. Note that it is possible for the new node i to become an adoptive parent for a child abandoned by s . If s can free up enough of its resources to serve i , s will admit i as a child, and abandon some of its children for adoption by other nodes. Otherwise, s cannot serve as parent of i . s will now try and find P_i (initialized to P_{max}) nodes that can serve as parents to i .

Step 3. If s can find at least P_i number of nodes that are eligible to serve i , then i is admitted and P_i eligible nodes with smallest level is selected as parents. We define the level of a node as the length of the longest path from the source to that node. If two eligible nodes are at the same level, we break ties by choosing the node with larger donation. The rationale behind this heuristic is that a parent with larger donation is less likely to be preempted, and since preemption may increase the level of a node and all its descendants, it is preferable for a node to “hide behind” a parent with larger donation.[†]

Step 4. If s cannot find enough eligible parents for i , then s tries to preempt some nodes for i . Let k be the number of eligible parents found in Step 3. s tries to find $P_i - k$ more eligible parents by asking them to abandon some of their children. The process is similar to Step 2 above, except that for each abandoned child c , c needs to find only one adoptive parent as replacement (instead of P_c as in Step 2). Again, s selects parents with lowest level as parents for i , breaking ties by selecting parents with larger donation.

Step 5. If P_i number of eligible parents cannot be found, even after asking peers to abandon their children, s reduces the number of parents required P_i by 1, and repeats Step 3, until either P_i parents are found, or until P_i is 0. When s cannot locate even a single parent to serve i , i 's request to join the session is rejected.

Figure 2 shows an example of preemption and adoption. A sequence of node insertion is illustrated from left to right. Each node is labeled with its id and a pair of numbers (R, B) , where R is the required bandwidth and

[†]To make Dagster topologically aware, the parent selections can be modified to consider metrics such as hop count and bottleneck bandwidth.

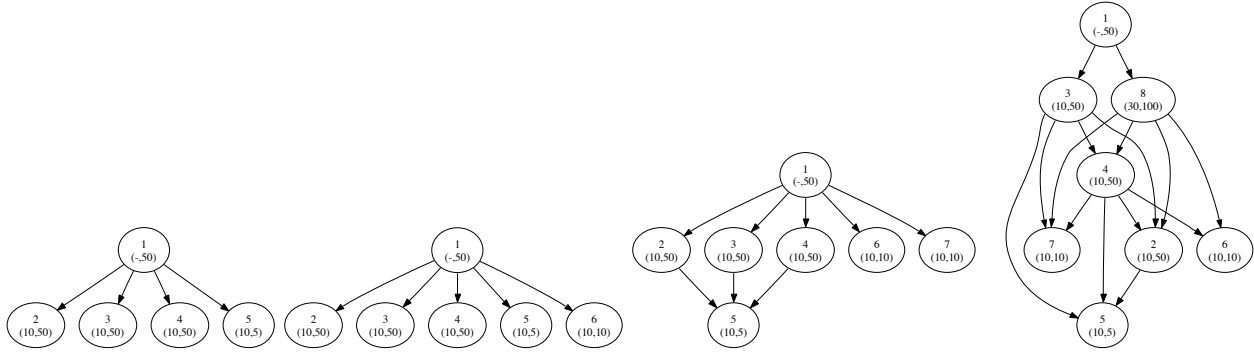


Figure 2. An Illustration of Dagster’s Algorithm

B is the donated bandwidth. The left most picture shows the initial DAG with five nodes. When node 6 is inserted, the source, node 1, is eligible to serve node 6. Thus, node 6 becomes a child of node 1. When node 7 arrives, node 1 has ran out of donated bandwidth. Since node 7 donates more than node 5, the source abandons node 5 to admit node 7 as its child. Node 5 is adopted by node 2, 3 and 4. Finally, node 8 arrives. Node 8 has even larger donation than the other nodes, and requested much higher bit-rate from the source. The source has to abandon node 2, 4, 6 and 7 in order to serve node 8. The final DAG is shown on the right.

4.4. Selecting Children To Abandon

A detail we omitted in the description of the algorithm above is how to select children to abandon. Given a set of children $C = \{c_1, c_2, \dots, c_n\}$ of a node i that can be adopted by other nodes, the amount of bandwidth we need to free b_f , we want to select a set $C' \subseteq C$ such that

$$\sum_{c_j \in C'} b_{i,c_j} \geq b_f$$

However, when more than one such subset are possible, we need to select a “good” subset. Since abandoning a child could increase its level, Dagster selects a subset C' such that the total number of descendants of the abandoned children is minimize. This problem is equivalent to the MINIMUM KNAPSACK problem, which is NP-complete. In our current implementation of Dagster, we employ the standard pseudo-polynomial, dynamic programming, “textbook” algorithm to select the children to abandon.²⁴

4.5. Media Segmentation

To enable streaming from multiple parents to a single node, Dagster utilizes recent advances in multiple descriptive coding (MDC). MDC encodes a media stream into multiple stripes of equal importance, and a node can receive *any* subset of the stripes to decode the stream. The more stripes a node receives, the higher the quality of the stream decoded. Unlike layered coding, stripes coded with MDC have no dependency among each other, thus making it more robust, at the price of additional coding overhead.

A simple MDC coding would be to divide a video stream into two stripes, one consists of even numbered frames and the other consists of odd numbered frames, and encode each stripe independently. A node that receives both stripes can decode and playback at full frame rate. If only one stripe is received, the node can playback at half the original frame rate. The coding overhead comes from lower compression ratio due to reduced temporal redundancy among the frames in each stripe. This coding method is known as *multiple state encoding*.²⁵

Dagster uses multiple state encoding to segment the media stream among multiple parents. A node assigns an id 0 to k to each of its parent. Parent with id j will encode and send frame $j, j + k, j + 2k, \dots$ to the child.

4.6. Dealing with Session Dynamics

A multicast session is dynamic in nature, with nodes joining and leaving occasionally. In Dagster, node departure and failure are treated in a rather straight forward way. When a node is removed from the overlay network, the source is notified. The source will then try to locate new parents for each of these children, in decreasing order of their donations. If one of these children cannot be re-admitted, then Dagster will try to recursively admit its children.

As new peers arrive and get inserted into the overlay DAG, available capacity and the structure of the DAG changes as well. Consider a node i with large donation that joins later in the session. The overall capacity of the system has increased, but nodes that join before i cannot take advantage of this. Dagster allows nodes to periodically probe the source to search for a better parent so that donations from newer peers can be fully utilized. This can also happen if a node experiences frequent packet loss and congestion.

4.7. Prevent Cheating

The use of centralized state management and decision making in Dagster disallows exchange of control messages that can modify the overlay topology among peers. This prevents peers from sending fake messages to preempt each other in the overlay tree. However, since Dagster’s algorithm rewards peer with large donation, a peer can cheat by telling the source that it is making large donation, but in fact is willing to send less. We can catch such cheaters if the children complain that they are not receiving the bandwidth they are suppose to get, but this can be done only if the amount of bandwidth used is larger than the actual donation.

One question that arises here is whether children can log fake complaints to frame their parents. While this is possible, a child has no incentive to frame its parents, because if the parents are punished, the child will be affected as well. The other question is how to ensure that a peer uses their promised donations so that the children can “audit” their parents behavior. A possible solution is to adopt the techniques from Ngan et. al¹⁸ and periodically reorganize the graph. These issues on cheating are not investigated further in this paper and remain as interesting open questions.

5. EXPERIMENTAL RESULTS

This section presents simulation results that show the feasibility of Dagster’s approach and its improvement over previously proposed algorithms. We simulated Dagster’s algorithm using ns-2, using the eBone network topology collected by RocketFuel.²⁶

We model participating peers based on the host characteristics collected by Sariou et. al.² The donated bandwidth for each peer is drawn from the up-link bandwidth distribution based on the paper, while the required video bit-rate is derived from the down-link bottleneck bandwidth reported. We set the donated bandwidth of the source to 4Mbps. Based on the reported down-link bottleneck bandwidth, a peer randomly chooses a required bit-rate that fits its down-link bandwidth. The maximum required bit-rate in our simulation is set at 768Kbps, roughly equivalent to a two-hour movie on a 700MB CD. The distribution used in our simulation is shown in Figure 3. In our simulation, we assume that peer arrival is a Poisson process. To study the “raw” performance of Dagster, we did not model node departure nor periodic rejoin as described in Section 4.6.

5.1. Rejection Rate

An important performance metric for end-host multicast scheme is rejection rate, or the percentage of peers that cannot be admitted. Dagster’s peer organization algorithm is designed for low rejection rate, and tries to admit as many receivers as possible. Dagster allows a node to preempt another node, thus increasing the probability of admission.

We compared the performance of Dagster’s algorithm with two other algorithms. The first one is *Best Fit*, which is similar to algorithms used in several previous work.^{5,8} The best fit algorithm inserts a newly arrival node into a parent with largest available bandwidth. A node is rejected if there does not exist a node with enough bandwidth to satisfy the required bandwidth. The second algorithm we compared against is a variation of Dagster’s algorithm that violates the rule of preemption. In other words, we removed contributor-awareness, and allowed a node to preempt any other nodes, without considering bandwidth donation.

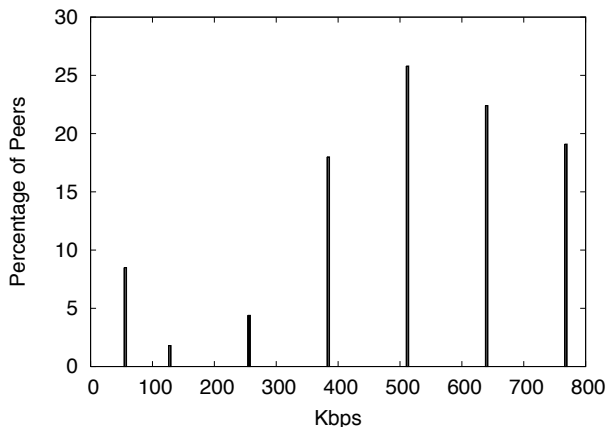


Figure 3. Histogram for Our Model of Required Bit-rate

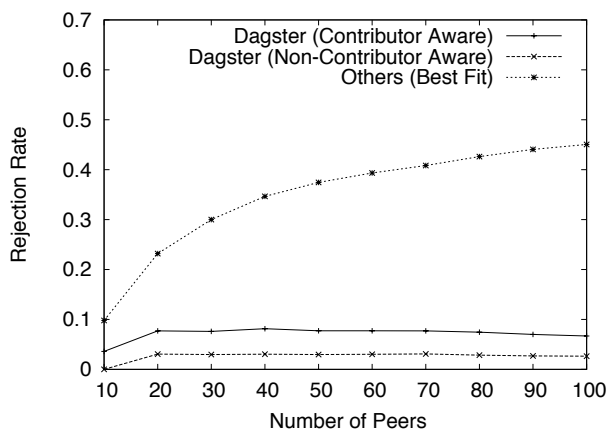


Figure 4. Rejection Rate for Different Algorithms.

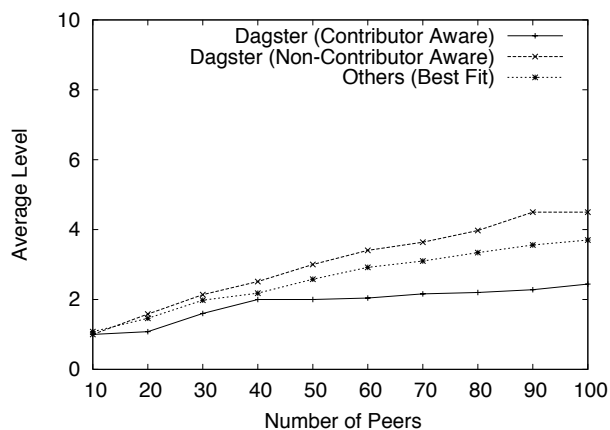


Figure 5. Average Node Levels for Different Algorithms.

Figure 4 shows the result of the comparisons. In this experiment, we set the maximum out-degree to 5 and maximum in-degree to 1 (This magic number 5 for out-degree is used in evaluation of P2Cast⁸ and PeerCast¹⁰ as well). This configuration always produces a tree, a common structure for organizing peers in the literature. We can see that Best Fit algorithm has a much higher rejection rate, and can admit slightly more than half the peers when number of peers is 100. Both Dagster’s algorithms perform much better, rejecting less than 10 percent of the clients. The results also show that without contributor-awareness, Dagster achieves a lower rejection rate. This result is expected, since removing the rule of preemption allows more flexibility in node insertions. However, as we will show in Section 5.3, removing contributor-awareness remove incentives for nodes to contribute higher bandwidth.

5.2. Average Node Level

Another important metric for evaluating overlay network is the diameter of the graph. It is desirable for media stream to pass through as few hops as possible, since it reduces the probability of service disruption and fluctuation. In Dagster, nodes with higher up-link donation are positioned fewer hops away from the source. Since these nodes can admit more children, Dagster’s algorithm leads to lower diameter of the graph and reduces the average level of nodes. A comparison of average node levels for Dagster and previous schemes using best fit algorithm is shown in Figure 5. This figure is produced using the same experimental settings as Figure 4.

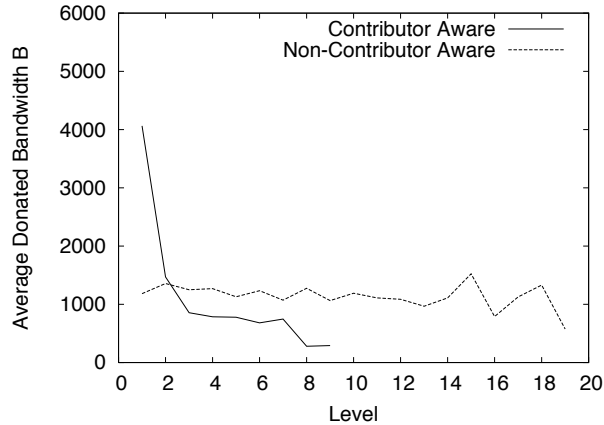


Figure 6. Relation between Donated Bandwidth B and Node Level.

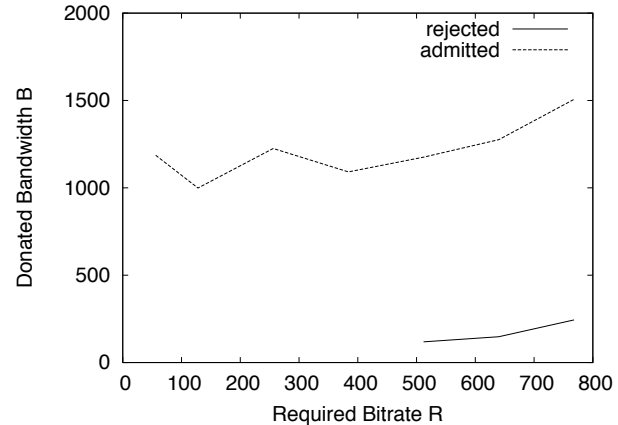


Figure 7. Average Donated Bandwidth B for Admitted Nodes and Rejected Nodes.

5.3. Providing Incentives

One of the rationals behind the rule of preemption is that it provides incentives for nodes to contribute larger amount of up-link bandwidth, as it will lead to lower rejection rate, and has a higher chance of being inserted closer to the source. In this section, we present experimental results that support the effectiveness of Dagster contributor-aware algorithm.

Figure 6 shows the relationship between average donated bandwidth and level of peers in 50 runs of simulations, each with 100 peers, maximum in-degree 1, and maximum out-degree 5. The figure shows that there is a clear relationship between the level of a node and its donated bandwidth when the rule of preemption is enforced (i.e., is contributor-aware). The higher the donation, the smaller the level of a node. Without contributor-awareness, there is no clear correlation between the level of a node and its donation. Thus, there is no incentive for contributing larger bandwidth.

Figure 7 illustrates another incentive for contributing larger up-link bandwidth. The figure plots the average donated bandwidth versus required bandwidth for all rejected and admitted nodes over 50 runs of our simulations using the same configuration as before. We can see that the average donated bandwidth for nodes that are admitted is higher than those who are rejected. Furthermore, rejected nodes have high R/B ratio, i.e., they ask to be served with high bandwidth stream, but only donate small amount of bandwidth in return.

5.4. Number of Parents

Next, we compare the effects of increasing the number of parents on the rejection rate. Since increasing the number of parents increases the number of outgoing connections, we set the maximum out-degree to be a function of maximum in-degree. In this experiment, we set maximum out-degree to be $5P_{max}$. Our experiments show that increasing the number of parents from 1 to larger than 1 reduced the rejection rate, but not by much (see Figure 8). This indicates that in terms of rejection rate, increasing the maximum in-degree has little advantages.

The main advantage of receiving from multiple parents is that it improves resilience to node failures. If one of the parents fails, the child can still receive from other parents, hence service to a node will not be disrupted, albeit the quality will be lower.

We constructed 50 instances of Dagster’s overlay graph with 100 peers, and computed the expected reduction in received bit-rate if each peer fails with some probability. The results are shown in Figure 9. The figure shows that if the maximum in-degree for each node is 1, the expected received rate for a node is only slightly above 60 percent of the original rate if the failure rate is 50 percent. In contrast, if a node has multiple parents, the expected rate is about 90 percent of the original.

Figure 10 shows the effects of the average level of a node, when we change the maximum in-degree and number of peers. The figure indicates that as the maximum in-degree increases, the average level of a node increases as

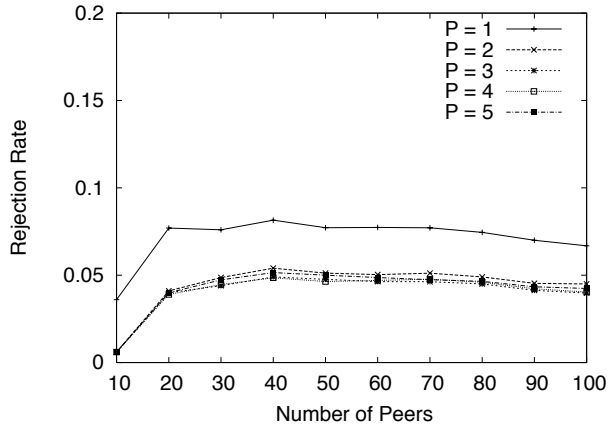


Figure 8. Rejection Rate for Different Maximum In-Degree P

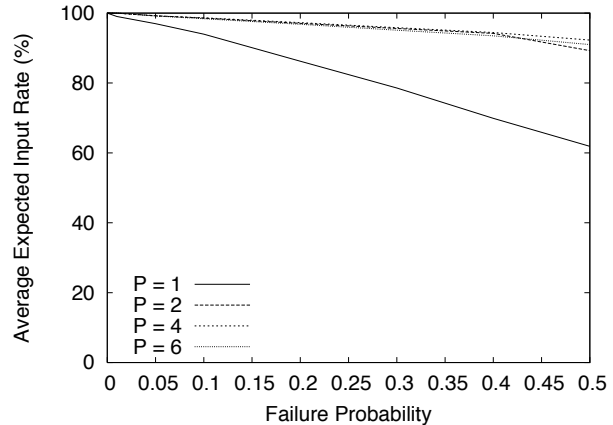


Figure 9. Expected Receiving Rate for Different Maximum In-Degree P , versus Node Failure Probability.

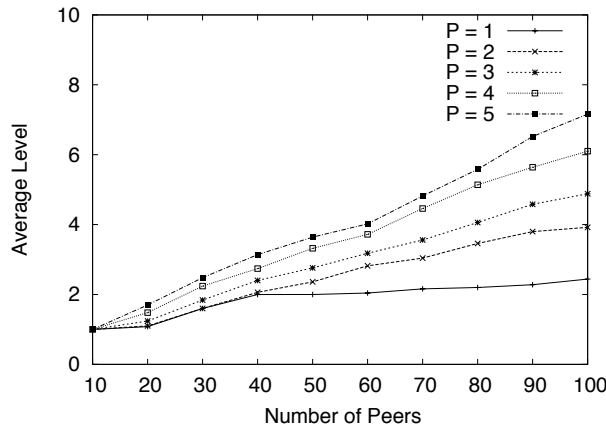


Figure 10. Average Level for Different Maximum In-Degree P , versus Number of Peers.

well. The reason behind this increase is that, the more parents a node has, the higher the chance that it will be assigned a parent with larger level. Since the level of a node is the maximum level of all its parents plus one, this change increases the level of the node as well.

Our results are somewhat surprising. We expected that increasing the number of parents will improve the performance of the system. While our results indicate that having multiple parents is definitely more beneficial compared to having a single parent, there are little advantages in increasing the number of parents to more than 2. Not only the rejection rate and expected input bit-rate do not improve much, doing so would reduce the performance of the system by increasing the average node levels. We suggest that each node should have 3 parents at most in Dagster. This value is enough to achieve acceptable frame rate if one of the parent has failed. Since common video streams are sent at the frame rate of 24 - 30 fps, failure of one parent reduces the frame rate to 16 - 20 fps, which is still in acceptable range based on previous user studies.²⁷

6. CONCLUSION

This paper presents Dagster, an end-host multicast scheme for delivering multimedia streams in a heterogeneous environment, where different receivers might need streams with different bit-rate, and up-link bandwidth can be vastly smaller than down-link bandwidth. To tackle this environment, Dagster supports delivering of media streams over a directed acyclic graph and an incentive scheme that rewards hosts that donate more bandwidth.

However, many questions remain. An important issue is how to make Dagster topologically aware so that the constructed overlay is efficient in terms of metrics such as link stress and stretch. Another issue is how to prevent peers from cheating. We briefly mentioned some ideas in this paper but the detail needs to be worked out before Dagster can be implemented and deployed on real networks.

REFERENCES

1. E. Adar and B. Huberman, "Free riding on Gnutella," *First Monday* **5**, Oct. 2000.
2. S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," *Multimedia Systems Journal* **9**, pp. 170–184, Aug. 2003.
3. M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five months in a torrent's lifetime," in *Passive and Active Measurements Workshop*, (Antibes Juan-les-Pins, France), Apr. 2004.
4. M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers and streaming media," in *Proceedings of HotNets-I*, (Princeton, NJ), 2002.
5. Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the Internet using an overlay multicast architecture," in *Proceedings of ACM SIGCOMM*, (San Diego, CA), Aug. 2001.
6. Y. Chu, A. Ganjam, T. S. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early experience with an Internet broadcast system based on overlay multicast," Technical Report CMU-CS-03-214, Carnegie Mellon University, Dec. 2003.
7. P. Francis, "Yoid: Extending the Internet multicast architecture, <http://www.icir.org/yoid/>," 2000.
8. Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer patching scheme for VoD service," in *Proceedings of World Wide Web Conference 2003*, (Budapest, Hungary), May 2003.
9. D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proceedings of INFOCOM 2003*, (San Francisco, CA), 2003.
10. H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peers," Tech. Rep. 2002-21, Stanford University, Stanford, CA, Mar. 2002.
11. V. Padmanabhan, H. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proceedings of IEEE ICNP*, (Atlanta, GA), Nov. 2003.
12. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," in *Proceedings of SOSP 2003*, (Lake Bolton, NY), Oct. 2003.
13. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proceedings of SOSP 2003*, (Bolton Landing, NY), 2003.
14. Y. Chu and H. Zhang, "Considering altruism in peer-to-peer internet streaming broadcast," in *Proceedings of NOSSDAV*, (Kinsale, Ireland), 2004.
15. H. T. Kung and C. Wu, "Differentiated admission for peer-to-peer systems: Incentivizing peers to contribute their resources," in *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, (Berkeley, CA), 2003.
16. P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," in *Proceedings of the 3rd ACM Conference on Electronic Commerce*, (Tampa, FL), 2001.
17. K. Lai, M. Feldman, I. Stoica, and J. Chuang, "Incentives for cooperation in peer-to-peer networks," in *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, (Berkeley, CA), 2003.
18. T. Ngan, D. Wallach, and P. Druschel, "Incentives-compatible peer-to-peer multicast," in *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems*, (Cambridge, MA), 2004.
19. E. Amir, S. McCanne, and Z. Hui, "An application level video gateway," in *Proceedings of 3rd ACM International Multimedia Conference and Exhibition*, pp. 255–266, (San Francisco, CA), Nov. 1995.
20. W. T. Ooi, R. van Renesse, and B. C. Smith, "The design and implementation of programmable media gateways," in *Proceedings of 10th. Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'00)*, (Chapel Hill, North Carolina), June 2000.
21. VideoLAN, "VLC media player 0.7.1." <http://www.videolan.org>, Apr. 2004.

22. I. Kouvelas, V. Hardman, and J. Crowcroft, "Network adaptive continuous-media applications through self organised transcoding," in *Proceedings of the 8th Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98)*, (Cambridge, U.K.), 1998.
23. FFMpeg, "FFMPEG multimedia systems 0.4.8." <http://ffmpeg.sf.net>, Sept. 2003.
24. S. Skiena, *The Algorithm Design Manual*, Spinger-Verlag, New York, NY, 1998.
25. J. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity," in *Proceedings of Visual Communication and Image Processing, (VCIP '01)*, pp. 392–409, (San Jose, CA), Jan. 2001.
26. N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with RocketFuel," in *Proceedings of ACM SIGCOMM*, (Pittsburgh, PA), 2002.
27. R. T. Apteker, J. A. Fisher, V. S. Kisimov, and H. Neishlos, "Video acceptability and frame rate," *IEEE Multimedia* **2**(3), pp. 32–40, 1995.