Resilient Peer-to-Peer Streaming*

Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou
Microsoft Research
{padmanab, helenw, pachou}@microsoft.com

Abstract

We consider the problem of distributing "live" streaming media content to a potentially large and highly dynamic population of hosts. Peer-to-peer content distribution is attractive in this setting because the bandwidth available to serve content scales with demand. A key challenge, however, is making content distribution robust to peer transience. Our approach to providing robustness is to introduce redundancy, both in network paths and in data. We use multiple, diverse distribution trees to provide redundancy in network paths and multiple description coding (MDC) to provide redundancy in data.

We present a simple tree management algorithm that provides the necessary path diversity and describe an adaptation framework for MDC based on scalable receiver feedback. We evaluate these using MDC applied to real video data coupled with real usage traces from a major news site that experienced a large flash crowd for live streaming content. Our results show very significant benefits in using multiple distribution trees and MDC, with a 22 dB improvement in PSNR in some cases.

I. Introduction

We consider the problem of distributing "live" streaming media content from a server to a potentially large and highly dynamic population of interested clients. We use the term "live" to refer to the simultaneous distribution of the same content to all clients; the content itself may either be truly live or a playback of a recording. Due to the lack of widespread support for IP multicast (especially at the inter-domain level), the server may resort to unicasting the stream to individual clients. However, this approach only scales up to a point. A surge in the client population, say due to a flash crowd, could easily overwhelm the server's bandwidth.

A range of solutions have been proposed in the literature and employed in practice. The content provider could purchase additional bandwidth and install a (possibly distributed) cluster of servers. Alternatively, the services of a content distribution network (CDN) such as Akamai could be used to achieve the necessary scaling, thereby relieving the content provider from the task of scaling their server site. However, these approaches may not be cost effective, at least for small or medium sized sites, because the normal traffic levels may not be high enough to justify the cost of purchasing additional bandwidth or subscribing to the services of a CDN. In fact, the volume of traffic at a small site, even during a flash crowd, may be too low to be of commercial interest to a CDN operator. (Consider, for instance, a flash crowd that overwhelms a server that is webcasting a high school football game.) Furthermore, there is some evidence that even large sites (e.g., CNN) are moving away from CDNs to in-house server farms [23].

An alternative to these infrastructure-based solutions is end-host-based or peer-to-peer content distribution.[1] A P2P approach is attractive in this setting because the bandwidth available to serve content scales with demand (i.e., the number of interested clients). This is the basis for the CoopNet system presented in this paper. CoopNet makes selective use of P2P networking, placing minimal demands on the peers. The goal is only to help a server tide over crises such as flash crowds rather than replace the server with a pure P2P system.

There are a few key issues that need to be addressed in CoopNet. First, users may be wary of dedicating their bandwidth to the common good, especially when ISPs charge based on (upstream) bandwidth usage. We address this issue in CoopNet by insisting that a node participate in and contribute bandwidth for content distribution only so long as the user is interested in the content. It stops forwarding traffic when the user tunes out. This requirement makes CoopNet fundamentally different from many other P2P systems (e.g., [12]) where nodes are expected to route traffic so long as they are online, even if they are themselves not interested in the corresponding content. We also insist that a node only contribute as much upstream bandwidth as it consumes in the downstream direction[2]. This creates a natural incentive structure where a node may tune in to higher bandwidth (and better quality) content if and only if it is also willing and able to forward traffic at the higher rate. We do not, however, consider the enforcement issue (e.g., blocking free-riders) in this paper.

A second key issue is that the nodes in CoopNet are inherently unreliable. The outgoing stream from a node may be disrupted because the user tunes out, the node crashes or loses connectivity, or simply because the upstream bandwidth is temporarily used up by a higher-priority user task (e.g., sending out an email with large attachments)[3]. The traditional approach to end-host-based application-level multicast, which involves constructing a single distribution tree, is vulnerable to such failures because the descendants of the failed nodes might experience severe disruption until the tree is repaired (or the failed nodes are revived). Parent-driven retransmission

*Please visit the CoopNet project page at http://www.research.microsoft.com/projects/CoopNet/ for additional information, including a pointer to a more detailed paper [28].

[1] We use the terms end-host-based multicast and peer-to-peer multicast synonymously in this paper.

[2] This restriction only applies to the *total* bandwidth in and out of a node aggregated over all trees. Thus the individual trees will still be "bushy", as explained in Section II-B

[3] We term these as "failures" although the node may not have actually failed.

(ARQ) is not a good fit because we are concerned with the failure of the parent node itself, not just network packet drops. So we address the robustness issue in CoopNet by introducing redundancy, both in network paths and in data. Multiple, diverse distribution trees spanning the set of participating nodes are constructed, thus providing redundancy in network paths. The streaming content is encoded using multiple description coding (MDC) [19] and the descriptions are distributed over different trees. As our experimental results show, this approach significantly improves the quality of the received stream in the face of a high level of node churn.

The use of multiple trees also enables us to achieve our goal of making the total upstream and downstream bandwidth consumptions equal at each node, while still maintaining a significant fan-out at each node. We explain in Section II-B and Figure 1 on how this is done.

In CoopNet, the server plays a central role in constructing and managing the distribution trees. The availability of a resourceful server that is likely to be far more robust than any individual peer greatly simplifies the system design. Note that in this "centralized" design, the most constrained resource, viz. bandwidth for forwarding the data stream, is still contributed by the distributed set of peers and scales with the population size. In this respect, our design is akin to that of the erstwhile Napster system. While the central server does constitute a single point of failure, it is also the source of the data stream. So failures of the server will disrupt the data stream regardless of how tree management is done.

Here are the specific contributions of this paper:

1) A simple, centralized tree management algorithm to construct and maintain a diverse set of trees.
2) A framework for adapting MDC based on scalable receiver feedback.
3) Evaluation of tree management and MDC adaptation using real video data coupled with real usage traces derived from the access logs of the MSNBC news site [2] that experienced a large flash crowd for live streaming content on Sep 11, 2001. Our results show the significant benefits of using multiple, diverse distribution trees and MDC. The peak signal-to-noise ratio (PSNR) of the received stream improves by up to 22 dB in some cases. Our results also indicate that MDC outperforms pure Forward Error Correction (FEC) in the face of wide variation in loss rate across clients.

In a previous workshop paper [29], we sketched the basic idea of CoopNet (viz., combining multiple distribution trees with MDC) and presented some preliminary analysis. This paper is substantially different in many respects, both in terms of algorithms and in terms of evaluation. The tree management algorithm significantly improves over our previous algorithm. The adaptation framework for MDC based on scalable receiver feedback, the application of MDC to real video data for performance evaluation, and the comparative evaluation of FEC and MDC are new in this paper.

There are some important issues that we do not discuss in this paper. First, we do not discuss the bandwidth heterogeneity issue here. Our longer technical report [28] presents a framework for accomodating bandwidth heterogeneity and congestion control based on our recent work on layered MDC [14]. Second, we do not discuss security issues such as assuring content integrity, maintaining user privacy, and preventing free-riders.

The rest of this paper is organized as follows. In Section II, we present the centralized tree management approach used in CoopNet. We discuss our MDC construction in Section III and the adaptation framework based on scalable receiver feedback in Section IV. We then present a performance evaluation of these in Section V using real video data and the flash crowd traces from MSNBC. We discuss related work in Section VI, and we conclude in Section VII with a summary of our contributions and an outline of our ongoing work.

II. Tree Management

We now discuss the problem of constructing and maintaining the distribution trees. The key challenge is to keep up with the frequent node arrivals and departures that may be typical of flash crowd scenarios. As noted in Section I, we assume that nodes participate and contribute bandwidth resources only for as long as they are interested in receiving content, so they may depart or fail with little notice.

A. Goals and Design Rationale

There are many and sometimes conflicting goals for the tree management algorithm:

1) **Short trees:** The trees should be as short as possible, i.e., have a minimal number of intermediate end-hosts between the root and the leaves. Shortness would minimize the probability of disruption due to the departure, failure, or congestion at an ancestor node. For it to be short, each tree should be balanced and as "bushy" as possible, i.e., the out-degree of each node should be as much as its bandwidth will allow. However, making the out-degree large (and thus consuming more bandwidth) may increase the likelihood of disruption in the CoopNet stream due to competing traffic from other applications.
2) **Tree diversity versus efficiency:** The distribution trees should be diverse, i.e., the set of ancestors of a node in each tree should be as disjoint as possible. The effectiveness of the MDC-based distribution scheme depends critically on the diversity of the distribution trees. However, striving for diversity may interfere with the goal of having efficient trees, i.e., ones whose structure closely matches the underlying network topology. For instance, if we wish to connect three nodes, one each located in New York (NY), San Francisco (SF), and Los Angeles (LA), the structure NY→SF→LA would likely be far more efficient than SF→NY→LA, where → denotes a parent-child relationship. Note that shortness could make a tree more efficient but not necessarily so.
3) **Quick join and leave:** The processing of node joins and leaves should be quick to ensure that an interested node starts receiving streaming content as soon as possible

after it joins (or migrates to a new parent, as discussed below) and with minimal interruption (in case one or more ancestors depart or fail). In particular, the number of network round-trips needed for the joins and leaves to complete should be minimal.
4) **Scalability:** The tree management algorithm should scale to a large number of nodes, with a correspondingly high rate of node arrivals and departures. For instance, in the extreme case of the flash crowd at MSNBC on September 11, the average rate of node arrivals and departures was 180 per second while the peak rate was about 1000 per second (both aggregated over a cluster of streaming servers). While a distributed algorithm might scale better than a centralized one, it is generally at the cost of longer join and leave processing time (i.e., more network round-trips are needed compared to the one needed with centralized tree management).

Some of these goals (appear to) conflict with each other, so we prioritize them as follows. Since resilience is our main objective, we choose to focus on building short and diverse trees with short join and leave times.

We prioritize shortness and diversity over efficiency because in the CoopNet setting, the peer nodes and their often constrained last-hop links are likely to be the causes of disruption. So it makes sense to try to minimize the number of ancestors that a node has and maximize their diversity. And since the live streaming application we consider is non-interactive, a modest delay (from the root to a node) of a few seconds may be acceptable. That said, having efficient trees would likely benefit the network as a whole by reducing bandwidth consumption on the backbone links. So we include efficiency as a secondary goal.

To enable quick joins and leaves, we use a centralized tree management scheme, where a central node (possibly the streaming server) coordinates tree construction and maintenance. We refer to this node as the "root" to connote the probability that it is (or is collocated with) the root of the distribution trees in practice. The root (e.g., the MSNBC server cluster) is often more resourceful and available than the individual clients, so leveraging it greatly simplifies tree management and consequently makes joins and leaves quick. A join or leave operation only requires one or two network round trips — one to the root and possibly one to the new parent.

The dependence on the root means that the system is not self-scaling, but only so with respect to control traffic pertaining to tree management; it is still self-scaling with respect to (the more expensive) data traffic. Thus the load imposed on the server is still greatly reduced compared to the situation today in a client-server setting. Our prototype implementation can keep up with about 400 joins and leaves per second on a laptop with a 2 GHz Mobile Pentium 4 processor. The tree management task is CPU-bound (the memory and network bandwidth requirements are quite low) and should scale with CPU speed. Should the tree management processing on one root node become a bottleneck, it would be easy to scale up using a (possibly distributed) cluster of roots and directing each client to one of the roots, say at random. A client would retain its association with the assigned root until it departs the system. If in addition the aggregate bandwidth of the root nodes (i.e., the source nodes of the data stream) is scaled up, it would result in shorter, and hence better, trees.

Another criticism of centralized tree management might be that the root is a single point of failure. Nonetheless, this may be a moot point in our setting because the root (or a node collocated with it) is also the source of the data stream. So the failure or disconnection of the root is also likely to disrupt the data stream[4].

B. Centralized Tree Management

The root coordinates all tree management functions. When a node wishes to join, it contacts the root, which responds with a designated parent node in each tree. The new node then contacts the parents to have the flow of data started. (Alternatively, the root could directly notify the parent nodes concurrently with its message to the new node, thereby reducing the join time by about an RTT.) When a node leaves gracefully, it informs the root. The root then finds a new parent for the children of the departed node (in each tree) and notifies the children of the identities of their new parents.

In addition, there is the problem of ungraceful leaves where a node departs because of a network disconnection, host crash, or another reason that gives it no opportunity to notify the root or its own children. To accommodate such ungraceful leaves (and general variability in network quality), each node monitors the packet loss rate of the incoming stream on each tree. Losses are deduced from gaps in the packet sequence number, or a stoppage in the packet stream (for instance, because the parent got disconnected).

If the loss rate on a tree exceeds a threshold, the node checks with its parent to see if the parent too is experiencing a high loss rate on that tree. (The network round trip needed for this check can possibly be saved by having the parent piggyback its packet loss rate information on the data stream it forwards to its children.) If the parent is also experiencing a high loss rate, then the cause of the problem is probably upstream of the parent. So the node holds off for a while before checking with its parent again, hoping that the parent (or one of its ancestors) will resolve the problem in the meantime.

If the parent is not experiencing the problem or it fails to respond or resolve the problem, the node contacts the root to request a new parent for itself in the affected tree. In addition to returning a new parent to the requesting node, the root also records the "complaint" against the old parent. Such complaint information could be used to guide future parent selection and possibly scale back the level of participation of the suspect parent, but we do not consider this issue further here. Note that with this protocol, *only the root of the affected subtree would*

[4]This statement is not strictly true because Internet connectivity is not always transitive [4]. A node may lose direct connectivity to the root and hence be unable to exchange tree management messages with it but yet be able to receive the data stream routed via its ancestors (i.e., an overlay path).

contact the server, so there is *not* an implosion of requests at the tree management server.

We now consider the question of how exactly the root chooses the set of parents for a node. We discuss two tree construction algorithms — randomized and deterministic.

1) Randomized Tree Construction: This algorithm was presented in our previous workshop paper [29]. The motivation is simple: since we would like the trees to be diverse, we randomize the process of tree construction within the constraints imposed by node bandwidth and the desire for short trees. The algorithm proceeds as follows. For each tree, we start at the root (i.e., the source of the data stream) and search down the tree until we get to a level that has one or more nodes with spare bandwidth to support a new child. (Note that this search is performed in the local data structures maintained at the root and does not involve any network communication.) We then randomly pick one of these nodes with "room" as the parent of the new node in that tree. To further increase diversity, we could randomly pick the parent from among nodes within K levels of the first level that has room. K would typically be set to a small value such as 1 or 2 to avoid sacrificing too much in terms of the shortness of the tree.

While the total upstream bandwidth consumption at a node aggregated over all trees is equal to the total downstream bandwidth consumption, the upstream and downstream bandwidths on the individual trees may not be equal. A node may have multiple children on one tree and none in others. So the trees will be somewhat bushy.

2) Deterministic Tree Construction: While randomization would result in a degree of tree diversity, the question is whether we can do better. We leverage the insightful observation made in the recent work on SplitStream [11] that the outgoing bandwidth constraint of nodes can be honored by making each node an interior node in just one tree. (That said, there are some crucial differences between SplitStream approach and ours, which are discussed in Section VI-A.) In our setting, the centralization of tree construction makes it relatively easy to honor the bandwidth constraints of each node. But we can use the idea of making each node an interior node in exactly one tree to make the trees more bushy and hence *shorter*. Figure 1 illustrates a simple example where doing so results in shorter trees than if tree construction where randomized.

Making the set of interior nodes in each tree disjoint also contributes to tree diversity and hence robustness. The failure of a single node would only disrupt one tree. However, in the MSNBC scenario considered in Section V, multiple nodes can fail concurrently, so it is not clear to what extent the disjointness of the interior nodes helps.

The deterministic algorithm proceeds as follows. When a new node joins, we first decide the tree in which it is going to be *fertile* (i.e., be an interior node that can have children); the node will be *sterile* (i.e., a leaf node) in all the remaining trees. We keep track of the number of fertile nodes in each tree, and (deterministically) pick the tree with the least number of fertile nodes as the one in which the new node will be fertile

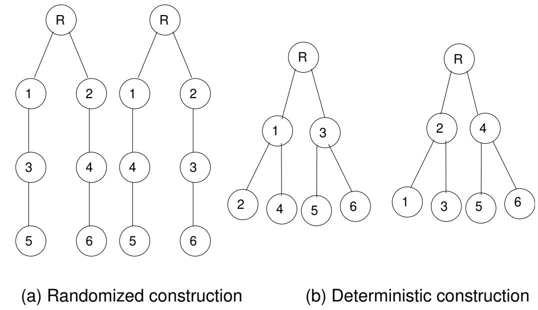

Fig. 1. The (total) out-degree limit for the root (R) is 4 while the limit for the other nodes is 2. By concentrating the out-degree of each node in one tree (its "fertile tree"), deterministic tree construction (case (b)) yields more bushy and hence shorter trees than randomized tree construction (case (a)).

(we term this the "fertile tree" of the node, the rest being its "sterile trees"). The goal is to roughly balance the number of fertile nodes in each tree.

To insert the new node into its fertile tree, we start at the root and proceed down until we reach a level that either has a node with room (i.e., with spare bandwidth) or a node with a sterile child. If a node with room is found at that level, we designate it as the parent. Otherwise, we designate a node with a sterile child as the parent of the new node and find a new parent for the sterile child, as discussed below. (The idea is to have the upper levels of the tree populated by fertile nodes, which can support children.) In both cases, the parent is chosen deterministically (say the first node meeting these criteria that is encountered in the search through our data structures). The disjointness of the interior (i.e., fertile) nodes across the trees makes randomization unnecessary.

To insert the new node into one of its sterile trees, we use a similar procedure as above except that we only consider nodes with spare bandwidth when searching for a parent. Since the new node is sterile in this tree, there is nothing to be gained from substituting an existing sterile node in the upper levels of the tree with the new node.

With this deterministic algorithm, it is possible (although quite unlikely in practice) that a tree runs out of capacity to support new nodes. This can happen, for instance, if a large number of departing nodes all happen to have been fertile in the same tree. When a tree runs out of capacity, we pick a fertile node from the tree with the largest number of fertile nodes and "migrate" it to the tree that is starved of capacity. Migration involves changing the designation of the node from fertile to sterile in one tree (and finding new parents for each of its children in that tree) and designating it as fertile in the starved tree.

Clearly, it would be desirable for both the deterministic and the randomized tree construction algorithms to be network topology-aware. We discuss this issue next.

3) Tree Efficiency/Topology Awareness: As noted in Section II-A, making the trees efficient is a (secondary) goal. The idea is to make the tree structure match the underlying network topology to the extent possible, thereby minimizing duplication of traffic on network links as well as the number of underlying IP hops traversed. Thus, given a choice of parents (subject to the diversity and shortness goals discussed above), we would like to pick a parent that is close in terms of network distance (and perhaps even on the same ISP network to conserve expensive egress bandwidth), where possible. Note that such proximity to parent nodes (in all trees) does not necessarily compromise tree diversity or robustness in the CoopNet setting. Given the high rate of node churn, departures or failures of end-nodes and/or their network links are more likely causes of disruption than failures in the interior of the network. So a set of distinct but nearby parents is still diverse under this failure model.

What we need is an efficient way to pick a proximate parent for a node without requiring extensive P2P network measurements. We use the simple delay-coordinates based "GeoPing" technique proposed in [27] for a somewhat different application (viz., determining the geographic location of Internet hosts). Each node maintains its "delay coordinates" of (average) ping times to a small set of landmark hosts (say 10 hosts). The pings are repeated at a low frequency and the averages recomputed to keep the coordinates up-to-date. When a node wishes to join the distribution trees, it reports its delay coordinates to the root. Once the root has identified a set of candidate parents in a tree (subject to the bandwidth and tree level considerations discussed above), it picks the one whose delay coordinates are closest to that of the new node (in terms of Euclidean distance).

We have conducted a separate study to evaluate the efficacy of the delay coordinates-based approach in finding proximate peers [21]. The results are encouraging — the latency to the peer selected based on delay coordinates is within 31% (1.31X) of the optimal 50% of the time and within 74% (1.74X) of the optimal 90% of the time. The choice based on delay coordinates is far better than that resulting from random selection. However, since we do not have delay coordinates information for the clients in the MSNBC trace, we do not consider proximity in the evaluation presented in this paper.

III. Multiple Description Coding

A. MDC Overview

Multiple description coding (MDC) is a method of encoding an audio and/or video signal into $M > 1$ separate streams, or *descriptions*, such that any subset of these descriptions can be received and decoded. The distortion with respect to the original signal is commensurate with the number of descriptions received; i.e., the more descriptions received, the lower the distortion and the higher the quality of the reconstructed signal. This differs from layered coding[5] in that in MDC every subset of descriptions must be decodable, whereas in layered coding only a nested sequence of subsets must be decodable. For this extra flexibility, MDC incurs a modest performance penalty relative to layered coding (Section III-D), which in turn incurs a slight performance penalty relative to single description coding.

[5]Layered coding is also known as embedded, progressive, or scalable coding.

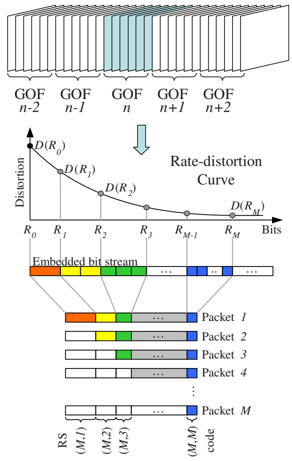

Fig. 2. Priority encoding packetization of a group of frames (GOF). The source bits in the range $[R_{i-1}, R_i)$ are mapped to i source blocks and protected with $M - i$ FEC blocks. Any m out of M packets can recover the initial R_m bits of the bit stream for the GOF.

Many multiple description coding schemes have been investigated over the years. For an overview see [19]. A particularly efficient and practical system is based on layered audio or video coding [30], [24], Reed-Solomon coding [36], priority encoded transmission [3], and optimized bit allocation [17], [33], [26]. In such a system the audio and/or video signal is partitioned into groups of frames (GOFs), each group having a duration of $T = 1$ second or so, for example. Each GOF is then independently encoded, error protected, and packetized into M packets, as shown in Figure 2. Both layered coding and Forward Error Correction (FEC) are building blocks for MDC. Layered coding is used by MDC to prioritize the streaming data. The bits from a GOF are sorted in a decreasing order of importance (where importance is quantified as the bit's contribution towards reducing signal distortion) to form an *embedded bit stream*. For example, bits between R_0 and R_1 are more important than the subsequent bits in the embedded stream in Figure 2. Forward Error Correction (FEC), such as Reed-Solomon encoding, is then used to protect data units to different extents depending on their importance.

M descriptions can accommodate up to M priority levels for a GOF. If any $m \leq M$ packets are received, then the initial R_m bits of the bit stream for the GOF can be recovered, resulting in distortion $D(R_m)$, where $0 = R_0 \leq R_1 \leq \cdots \leq R_M$ and consequently $D(R_0) \geq D(R_1) \geq \cdots \geq D(R_M)$. Thus all M packets are equally important; only the number of received packets determines the reconstruction quality of the GOF. Further, the expected distortion is $\sum_{m=0}^{M} p(m)D(R_m)$, where $p(m)$ is the probability that m out of M packets are received. Given $p(m)$ and the operational rate-distortion function $D(R)$, this expected distortion can be minimized using a simple procedure that adjusts the rate points $R_1, \ldots, R_M$ subject to a constraint on the packet length [17], [33], [26] [6]. By assigning the mth packet in each GOF to the mth description, the entire audio and/or video signal is represented by M descriptions, where each description is a sequence of packets transmitted

[6]The "optimizer" in our system (Figure 3) performs this function.

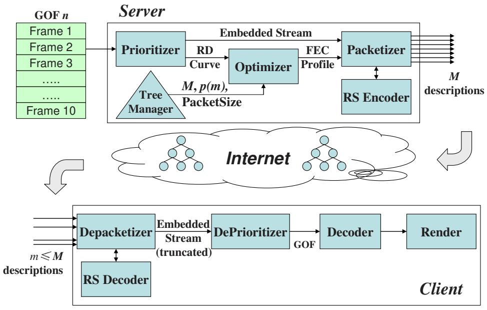

Fig. 3. CoopNet MDC System Architecture

at the rate of 1 packet per GOF. It is simple to generate these optimized M descriptions on the fly [35], assuming that the signal is already coded with a layered codec.

Pure FEC is actually a special case of MDC where all the streaming data is accorded the same priority. Hence, it is much less flexible in adapting to wide variation in packet loss rates across clients, as is likely in the CoopNet setting. We compare MDC and FEC using real video data and real flash crowd traces in Section V-G.

In summary, the independence and priority encoding of the MDC descriptions offers efficient data redundancy needed for robust peer-to-peer media streaming.

B. *CoopNet MDC System Architecture*

In this section, we present the CoopNet MDC system architecture. Figure 3 shows the architecture we have implemented. The input stream is from a layered codec; in our implementation, we use a variant of the PFGS codec (also known as the SMART codec) reported in [37]. The sequence of operations is as follows:

1) Frames in a GOF are partitioned into a set of data units that carry rate-distortion information. The prioritizer prioritizes and sorts these data units according to their contribution towards reducing signal distortion. The larger the reduction per byte a data unit offers, the higher its priority and the greater the protection it is given using FEC encoding. The prioritizer produces an *embedded bit stream* (i.e., the data units sorted by importance), and rate-distortion information (RD Curve). The latter is fed into the optimizer.
2) The tree manager computes the $p(m)$ distribution (i.e., the probability distribution of the number of descriptions received by clients) based on the scalable feedback received from clients (Section IV) and feeds this into the optimizer.
3) Using the number of descriptions (M), the packet size (P), the $p(m)$ distribution, and the RD curve (received from the prioritizer), the optimizer produces a priority encoding profile (FEC profile), $R_1, \ldots, R_M$, for optimal packetization (Figure 2). Thus, *MDC continuously adapts to the incidence of packet loss in the network*, with more redundancy added when packet loss is frequent and vice versa.
4) The packetizer FEC-encodes the embedded stream that is produced by the prioritizer according to the FEC profile from the optimizer, and produces M packets, each with the GOF number (n) recorded in its header (Figure 2). The streaming server distributes these packets over its trees.
5) The M packets traverse multiple CoopNet trees and may experience different amounts of delay in reaching a client. The packets received at a client are synchronized using the GOF number (n) contained in their headers. Due to network congestion or disruption caused by node departures, some packets may be lost, and only a subset of the M packets (descriptions) corresponding to a GOF may be received by a client.
6) Upon receiving a subset of the M packets in a GOF, the de-packetizer at a client FEC-decodes the received packets, and assembles an embedded stream that is a prefix of the original embedded stream generated at the server.
7) The de-prioritizer retrieves the individual data units from the embedded stream and sorts them by their media decode time. The quality of this reconstructed GOF depends on the number of descriptions received.
8) Finally, a media player decodes the GOF and renders it at the client.

C. *Configuring MDC*

A number of parameters affect the MDC construction: the stream bit rate R, the GOF duration G, the total number of descriptions M, the packet size P, and the probability distribution $p(m)$. We fix P to be 1250 bytes, leaving about 250 bytes (in a 1500-byte network packet) for headers and auxiliary information as necessary. Given a desired stream bit rate R and packet size P, M and G are related by $M = GR/P$. For example, for a stream bit rate $R = 160$ Kbps, packet size $P = 1250$ bytes and a GOF duration $G = 1$ second, $M = 16$ descriptions are generated. However, this does not necessarily mean that 16 distribution trees are needed. When the number of trees T is less than M, $\frac{M}{T}$ descriptions are distributed over each tree; e.g., with $T = 8$ trees, 2 descriptions are distributed over each tree.

D. *MDC Tuning and Evaluation*

We did a preliminary evaluation to tune some MDC parameters. Due to space limitations, we only summarize our main findings here; details are contained in our technical report [28].

- The amount of FEC redundancy needed decreases as the number of trees grows, because it becomes less probable that a large fraction of descriptions will be lost. For instance, when the (independent) failure probability of each tree is 2%, 8 trees result in 30% redundancy and 16 trees in 20% redundancy. However, tree management overhead grows with the number of trees, so we settled on 8 trees for our experiments.
- A GOF size of 1 second offers a good compromise of both low coding delay and the ability to accomodate variations in the bit rate across frames in the GOF.

- The computational complexity of our MDC implementation is low. The processing time for a 1-second GOF on a 1.7 GHz P4 desktop PC is as follows: 8-11 ms for prioritization and optimization, and 4-6 ms each for packetization and de-packetization. (The ranges reflect variation across different video clips.)

IV. Scalable Client Feedback

The streaming server periodically gathers client reception information to derive $p(m)$, the probability distribution of receiving m descriptions. This information is fed into the MDC optimizer (Figure 3), allowing adaptation to dynamic network conditions and client population. The $p(m)$ distribution reflects packet loss both due to client churn and network congestion.

Having reports sent directly from the clients to the server would not scale to large numbers of clients. Instead, we use a subset of the distribution trees for propagating and aggregating client reports from the leaves to the root (i.e., the server). (Note that the client reports flow in the direction opposite to the flow of data down the trees.) The use of more than one distribution tree makes the feedback process resilient to packet loss.

In more detail, during each report interval, a client C records a histogram of the number of descriptions received for each GOF. At the end of the report interval, C adds to this histogram the histograms reported by each of its children and sends the accumulated histogram in one report to its parent. This report thus contains a histogram of the number of descriptions received by all clients in the subtree rooted at C. This happens recursively from the leaves to the server (i.e., the root). Finally, the server normalizes the histogram to generate $p(m)$ and feeds it to the MDC optimizer (Figure 3).

With this scalable protocol, *the server only receives feedback from its immediate children* on the one or more trees used to carry feedback information. So there is *no* implosion of feedback messages at the server.

V. Performance Evaluation

We now present our performance evaluation of CoopNet. We first describe the data sets and the experimental methodology used for the evaluation, and then discuss the individual experiments.

A. Methodology and Data Sets

We used our implementations of the tree manager (Section II-B) and the MDC pipeline (Section III-B) to do the performance evaluation. A simulated stream of client joins and leaves based on the MSNBC flash crowd traces is fed into the tree manager. In most of our experiments, packet loss results only from node departures, which impact the descendent nodes for a duration equal to the *repair interval*. In Section V-F, we also consider the impact of network packet drops simulated at the outgoing links of the nodes. For each GOF interval, the tree manager computes the $p(m)$ distribution (Section III-A) corresponding to the number of descriptions received over all clients. This information is then fed into the MDC pipeline, where it is combined with rate-distortion information from real video sequences to continuously reoptimze the MDC construction (Section III-B). We report the average quality of the received stream at the clients, quantified using the Peak Signal-to-Noise Ratio (PSNR) metric, which is computed from the luminance distortion D: $PSNR = 10Log_{10}(255^2/D)$. To give the reader an intuitive feel for PSNR, Figure 4 compares image quality at various PSNR values with the original image.

In our trace-driven evaluation, we have ignored network topology considerations since we have little information besides the IP addresses of clients (in particular, we do not have the "delay coordinates" of the clients (Section II-B.3)). We have conducted a separate study to evaluate the efficacy of the delay coordinates-based approach in finding proximate peers [21].

1) Flash Crowd Trace: In our study, we use a trace of the flash crowd that occurred at MSNBC on Sep 11, 2001. The 1700-second long trace records accesses made by clients to a live 100 Kbps Windows Media stream. The individual clients are identified using a unique "player ID" reported by the Windows Media player. (This helps get around the ambiguity introduced by NATs in IP address-based client identification.) The trace reports the time and duration for which each client was tuned in.

Figure 5 shows the number of clients that simultaneously tuned in to the live stream as a function of time. The peak number of simultaneous clients exceeded 17,000. (The dip around the 1000-second mark is apparently due to a restart of the serving process.) The average rate of node arrivals and departures was 180 per second while the peak rate was about 1000 per second. Over 70% of the clients tuned in to the live stream for less than a minute. We suspect that the short lifetimes were because users were frustrated by the poor quality the video stream received during the flash crowd. If the quality were improved (say using a CoopNet-like approach to relieve the server), client lifetimes may well been longer. This reduction in the churn rate would, in turn, have improved the quality of the stream delivered by CoopNet.

In our simulations, we assume that clients stop participating

Fig. 4. PSNR vs. Perceived Quality: original, 30dB, 25dB, and 23dB.

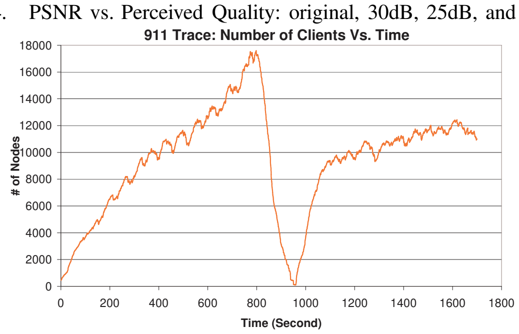

Fig. 5. 911 Trace: Number of Clients Vs. Time

Sequence	Motion-characteristics	Texture-characteristics
Akiyo	Static background, talking head	Easy texture
Foreman	High motion in the first half, almost static in the latter	Relatively easy texture in the first half, relatively detailed in the latter
Stefan	High motion	Detailed texture

TABLE I

CHARACTERISTICS OF THE THREE MPEG TEST SEQUENCES

Parameter	Value
Root bandwidth	20 Mbps
Peer bandwidth	160 Kbps
Stream bandwidth	160 Kbps
Packet size	1250 bytes
GOF duration	1 second
# descriptions	16
# trees	1, 2, 4, 8, 16
Reporting interval	1 second
Repair interval	1, 5, 10 seconds

TABLE II

SIMULATION PARAMETERS

in CoopNet, and stop forwarding traffic, the moment they depart. The departure may have been caused by a machine crash or network disconnection, or a shift in the user's focus to a different stream or a different application that immediately starts consuming the client's limited bandwidth.

2) Video Data: We do not have a recording of the actual video data that was streamed out by MSNBC on Sep 11. In its place, we use three different 10-second QCIF (176x144) standard MPEG test sequences, each encoded at 10 frames per second. Table I lists the characteristics of these video clips.

For our trace-driven evaluation, we continuously replay a clip to decouple variations in quality inherent in the video from variations due to CoopNet dynamics.

3) Parameter Settings: The parameters for our simulation experiments are set as listed in Table II. The stream bandwidth and the outgoing bandwidth available at each node are each set to 160 Kbps. The bandwidth of the root is set to 20 Mbps. With T trees, the stream bandwidth per tree is $\frac{160}{T}$ Kbps. So the total out-degree of a peer node (i.e., the maximum aggregate number of children it can have across all trees) is T and that of the root is $125T$.

The reporting interval is the frequency at which each node feeds back packet loss information ($p(m)$) to its parent, using the scalable feedback protocol discussed in Section IV. In our experiments, we set the reporting interval to 1 second, which is reasonable because the feedback packet is less than 100 bytes in size.

The repair interval is the time it takes for the tree to be repaired after the departure of a node. By default, we set this to 1 second, but we also consider larger settings (5 and 10 seconds) in Section V-E.

Unless indicated otherwise, the results presented are for the Akiyo (news reader) clip with our new deterministic tree construction algorithm (Section II-B.2). We do present some results for the other clips and for the old randomized tree construction algorithm (Section II-B.1).

B. Impact of Number of Distribution Trees

We first consider the benefits of having multiple, diverse distribution trees in the context of the deterministic tree construction algorithm. Figure 6 shows the PSNR (calculated from the distortion averaged across all clients) as a function of time for the cases of 1, 2, 4, 8, and 16 trees. We see that PSNR improves as the number of trees increases. The jump is most significant when we go from 1 tree (i.e., no path diversity) to multiple trees.

The PSNR curves dip around the 800 second point for both the 1-tree and 2-tree cases (and less noticeably for the other cases). This corresponds to the peak in client population (Figure 5) and a high churn rate. A large client population means deeper trees, which increases the likelihood of disruption due to the departure of a node's ancestor(s). Soon after that, PSNR spikes up as the client population drops, to the point where almost all nodes can directly become children of the root and hence experience little disruption.

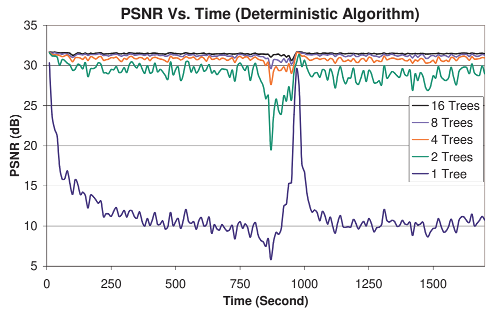

Fig. 6. PSNR (averaged across all clients) versus time for the deterministic tree construction algorithm. The number of trees is varied from 1 to 16.

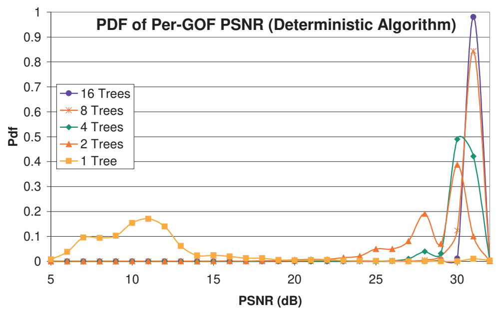

Fig. 7. PDF of the per-GOF PSNR for the deterministic tree construction algorithm. The number of trees is varied from 1 to 16.

Figure 7 presents an alternative view of the same data. For each GOF, we compute the distortion averaged across all clients, and calculate the corresponding PSNR. We then plot the PDF of these per-GOF PSNR values (recall from Table II that the GOF size is set to 1 second, so there are 1700 GOFs during the 1700-second period). As the number of trees increases, the peak of the PDF grows taller and moves to the right, indicating an improvement in PSNR. We also note that 8 trees perform almost as well as 16 trees.

With 8 or 16 trees, almost all the clients receive most or all of the descriptions, thereby achieving high quality. Thus the multiple diverse trees not only improve the average quality across clients but also ensure that few clients experience poor quality.

C. Comparison of the 3 Video Clips

Figure 8 shows a comparison of PSNRs of the three MPEG test sequence video clips across 1700 seconds of the trace. The wide gaps in PSNR across the clips in the 8-tree case result from the different levels of movement in the clips. For a given bit rate, the clips with fast changing scenes that are hard to compress suffer more in quality. The Stefan sequence with 8 trees is barely viewable, while the Akiyo sequence has significantly sharper images. Nevertheless, the important point here is that for all three clips, CoopNet with 8 trees does significantly better than the single tree case, where the video often just freezes.

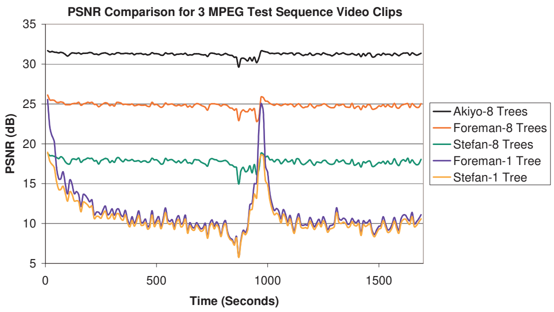

Fig. 8. PSNR Comparison of 3 MPEG Test Sequence Video Clips with Deterministic Algorithm

D. Randomized versus Deterministic Tree Construction

Next, we compare the performance of our old randomized tree construction algorithm (Section II-B.1) and the new deterministic algorithm (Section II-B.2). Figure 9 shows the PDF of the per-GOF PSNR values for the two algorithms when the number of trees is 8. The deterministic algorithm performs significantly better because it is able to construct shorter and also more diverse trees.

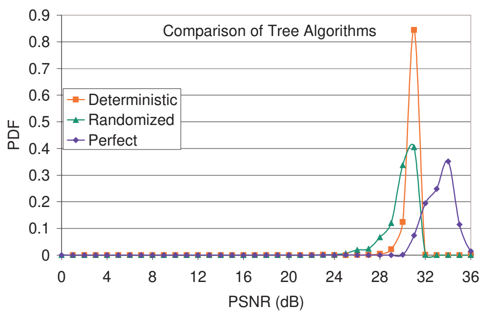

Fig. 9. A comparison of different tree construction algorithms for the case of 8 trees. The "perfect" tree construction algorithm refers to the ideal but impractical case where the trees are constructed afresh from scratch (using the deterministic algorithm) every second.

To quantify the penalty incurred due to *evolutionary* tree construction (i.e., incremental updates as nodes join and leave), we also consider the case where trees are constructed afresh from scratch (using the deterministic algorithm) every second. This is labelled as "perfect" tree construction in Figure 9. Clearly, perfect tree construction is impractical because of the overhead and disruption it would result in, but it provides a useful basis for comparison. From Figure 9, we observe that evolutionary tree construction does incur a significant penalty. The reason is that a skewed sequence of joins and leaves can result in unbalanced trees that are deeper than ideal. In future work, we plan to consider augmenting evolutionary tree construction with selective re-balancing to correct significant skews in the trees, if and when they occur.

E. Impact of Repair Interval

Thus far we have assumed that it takes 1 second for a tree to be repaired following the departure of a node. This may be reasonable for graceful leaves, where the departing node has the opportunity to notify the root of its intention to leave. In this case, the entire repair process takes only 1-2 network round-trips (Section II-B).

However, in the case of an ungraceful leave (say due to a node or network failure), the departing node is unable to notify the root or its own children. The children of the departing node need to infer the departure of their parent based on an upswing in the packet loss rate or a complete stoppage of the packet stream. With our settings of 16 descriptions, GOF duration of 1 second, and 8 trees, only 2 packets are sent down each tree every second. So 1 second is likely too short a duration in which to make a reliable determination of the parent's departure.

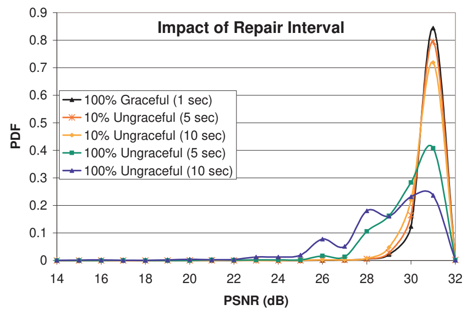

Fig. 10. Impact of the repair interval for deterministic tree construction with 8 trees. We consider repair intervals of 1, 5, and 10 seconds for all leaves. We also consider the case where 90% of the leaves are graceful (with a repair interval of 1 second) while the remain 10% have a repair interval of 5 or 10 seconds.

Therefore, we experiment with longer repair intervals — 5 seconds and 10 seconds — that provide a greater opportunity for failure detection. Figure 10 shows that when all leaves are ungraceful, with a repair interval of 5 or 10 seconds, quality suffers significantly compared to the case where repairs only take 1 second. The reason for this degradation is that the longer the repair interval, the larger the number of concurrent failures and so the higher the likelihood of disruption to the stream received by a client.

It might, however, be reasonable to consider the case where the majority of leaves are graceful (with a repair interval of 1 second) and only a minority are ungraceful, with a longer 5 or 10 second repair interval. Figure 10 shows that when only 10% of the leaves are ungraceful, the quality is almost as good as when all the leaves are graceful. The diversity provided by the 8 distribution trees makes it unlikely for a client to suffer from ungraceful leaves of its ancestors in all trees.

One might wonder why graceful leaves must result in any disruption at all. The point is that graceful leaves may yet be *immediate*. For instance, the leave may be triggered by the user switching to a new stream/channel (e.g., channel surfing during a major news event) or launching another, higher-priority application that immediately starts consuming most or all of a client's bandwidth. Thus while the gracefully departing node might have the opportunity to send a short notification message to the root, it would not, in general, be able to continue forwarding traffic from the old stream.

F. The Impact of Network Packet Loss

Thus far we have only considered distruption caused by client departures and failures. We now evaluate the impact of network packet loss by introducing packet loss at the bandwidth-constrained outgoing links of clients. We experimented with three scenarios: (1) a loss rate of 0.01 on the outgoing links of all clients; (2) a loss rate of 0.1 on the outgoing links of all clients; (3) a loss rate of 0.1 on the outgoing links of 10% of the clients chosen at random. Figure 11 shows the results. While cases (1) and (3) have the same average loss rate over all outgoing links, case (3) has a better PSNR because tree diversity is more effective when losses are concentrated on a few links. As expected, with a high loss rate of 0.1 on all the outgoing links, PSNR degrades significantly.

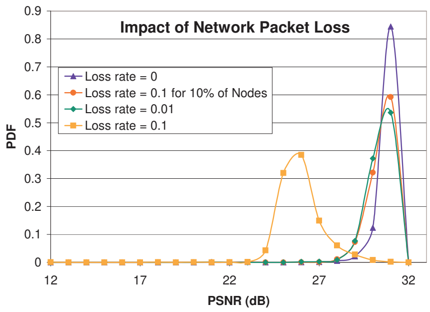

Fig. 11. Impact of Network Packet Loss

G. MDC versus FEC

Pure FEC is a special case of MDC where all data units in a GOF are accorded the same priority. FEC is ideal when all clients experience similar loss rates, but adapts poorly to wide variations in loss rate across clients. We compare FEC to MDC with our trace using 8 trees and the deterministic tree construction algorithm. For each GOF, we measure the amount of redundancy introduced by our adaptive MDC protocol, and use roughly the same amount of redundancy for FEC. For example, if the redundancy of MDC (i.e., the ratio of source bytes plus parity bytes to source bytes) is r , then the number of redundancy packets for pure FEC is set to $f = round(M - \frac{M}{r})$. The FEC configuration is optimal when the loss rate experienced by *all* clients is $\frac{f}{M}$. When the actual loss rate is lower than this threshold, there is unnecessary FEC redundancy that could have been used for more source bytes to improve the streaming quality. When the loss rate is higher, no source bytes can be recovered. MDC exactly addresses this inflexibility by using priority encoding based on a loss distribution rather than a single loss rate. From Figure 12, we can see that MDC yields significantly better PSNR values and is more robust in the face of high client churn.

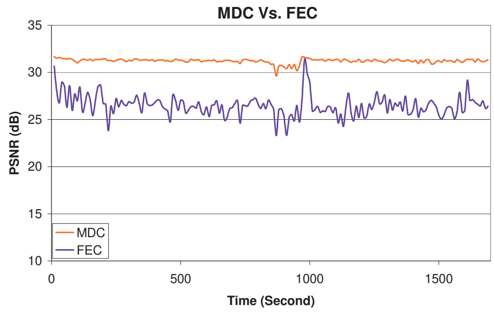

Fig. 12. MDC Vs. FEC

VI. Related Work

The literature relevant to our work spans multiple areas. We discuss work on application-level multicast and that on source coding and path diversity in turn.

A. Application-level Multicast

The deployment of IP multicast [18], especially at the inter-domain level, has been slow due to technical and operational concerns [16]. This has spurred the development of application-level multicast schemes where end-hosts (clients and/or servers) perform the role of "routers".

Narada [16] and Scattercast [13] build application-level meshes formed by connections among a subset of node pairs. The links in the mesh are monitored periodically to improve the quality of the mesh. An efficient application-level multicast tree is formed by running a reverse path forwarding algorithm on the mesh. The choice of link metrics depends on the application. For instance, [15] proposes a combination of bandwidth and latency metrics for a conferencing application.

It is interesting to note that the set of links spanned by the multiple trees in CoopNet can also be viewed as a mesh (although not as carefully optimized as in Narada or Scattercast). However, unlike CoopNet, Narada and Scattercast use a single optimized tree (per source), so the benefits of path diversity are not (fully) realized. Also, these protocols are clearly not designed for large groups (for instance, node arrival and departure information is disseminated to all members of the mesh). However, these could be used in the context of CoopNet for communication among a small, stable set of distributed servers.

An alternative approach is NICE [7], which uses a hierarchy to scale better than a mesh-based protocol. However, NICE is not optimized for a high rate of node churn. Joins require $O(log(N))$ network round-trips, where N is the size of the tree, and disruptions in the tree due to node failures can take up to 30 seconds to heal. In contrast, CoopNet exploits the availability of a stable and resourceful server to optimize these operations.

In ALMI [31] and Overcast [20], a central node coordinates tree management, as in CoopNet. ALMI tries to construct a degree-bounded minimum spanning tree while Overcast seeks a deep tree that maximizes bandwidth to the root. The trees so constructed do not conform to CoopNet's goals of low tree depth and high tree diversity.

Recent work has leveraged the scalable routing substrate provided by distributed hash tables (DHTs) to build efficient multicast trees (e.g., Bayeux [38], Scribe [12]). It is unclear how well these perform in the face of a high rate of node churn, especially since the data structures needed for efficient routing are updated lazily. Furthermore, a fundamental difference compared to CoopNet is that in these systems nodes can be called on to forward traffic even if they are not themselves interested in the data.

All of the above pieces of work differ from CoopNet in that they seek to build a single distribution tree, so issues such as tree diversity are not a consideration.

Snoeren et al. [34] propose a mesh-based content routing scheme using XML. The mesh is constructed to ensure n (overlay) router disjoint paths from the source to each receiver. Duplicate data packets are then sent over these paths, and the receiver discards all but the first copy. Clearly, such duplication would not be feasible when bandwidth is scarce.

SplitStream [11], like CoopNet, advocates the use of multiple distribution trees. The key goal is to evenly distribute forwarding load across the nodes by making a node a leaf in all but one tree. We leverage this idea of interior-node-disjointness in our new, deterministic CoopNet tree construction algorithm (in contrast to our earlier randomized algorithm [29]) but for a different reason — to make the trees as short as possible. SplitStream and CoopNet differ in a fundamental way. SplitStream is built on top of the distributed Scribe protocol [12], which assumes that uninterested nodes will be available to forward traffic. In fact, whether a node is called upon to forward traffic depends only on its node ID and the multicast group ID. It is possible that a node may be assigned more children than it can handle, trying to avoid which may require sacrificing interior-node-disjointness. In contrast, centralized tree management in CoopNet makes it easy to guarantee even load distribution and interior-node-disjointness[7].

Banerjee et al. [8] suggest that any traditional single-tree-based multicast scheme can be made resilient by duplicating packets along a small number of randomly chosen additional overlay links. It is unclear how this resilience mechanism can be made adaptive to the incidence of packet loss in the network. Nevertheless, it would be interesting to compare the resilience of this scheme with that of the multiple-tree plus MDC approach in CoopNet.

B. Source Coding and Path Diversity

Several researchers have advocated the use of source coding, possibly in conjunction with path diversity, to make data transfer robust to packet loss.

Digital Fountain [10] uses Tornado codes (a form of erasure coding) coupled with multiple multicast groups to distribute files scalably to a heterogeneous population of clients. The source transmits the coded blocks repeatedly and clients tune in until they have received a sufficient number of blocks for decoding. Such repeated transmissions, however, are not feasible in our live streaming context.

Byers et al. [9] use the Digital Fountain erasure coding technique and parallel downloads to take advantage of lateral bandwidth between peers (like P2P file sharing systems like KaZaa [1] do). The use of multiple trees in CoopNet also results in a form of parallel download, but the goal is to gain robustness rather than speed and the focus is on live streaming content rather than files. Also, as discussed in Section V-G, MDC offers the advantage of more graceful degradation compared to FEC.

The use of multiple description coding in conjunction with multipath routing in (telephone) networks dates back to the late 1970s [19]. The application of this approach in the context of the Internet has received increasing attention in recent years. Apostopolous et al. [5][6] advocate the use of MDC and path diversity for on-demand streaming from a content distribution network. The idea is for the client to request distinct descriptions from two or more server nodes (akin to parallel downloads). However, path diversity and MDC are of limited help if the last hop to the client rather than the server's network connection is the bottleneck. In contrast, in a P2P setting like CoopNet, the constrained upstream bandwidth at peers and the transience of the peers makes path diversity and MDC advantageous.

Lee et al. [22] present a framework where feedback from an AIMD congestion control protocol (in the form of transmission rate and packet loss profile) is used to optimize an MDC coder. This is related to the MDC adaptation in CoopNet. However, CoopNet focuses on a multicast setting rather than unicast, with a fixed transmission rate (for each layer). In a different paper [32], the same authors present some preliminary ideas on applying MDC in a multicast setting. Their proposal is to have application-level proxies re-encode the stream at bottleneck links. However, such an optimal placement of proxies may be infeasible when the last-hop links to the clients are the bottlenecks. Still, an interesting question for future investigation is how an approach based on re-encoding streams to match the bandwidth of a client group compares with the layered approach advocated by McCanne et al. [25].

[7]In CoopNet, even load distribution (i.e., the in and out bandwidth usage of nodes being equal) is guaranteed whether the trees are interior-node-disjoint or not. Interior-node-distjointness is desirable because it results in the shortest possible trees.

VII. Conclusion

In this paper, we have considered the problem of supporting resilient live streaming using application-layer multicast over an inherently unreliable set of peers. Our motivation is alleviating flash crowds at live streaming servers by recruiting clients to help forward traffic. We make minimal assumptions about the willingness of clients to contribute bandwidth. In particular, we assume that a client will only help forward a stream while it is interested in receiving the stream.

Our solution, CoopNet, provides resilience by introducing redundancy both in network paths (via multiple, diverse distribution trees) and in data (using MDC). A centralized tree management protocol is used to construct short and diverse trees and support quick joins and leaves. A scalable feedback mechanism is used to drive an adaptive MDC optimization algorithm. We have evaluated CoopNet using flash crowd traces from a busy news site couple with real video data. Our results indicate that multiple trees provide a significant improvement in the video quality received by clients (an improvement of up to 22 dB in PSNR in some cases). We also found that MDC outperforms FEC in the face of wide variation in loss rate across clients.

In ongoing work, we are working on a novel parent- and child-driven congestion control scheme that takes advantage of a layered MDC construction we have developed [14] and the tree diversity inherent in CoopNet.

Acknowledgements

We are grateful to Steven Lautenschlager, Ted McConville, and Dave Roth for providing us the MSNBC streaming traces. We would also like to thank Steve Zabinsky for help with the MDC implementation and Ashwin Bharambe for optimizations to the CoopNet tree manager.

References

[1] KaZaa. http://www.kazaa.com.

[2] MSNBC. http://www.msnbc.com.

[3] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. Priority Encoding Transmission. *IEEE Trans. Information Theory*, 42:1737–1744, November 1996.

[4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. ACM SOSP*, October 2001.

[5] J. Apostolopoulos. Reliable Video Communication over Lossy Packet Networks using Multiple State Encoding and Path Diversity. In *VCIP*, January 2001.

[6] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On Multiple Description Streaming with Content Delivery Networks. In *Proc. IEEE INFOCOM*, June 2002.

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

[8] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient Multicast using Overlays. In *Proc. ACM SIGMETRICS*, June 2003.

[9] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proc. ACM SIGCOMM*, August 2002.

[10] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proc. ACM SIGCOMM*, September 1998.

[11] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Content Distribution in a Cooperative Environment. In *Proc. IPTPS*, February 2003.

[12] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE JSAC*, 20(8):100–110, October 2002.

[13] Y. Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, U.C. Berkeley, December 2000.

[14] P. A. Chou, H. J. Wang, and V. N. Padmanabhan. Layered Multiple Description Coding. In *Proc. Packet Video Workshop*, April 2003.

[15] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proc. ACM SIGCOMM*, August 2001.

[16] Y. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proc. ACM SIGMETRICS*, June 2000.

[17] G. Davis and J. Danskin. Joint source and channel coding for image transmission over lossy packet networks. In *Conf. Wavelet Applications to Digital Image Processing*. SPIE, August 1996.

[18] S. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proc. ACM SIGCOMM*, August 1988.

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

[20] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. OSDI*, October 2000.

[21] K. Lakshminarayanan and V. N. Padmanabhan. Some Findings on the Network Performance of Broadband Hosts. In *Proc. ACM/USENIX Internet Measurement Conference*, October 2003.

[22] K. W. Lee, R. Puri, T. Kim, K. Ramchandran, and V. Bharghavan. An Integrated Source Coding and Congestion Control Framework for Video Streaming in the Internet. In *Proc. IEEE INFOCOM*, March 2000.

[23] W. LeFebvre. CNN.com: Facing a World Crisis. Invited talk at the USENIX Technical Conference, June 2002.

[24] Z. Lu and W. A. Pearlman. An Efficient, Low-complexity Audio Coder Delivering Multiple Levels of Quality for Interactive Applications. In *Proc. Workshop on Multimedia Signal Processing*, pages 529–534. IEEE, December 1998.

[25] S. R. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM*, August 1996.

[26] A. E. Mohr, E. A. Riskin, and R. E. Ladner. Unequal Loss Protection: Graceful Degradation of Image Quality over Packet Erasure Channels through Forward Error Correction. *IEEE J. Selected Areas in Communications*, 18(6):819–829, June 2000.

[27] V. N. Padmanabhan and L. Subramanian. Determining the Geographic Location of Internet Hosts. In *Proc. ACM SIGCOMM*, August 2001.

[28] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient Peer-to-Peer Streaming. Technical Report MSR-TR-2003-11, Microsoft Research, Redmond, WA, March 2003.

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

[30] W. A. Pearlman, B.-J. Kim, and Z. Xiong. Embedded Video Subband Coding with 3D SPIHT. In P. Topiwala, editor, *Wavelet Image and Video Compression*. Kluwer, 1998.

[31] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *Proc. USITS*, March 2001.

[32] R. Puri, K. W. Lee, K. Ramchandran, and V. Bharghavan. Application of FEC-based Multiple Description Coding for Internet Video Streaming and Multicast. In *Proc. Packet Video Workshop*, May 2000.

[33] R. Puri and K. Ramchandran. Multiple Description Source Coding Through Forward Error Correction Codes. In *Proc. Asilomar Conference on Signals, Systems, and Computers*. IEEE, October 1999.

[34] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-Based Content Routing Using XML. In *Proc. ACM SOSP*, October 2001.

[35] V. Stanković, R. Hamzaoui, and Z. Xiong. Packet Loss Protection of Embedded Data with Fast Local Search. In *Proc. Int'l Conf. Image Processing*. IEEE, September 2002.

[36] S. B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1995.

[37] F. Wu, S. Li, and Y. Zhang. A Framework for Efficient Progressive Fine Granularity Scalable Video Coding. *IEEE CSVT*, 11(3):332–344, March 2001.

[38] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proc. NOSSDAV*, April 2001.