Distributing Streaming Media Content Using Cooperative Networking*

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

Kunwadee Sripanidkulchai
Carnegie Mellon University
kunwadee@cs.cmu.edu

ABSTRACT

In this paper, we discuss the problem of distributing streaming media content, both live and on-demand, to a large number of hosts in a scalable way. Our work is set in the context of the traditional client-server framework. Specifically, we consider the problem that arises when the server is overwhelmed by the volume of requests from its clients. As a solution, we propose *Cooperative Networking (CoopNet)*, where clients cooperate to distribute content, thereby alleviating the load on the server. We discuss the proposed solution in some detail, pointing out the interesting research issues that arise, and present a preliminary evaluation using traces gathered at a busy news site during the flash crowd that occurred on September 11, 2001.

Categories and Subject Descriptors

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

General Terms

Design, Performance, Measurement

Keywords

Streaming media, content distribution networks, peer-to-peer networks, multiple description coding

1. INTRODUCTION

There has been much work in recent years on the topic of content distribution. This work has largely fallen into two categories: (a) infrastructure-based content distribution, and (b) peer-to-peer content distribution. An infrastructure-based content distribution network (CDN) (e.g., Akamai) complements the server in the traditional client-server framework. It employs a dedicated set of machines to store and distribute content to clients on behalf of the server. The dedicated infrastructure, including machines and networks links, is engineered to provide a high level of performance guarantees. On the other hand, peer-to-peer content distribution relies on clients to host content and distribute it to other clients. The P2P model replaces rather than complements the client-server framework. Typically, there is no central server that holds content. Examples of P2P content distribution systems include Napster and Gnutella.

In this paper, we discuss Cooperative Networking (CoopNet), an approach to content distribution that combines aspects of infrastructure-based and peer-to-peer content distribution. Our focus is on distributing streaming media content, both live and on-demand. Like infrastructure-based content distribution, we seek to complement rather than replace the traditional client-server framework. Specifically, we consider the problem that arises when the server is overwhelmed by the volume of requests from its clients. For instance, a news site may be overwhelmed because of a large "flash crowd" caused by an event of widespread interest, such as a sports event or an earthquake. A home computer that is webcasting a birthday party live to friends and family might be overwhelmed even by a small number of clients because of its limited network bandwidth. In fact, the large volume of data and the relatively high bandwidth requirement associated with streaming media content increases the likelihood of the server being overwhelmed in general. Server overload can cause significant degradation in the quality of the streaming media content received by clients.

CoopNet addresses this problem by having clients cooperate with each other to distribute content, thereby alleviating the load on the server. In the case of on-demand content, clients cache audio/video clips that they viewed in the recent past. During a period of overload, the server redirects new clients to other clients that had downloaded the content previously. In the case of live streaming, the clients form a distribution tree rooted at the server. Clients that receive streaming content from the server in turn stream it out to one or more of their peers.

The key distinction between CoopNet and pure P2P systems like Gnutella is that CoopNet complements rather than replaces the client-server framework of the Web. There is still a server that hosts content and (directly) serves it to clients. CoopNet is only invoked when the server is unable to handle the load imposed by clients. The presence of a central server simplifies the task of locating content. In contrast, searching for content in a pure P2P system entails an often more

*For more information, including an extended version of this paper [13], please visit the CoopNet project Web page at http://www.research.microsoft.com/~padmanab/projects/CoopNet/.

NOSSDAV'02, May 12-14, 2002, Miami, Florida, USA.

expensive distributed search [17, 18, 20].

Individual clients may only participate in CoopNet for a short period of time, say just a few minutes, which is in contrast to the much longer participation times reported for systems such as Napster and Gnutella [19]. For instance, in the case of live streaming, a client may tune in for a few minutes during which time it may be willing to help distribute the content. Once the client tunes out, it may no longer be willing to participate in CoopNet. This calls for a content distribution mechanism that is robust against interruptions caused by the frequent joining and leaving of individual peers.

To address this problem, CoopNet employs *multiple description coding (MDC)*. The streaming media content, whether live or on-demand, is divided into multiple sub-streams using MDC and each sub-stream is delivered to the requesting client via a different peer. This improves robustness and also helps balance load amongst peers.

The rest of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we discuss the operation of CoopNet for live and on-demand content, and present an outline of multiple description coding. In Section 4, we use traces from the flash crowd that occurred on September 11, 2001 to evaluate how well CoopNet would have performed for live and on-demand content. We present our conclusions in Section 5.

2. RELATED WORK

As noted in Section 1, two areas of related work are infrastructure-based CDNs and peer-to-peer systems. Infrastructure-based CDNs such as Akamai employ a dedicated network of thousands of machines in distributed locations, often with leased links inter-connecting them, to serve content on behalf of servers. When a client request arrives (be it for streaming media or other content), the CDN redirects the client to a nearby replica server. The main limitation of infrastructure-based CDNs is that their cost and scale is only appropriate for large commercial sites such as CNN and MSNBC. A second issue is that it is unclear how such a CDN would fare in the face of a large flash crowd that causes a simultaneous spike in traffic at many or all of the sites hosted by the CDN.

Peer-to-peer systems such as Napster and Gnutella depend on little or no dedicated infrastructure[1]. There is, however, the implicit assumption that the individual peers participate for a significant length of time (for instance, [19] reports a median session duration of about an hour both for Napster and for Gnutella). In contrast, CoopNet seeks to operate in a highly dynamic situation such as a flash crowd where an individual client may only participate for a few minutes. The disruption that this might cause is especially challenging for streaming media compared to static file downloads, which is the primary focus of Napster and Gnutella. The short lifetime of the individual nodes poses a challenge to distributed search schemes such as CAN [17], Chord [20], Pastry [18], and Tapestry [22].

Work on application-level multicast (e.g., ALMI [14], End System Multicast [3], Scattercast [2]) is directly relevant to the live streaming aspect of CoopNet. CoopNet could benefit from the efficient tree construction algorithms developed in previous work. Our focus here, however, is on using real traces to evaluate the efficacy of CoopNet. Thus we view our work as complementing existing work on application-level multicast. We also consider the on-demand streaming case, which does not quite fit in the application-level multicast framework.

Existing work on distributed streaming (e.g., [9]) is also directly relevant to CoopNet. A key distinction of our work is that we focus on the distruption and packet loss caused by node arrivals and departures, which is likely to be significant in a highly dynamic environment. Using traces from the September 11 flash crowd, we are able to evaluate this issue in a realistic setting.

Systems such as SpreadIt [5], Allcast [23] and vTrails [24] are perhaps closest in spirit to our work. Like CoopNet, they attempt to deliver streaming content using a peer-to-peer approach. SpreadIt differs from CoopNet is a couple of ways. First, it uses only a single distribution tree and hence is vulnerable to disruptions due to node departures. Second, the tree management algorithm is such that the nodes orphaned by the departure of their parent might be bounced around between multiple potential parents before settling on a new parent. In contrast, CoopNet uses a centralized protocol (Section 3.3), which enables much quicker repairs.

It is hard for us to do a specific comparison with Allcast and vTrails, in the absence of published information.

3. COOPERATIVE NETWORKING (COOPNET)

In this section, we present the details of CoopNet as it applies to the distribution of streaming media content. We first consider the live streaming case, where we discuss and analyze multiple description coding (MDC) and distribution tree management. We then turn to the on-demand streaming case.

3.1 Live Streaming

Live streaming refers to the synchronized distribution of streaming media content to one or more clients. (The content itself may either be truly live or pre-recorded.) Therefore multicast is a natural paradigm for distributing such content. Since IP multicast is not widely deployed, especially at the inter-domain level, CoopNet uses application-level multicast instead.

A *distribution tree* rooted at the server is formed, with clients as its members. Each node in the tree transmits the received stream to each of its children using unicast. The out-degree of each node is constrained by the available outgoing bandwidth at the node. In general, the degree of the root node (i.e., the server) is likely to be much larger than that of the other nodes because the server is likely to have a much higher bandwidth than the individual client nodes.

One issue is that the peers in CoopNet are far from being dedicated servers. Their ability and willingness to participate in CoopNet may fluctuate with time. For instance, a client's participation may terminate when the user tunes out of the live stream. In fact, even while the user is tuned in to the live stream, CoopNet-related activity on his/her machine may be scaled down or stopped immediately when the user initiates other, unrelated network communication. Machines can also crash or become disconnected from the network.

With a single distribution tree, the departure or reduced availability of a node has a severe impact on its descendants. The descendants may receive no stream at all until the tree has been repaired. This is especially problematic because node arrivals and departures may be quite frequent in flash crowd situations. To reduce the disruption caused by node

[1]Napster has central servers, but these only hold indices, not content.

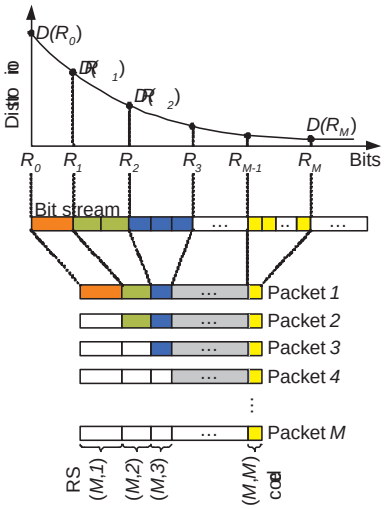


Figure 1: Priority encoded packetization of a group of frames (GOF). Any m out of M packets can recover the initial R_m bits of the bit stream for the GOF.

departures, we advocate having multiple distribution trees spanning a given set of nodes and transmitting a different MDC description down each tree. This would diminish the chances of a node losing the entire stream (even temporarily) because of the departure of another node. We discuss this further in Section 3.2.

The distribution trees need to be constantly maintained as new clients join and existing ones leave. In Section 3.3, we advocate a centralized approach to tree management, which exploits the availability of a resourceful server node, coupled with client cooperation, to greatly simplify the problem.

3.2 Multiple Description Coding (MDC)

Multiple description coding is a method of encoding the audio and/or video signal into $M > 1$ separate streams, or *descriptions*, such that any subset of these descriptions can be received and decoded into a signal with distortion (with respect to the original signal) commensurate with the number of descriptions received; that is, the more descriptions received, the lower the distortion (i.e., the higher the quality) of the reconstructed signal. This differs from layered coding[2] 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, which in turn incurs a slight performance penalty relative to single description coding.

Several multiple description coding schemes have been investigated over the years. For an overview see [6]. A particularly efficient and practical system is based on layered audio or video coding [15, 7], Reed-Solomon coding [21], priority encoded transmission [1], and optimized bit allocation [4, 16, 8]. In such a system the audio and/or video signal is partitioned into groups of frames (GOFs), each group having duration $T = 1$ second or so. Each GOF is then independently encoded, error protected, and packetized into M packets, as shown in Figure 1.

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 distortion-rate 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 [4, 16, 8]. By sending 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 at rate 1 packet per GOF.

It is a very simple matter to generate these optimized M descriptions on the fly, assuming that the signal is already coded with a layered codec.

3.2.1 CoopNet Analysis: Quality During Multiple Failures

Let us consider how multiple description coding achieves robustness in CoopNet. Suppose that the server encodes its AV signal into M descriptions as described above, and transmits the descriptions down M different distribution trees, each rooted at the server. Each of the distribution trees conveys its description to all N destination hosts. Ordinarily, all N destination hosts receive all M descriptions. However, if any of the destination hosts fail (or leave the session), then all of the hosts that are descendents of the failed hosts in the mth distribution tree will not receive the mth description. The number of descriptions that a particular host will receive depends on its location in each tree relative to the failed hosts. Specifically, a host n will receive the mth description if none of its ancestors in the mth tree fail. This happens with probability $(1-\epsilon)^{A_n}$, where A_n is the number of the host's ancestors and ϵ is the probability that a host fails (assuming independent failures). If hosts are placed at random sites in each tree, then the unconditional probability that any given host will receive its mth description is the average $\theta_N = (1/N)\sum_{n=1}^{N}(1-\epsilon)^{A_n}$ across all hosts in the tree. Thus the number of descriptions that a particular host will receive is randomly distributed according to a Binomial(M, θ_N) distribution, i.e., $p(m) = \binom{M}{m}\theta_N^m(1-\theta_N)^{M-m}$. Hence for large M, the fraction of descriptions received is approximately Gaussian with mean θ_N and variance $\theta_N(1-\theta_N)$. This can be seen in Figure 2, which shows (in bars) the distribution $p(m)$ for various values of $M = 2, 4, 8, 16$ and $N = 10, 1000, 100000$. In the figure, to compute θ_N we assumed balanced binary trees with N nodes and probability of host failure $\epsilon = 1\%$. Note that as N grows large, performance slowly degrades, because the depth of the tree (and hence $1-\theta_N$) grows like $\log_2 N$.

The distribution $p(m)$ can be used to optimize the multiple description code by choosing the rate points $R_0, R_1, \ldots, R_M$ to minimize the expected distortion $\sum_{m=0}^{M} p(m)D(R_m)$ subject to a packet length constraint. Figure 2 shows (in lines), the quality associated with each $p(m)$, measured as SNR in dB, i.e., $10\log_{10}(\sigma^2/D(R_m))$, as a function of the number of received descriptions, $m = 0, 1, \ldots, M$. In the figure, to compute the rate points $R_0, R_1, \ldots, R_M$ we assumed an operational distortion-rate function $D(R) = \sigma^2 2^{-2R}$, which is asymptotically typical for any source with variance σ^2, where R is expressed in bits per symbol, and we assumed a packet length constraint given as $R = 8$.

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

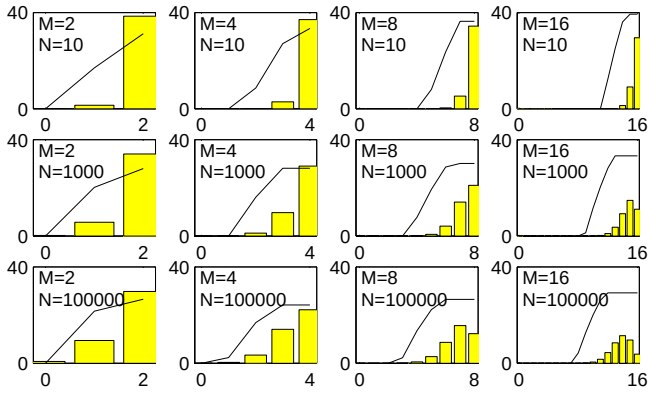


Figure 2: SNR in dB (line) and probabililty distribution (bars) as a function of the number of descriptions received, when the probability of host failure is $\epsilon = 1\%$.

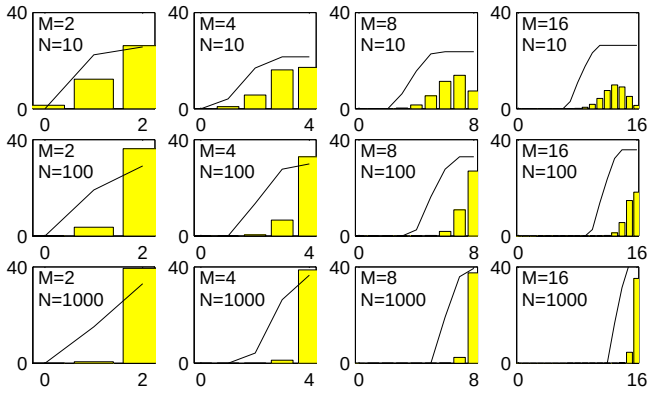


Figure 3: SNR in dB (line) and probabililty distribution (bars) as a function of the number of descriptions received during the failure of a single host.

3.2.2 *CoopNet Analysis: Quality During Single Failure*

The time it takes to repair the trees is called the repair time. If ϵ of the hosts fail during each repair time, then the average length of time that a host participates in the session is $1/\epsilon$ repair times. When the number of hosts is small compared to $1/\epsilon$, then many repair times may pass between single failures. In this case, most of the time all hosts receive all descriptions, and quality is excellent. Degradation occurs only when a single host fails. Thus, it may be preferable to optimize the MDC system by minimizing the distortion expected during the repair interval in which the single host fails, rather than minimizing the expected distortion over all time. To analyze this case, suppose that a single host fails randomly. A remaining host n will not receive the mth description if the failed host is an ancestor of host n in the mth tree. This happens with probability $A_n/(N-1)$, where A_n is the number of ancestors of host n. Since hosts are place at random sites in each tree, the unconditional probability that any given host will receive its mth description is the average $\theta_N = (1/N)\sum_{n=1}^{N}(1 - A_n/(N-1))$. Thus the number of descriptions that a particular host will receive is randomly distributed according to a Binomial(M, θ_N) distribution. Equivalently, the expected number of hosts that receive m descriptions during the failure is $(N-1)p(m)$, where $p(m) = \binom{M}{m}\theta_N^m(1-\theta_N)^{M-m}$. This distribution can be used to optimize the multiple description code for the failure of a single host. Figure 3 illustrates this distribution and the corresponding optimized quality as a function of the number of descriptions received, for $M = 2, 4, 8, 16$ and $N = 10, 100, 1000$. Note that as M increases, for fixed N, the distribution again becomes Gaussian. One implication of this is that the expected number of hosts that receive 100% of the descriptions decreases. However it is also the case that the expected number of hosts that receive fewer than 50% of the descriptions decreases, resulting in an increase in quality on average. Further, as N increases, for fixed M, performance becomes nearly perfect, since $\theta_N \geq 1 - \log_2 N/N$, which goes to 1. However, for large N, it becomes increasingly difficult to repair the trees before a second failure occurs.

3.2.3 *Further Analyses*

These same analyses can be extended to d-ary trees. It is not difficult to see that for $d \geq 2$, a d-ary trees with $N^{\log_2 d} \geq N$ nodes has the same height, and hence the same performance, as a binary tree with only N nodes. Thus when each node has a large out-degree, i.e., when each host has a large uplink bandwidth, much larger populations can be handled. Interestingly, the analysis also applies when $d = 1$. So, if each host can devote only as much uplink bandwidth as its downlink video bandwidth (which is typically the case for modem users), then the descriptions can still be distributed peer-to-peer by arranging the hosts in a chain, like a bucket brigade. It can be shown that when the order of the hosts in the chain is random and independent for each description, then for a single failure the number of hosts receiving m out of M descriptions is binomially distributed with parameters M and θ_N, where $\theta_N = (N+1)/2N$. Although this holds for any N, it is most suitable for smaller N. For larger N, it may not be possible to repair the chains before other failures occur. In fact, as N goes to infinity, the probability that any host receives any descriptions goes to zero.

In this section we have proposed optimizing the MDC system to the unconditional distribution $p(m)$ derived by averaging over trees and hosts. Given any set of trees, however, the distribution of the number of received descriptions varies widely across the set of hosts as a function of their upstream connectivity. By optimizing the MDC system to the unconditional distribution $p(m)$, we are not minimizing the expected distortion for any given host, but rather minimizing the sum of the expected distortions across all hosts, or equivalently, minimizing the expected sum of the distortions over all hosts.

3.3 Tree Management

We now discuss the problem of constructing and maintaining the distribution trees in the face of frequent node arrivals and departures. There are many (sometimes conflicting) goals for the tree management algorithm:

1. **Short and wide tree:** The trees should be as short as possible so as to minimize the latency of the path from the root to the deepest leaf node and to minimize the probability of disruption due to the departure of an ancestor node. For it to be short, the tree should be balanced and as wide 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 may leave little bandwidth for non-CoopNet (and higher priority) traffic emanating from the node. Interference due to

such traffic could cause a high packet loss rate for the CoopNet streams.

2. **Efficiency versus tree diversity:** The distribution trees should be efficient in that their structure should closely reflect the underlying network topology. So, 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 (→ denotes a parent-child relationship). However, striving for efficiency may interfere with the equally important goal of having diverse distribution trees. The effectiveness of MDC-based distribution scheme described in Section 3.2 depends critically on the diversity of the distribution trees.

3. **Quick join and leave:** The processing of node joins and leaves should be quick. This would ensure that the interested nodes would receive the streaming content as quickly as possible (in the case of a join) and with minimal interruption (in the case of a leave). However, the quick processing of joins and leaves may interfere with the efficiency and balanced tree goals listed above.

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 deparurtes was 180 per second while the peak rate was about 1000 per second.

With these requirements in mind, we now describe our approach to tree construction and management. We first describe the basic protocol and then discuss optimizations.

3.3.1 Basic Protocol

We exploit the presence of a resourceful server node to build a simple and efficient protocol to process node joins and leaves. While it is centralized, we argue that this protocol can scale to work well in the face of extreme flash crowd situations such as the one that occurred on September 11. Despite the flash crowd, the server is not overloaded since the burden of distributing content is shared by all peers. Centralization also simplifies the protocol greatly, and consequently makes joins and leaves quick. In general, a criticism of centralization is that it introduces a single point of failure. However, in the context of CoopNet, the point of centralization is the server, which is also the source of data. If the source (server) fails, it may not really matter that the tree management also breaks down. Also, recall from Section 1 that the goal of CoopNet is to complement, not replace, the client-server system.

The server has full knowledge of the topology of all of the distribution trees. When a new node wishes to join the system, it first contacts the server. The new node also informs the server of its available network bandwidth to serve furture downstream nodes. The server responds with a list of designated parent nodes, one per distribution tree. The designated parent node in each tree is chosen as follows. Starting at the server, we work our way down the tree until we get to a level where there are one or more nodes that have the necessary spare capacity (primarily network bandwidth) to serve as the parent of the new node. (The server could itself be the new parent if it has sufficient spare capacity, which it is likely to have during the early stages of tree construction.) The server then picks one such node at random to be the designated parent of the new node. This top-down procedure ensures a short and largely balanced tree. The randomization helps make the trees diverse. Upon receiving the server's message, the new node sends (concurrent) messages to the designated parent nodes to get linked up as a child in each distribution tree. In terms of messaging costs, the server receives one message and sends one. Each designated parent receives one message and sends one (an acknowledgement). The new node sends and receives $M + 1$ messages, where M is the number of MDC descriptions (and hence distribution trees) used.

Node departures are of two kinds: graceful departures and node failures. In the former case, the departing node informs the server of its intention to leave. For each distribution tree, the server identifies the children of the departing node and executes a join operation on each child (and implicitly the subtree rooted at the child) using the top-down procedure described above. The messaging cost for the server would at most be $\sum_i d_i$ sends and $\sum_i d_i$ receives, where d_i is the number of children of the departing node in the ith distribution tree. (Note that the cost would be somewhat lower in general because a few of the children may be in common across multiple trees.) Each child sends and receives $M + 1$ messages. To reduce its messaging load, the server could make the determination of the designated parent for each child in each tree and then leave it to another node (such as the departing node, if it is still available) to convey the information to each child. In this case, the server would have to send and receive just one message.

A node failure corresponds to the case where the departing node leaves suddenly and is unable to notify either the server or any other node of its departure. This may happen because of a computer crashing, being turned off, or becoming disconnected from the network. We present a general approach for dealing with quality degradation due to packet loss; node failure is a special case where the packet loss rate experienced by the descendants of the failed node is 100%. Each node monitors the packet loss rate it is experiencing in each distribution tree. When the packet loss rate reaches an unacceptable level (a threshold that needs to be fine-tuned based on further research), a node contacts its parent to check if the parent is experiencing the same problem. If so, the source of the problem (network congestion, node failure, etc.) is upstream of the parent and the node leaves it to the parent to deal with it. (The node also sets a sufficiently long timer to take action on its own in case its parent has not resolved the problem within a reasonable period of time.) If the parent is not experiencing a problem or it does not respond, the affected node will contact the server and execute a fresh join operation for it (and its subtree) to be moved to a new location in the distribution tree.

3.3.2 Optimizations

We now discuss how to make the distribution trees efficient, as discussed above. The basic idea here is to preferentially attach a new node as the child of an existing node that is "nearby" in terms of network distance (i.e., latency). The definition of "nearby" needs to be broad enough to accomodate significant tree diversity. When trying to insert a new node, the server first identifies a (sufficiently large) subset of nodes that are close to the new node. Then using the random-

ized top-down procedure discussed in Section *3.3.1*, it tries to find a parent for the new node (in each tree) among the set of nearby nodes. Using this procedure, it is quite likely that many of the parents of the new node (on the the various distribution trees) will be in the same vicinity, which is beneficial from an efficiency viewpoint. We argue that this also provides sufficient diversity since the primary failure mode we are concerned with is node departures and node failures. So it does not matter much that all of the parents may be located in the same vicinity (e.g., same metropolitan area).

To determine the network distance between two nodes, we use a technique previously proposed in [12], [17], and [10]. Each node determines its network "coodinates" by measuring the network latency to a set of landmark hosts. The server keeps track of the coordinates of all nodes currently in the system and determines whether two nodes are proximate by comparing their coordinates.

3.3.3 *Feasibility of the Centralized Protocol*

The main question regarding the feasibility of the centralized tree management protocol is whether the server can keep up. To answer this question, we consider the September 11 flash crowd at MSNBC, arguably an extreme flash crowd situation. At its peak, there were 18,000 nodes in the system and the rate of node arrivals and departures was 1000 per second.[3] (The average numbers were 10000 nodes and 180 arrivals and departures per second.) In our calculations here, we assume that the number of distribution trees (i.e., the number of MDC descriptions) is 16 and that on average a node has 4 children in a tree. We consider various resources that could become a bottleneck at the server (we only focus on the impact of tree management on the server):

- **Memory:** To store the entire topology of one tree in memory, the server would need to store as many pointers as nodes in the system. Assuming a pointer size of 8 bytes (i.e., a 64-bit machine) and auxiliary data of 24 bytes per node, the memory requirement would be about 576 KB. Since there are 16 trees, the memory requirement for all trees would be 9.2 MB. In addition, for each node the server needs to store its network coordinates. Assuming this is a 10-dimensional vector of delay values (2 bytes each), the additional memory requirement would be 360 KB. So the total memory requirement at the server would be under 10 MB, which is a trivial amount for any modern machine.

- **Network bandwidth:** Node departures are more expensive than node arrivals, so we focus on departures. The server needs to designate a new parent in each distribution tree for each child of the departing node. Assuming that nodes are identified by their IP addresses (16 bytes assuming IPv6) and that there are 4 children per tree on average, the total amount of data that the server would need to send out is 1 KB. If there are 1000 departures per second, the bandwidth requirement would be 8 Mbps. This is likely to be a small fraction of the network bandwidth at a large server site such as MSNBC.

- **CPU:** Node departure involves finding a new set of parents for each child of the departing node. So the CPU cost is roughly equal to the number of children of the departing node times the cost of node insertion. To insert a node, the server has to scan the tree levels starting with the root until it reaches a level containing one or more nodes with the spare capacity to support a new child. The server picks one such node at random to be the new parent. Using a simple array data structure to keep track of the nodes in each level of the tree that have free capacity, the cost of picking a parent at random can be made (a small) constant. Since the number of levels in the tree is about $log(N)$, where N is the number of nodes in the system, the node insertion cost (per tree) is $O(log(N))$. (With $N = 18,000$ and an average of 4 children per node, the depth of the tree will be about 9.)

  A departure rate of 1000 per second would result in 64,000 insertions per second (1000 departures times 4 children per departing node times 16 trees). Given that memory speed by far lags CPU speed, we only focus on how many memory lookups we can do per insertion. Assuming a 40 ns memory cycle, we are allowed about 390 memory accesses per insertion, which is likely to be more than sufficient.

In general, the centralized approach can be scaled up (at least in terms of CPU and memory resources) by having a cluster of servers and partitioning the set of clients across the set of server nodes.

We are in the process of benchmarking our implementation to confirm the rough calculations made above.

3.4 On-demand Streaming

We now turn to on-demand streaming, which refers to the distribution of pre-recorded streaming media content on demand (e.g., when a user clicks on the corresponding link). As such, the streams corresponding to different users are not synchronized. When the server receives such a request, it starts streaming data in response if its current load condition permits. However, if the server is overloaded, say because of a flash crowd, it instead sends back a response including a short list of IP addresses of clients (peers) who have downloaded (part or all of) the requested stream and have expressed a willingness to participate in CoopNet. The requesting client then turns to one or more of these peers to download the desired content. Given the large volume of streaming media content, the burden on the server (in terms of CPU, disk, and network bandwidth) of doing this redirection is quite minimal compared to that of actually serving the content. So we believe that this redirection procedure will help reduce server load by several orders of magnitude.

While the procedure described above is similar to one that might apply to static file content, there are a couple of important differences arising from the streaming nature of the content. First, a peer may only have a part of the requested content because, for instance, the user may have stopped the stream halfway or skipped over portions. So in its initial handshake with a peer, a client finds out which part of the requested content is available at the peer and accordingly plans to make requests to other peers for the missing content, if any.

A second issue is that, as with the live streaming case, peers may fail, depart, or scale back their participation in CoopNet

[3] One reason for the high rate of churn may be that users were discouraged by the degradation in audio/video quality caused by the flash crowd, and so did not stay for long. However, we are not in a position to confirm that this was the case.

at any time. In contrast with file download, the time-sensitive nature of streaming media content makes it especially susceptible to such disruptions. As a solution, we propose the use of *distributed streaming* where a stream is divided into a number of substreams, each of which may be served by a different peer. Each substream corresponds to a *description* created using MDC (Section 3.2). Distributed streaming improves robustness to disruptions caused by the untimely departure of peer nodes and/or network connectivity problems with respect to one or more peers. It also helps distribute load more evenly among peers.

4. PERFORMANCE EVALUATION

We now present a performance evaluation of CoopNet based on simulations driven by traces of live and on-demand content served by MSNBC on September 11, 2001.

4.1 Live Streaming

We evaluate the MDC-based live streaming design using traces of a 100kbps live stream. The trace started at 18:25 GMT (14:25 EST) and lasted for more than one hour (4000 seconds).

4.1.1 Trace Characteristics

Figure 4 shows the time series of the number of clients simultaneously tuned in to the live stream. The peak number of simultaneous clients exceeds 17,000. On average, there are 84 clients departing every second. (We are unable to definitely explain the dip around the 1000-seond mark, but it is possibly due to a glitch in the logging process.) Over 70% of the clients remain tuned in to the live stream for less than a minute. We suspect that the short lifetimes could be because users were frustrated by the poor quality the video stream during the flash crowd. If the quality were improved (say using CoopNet to relieve the server), client lifetimes may well become longer. This, in turn, would increase the effectiveness of CoopNet.

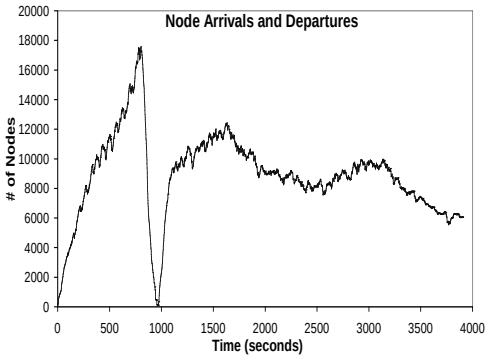


Figure 4: Number of clients and departures.

4.1.2 Effectiveness of MDC

We evaluate the impact of MDC-based distribution (Section 3.2) on the quality of the stream received by clients in the face of client departures. When there are no departures, all clients receive all of the MDC descriptions and hence perceive the full quality of the live stream.

We have conducted two simulation experiments. In the first experiment, we construct completely random distribution trees at the end of the repair interval following a client departure. We then analyze the stream quality received by the remaining clients. The random trees are likely to be diverse (i.e., uncorrelated), which improves the effectiveness of MDC-based distribution. In the second experiment, we simulate the tree management algorithm described in Section 3.3. Thus the distribution trees are evolved based on the node arrivals and departures recorded in the trace. We compare the results of these two experiments at the end of the section.

M	100%	[87.5,100)	[75,87.5)	[50,75)	[25,50)	0
1	98.1	0	0	0	0	1.90
2	94.80	0	0	5.05	0	0.16
4	89.54	0	9.24	1.13	0.09	0.005
8	82.07	14.02	3.19	0.70	0.016	0
16	71.26	25.11	3.26	0.37	0.002	0

Table 1: Random Tree Experiment: probability distribution of descriptions received vs. number of distribution trees

In more detail, we conducted the random tree experiment as follows. For each repair interval, we construct M distribution trees (corresponding to the M descriptions of the MDC coder) spanning the N nodes in the system at the beginning of the interval. Based on the number of departing clients, d, recorded through the end of the repair interval, we randomly remove d nodes from the tree, and compute the number of descriptions received by the remaining nodes. The perceived quality of the stream at a client is determined by the fraction of descriptions received by that client. The set of distribution trees is characterized by three parameters: the number of trees (or, equivalently, descriptions), the maximum out-degree of nodes in each tree, and the out-degree of the root (i.e., the live streaming server). The out-degree of a node is typically a function of its bandwidth capacity. So the root (i.e., the server) tends to have a much larger out-degree than bandwidth-constrained clients. In our random tree construction, each client is assigned a random degree subject to a maximum. We varied the degree of the root and the number of descriptions to study their impact on received stream quality. We set the repair time to 1 second; we investigate the impact of repair time in Section *4.1.3*.

Table 1 shows how the number of distribution trees, M, affects the fraction of descriptions received (expressed as a percentage, P). We compute the distribution of P by averaged across all client departures. We set the maximum out-degree of a client to 4 and the root degree to 100. We vary the number of descriptions among 1, 2, 4, 8, or 16. Each column represents a range of values of P. For each pair of the range and number of descriptions, we list the average percentage of clients that receive at that level of quality. For example, the first table entry indicates that when using 2 descriptions, 94.80% of clients receive 100% of the descriptions (i.e., both the descriptions).

As the number of descriptions increases, the percentage of clients that receive the all of the descriptions (i.e., $P = 100\%$) decreases. Nonetheless, the percentage of clients corresponding to small values of P decreases dramatically. With 8 descriptions, 96% (82.07% + 14.02%) of clients receive more than 87.5% of the descriptions. For both 8 and 16 descriptions, all clients receive at least one description. Figure 5 shows the corresponding SNR. Figure 6 compares the SNR over time for the 16-description case and the single descrip-

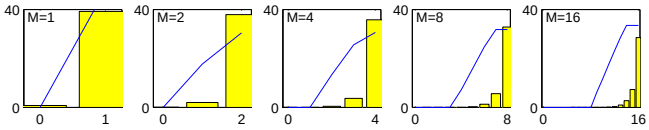


Figure 5: Random Tree Experiment: SNR in dB (line) and probabililty distribution (bars) as a function of the number of descriptions received

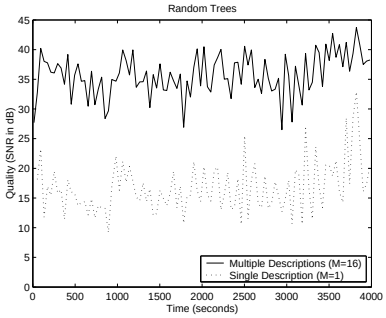


Figure 6: Random Tree Experiment: The SNR over time for the MDC and SDC cases. At each time instant, we compute the average SNR over all clients.

tion (SDC) case. MDC demonstrates a clear advantage over SDC.

In our second experiment, we evolved the distribution trees by simulating the tree management algorithm from Section 3.3. We set the root (i.e., server) out-degree to 100. The maximum out-degree of a client is set to 4. Table 2 shows the probability distribution of the descriptions received upon client departures. Figure 7 shows the corresponding SNR. The results are comparable to those of the random tree experiment. This suggests that our tree management algorithm is able to preserve significant tree diversity even over a long period of time (more than an hour in this case).

4.1.3 *Impact of Repair Time*

Finally, we evaluate the impact of the time it takes to repair the tree following a node departure. Clearly, the longer the repair time, the greater is the impact on the affected nodes. Also, a longer repair time increase the chances of other nodes departing before the repair is completed, thereby causing further disruption.

We divide time into non-overlapping repair intervals and assume that all leaves happen at the beginning of each interval. We then compute fraction of descriptions received averaged over all nodes (this is the quantity θ_N discussed in Section 3.2). As in Section 3.2, assume a balanced binary tree at all times.

M	100%	[87.5,100)	[75,87.5)	[50,75)	[25,50)	0
1	98.34	0	0	0	0	1.66
2	96.5	0	0	3.42	0	0.08
4	93.3	0	6.31	0.36	0.03	0
8	87.14	11.34	1.29	0.20	0.02	0
16	77.26	21.62	0.99	0.11	0.01	0

Table 2: Evolving Tree Experiment: probability distribution of descriptions received vs. number of distribution trees

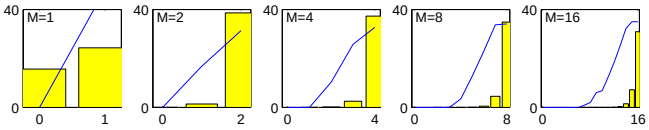


Figure 7: Evolving Tree Experiment: SNR in dB (line) and probabililty distribution (bars) as a function of the number of descriptions received

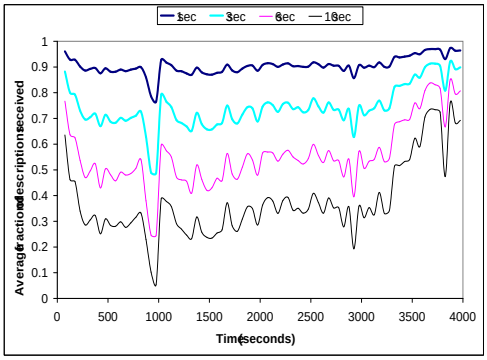


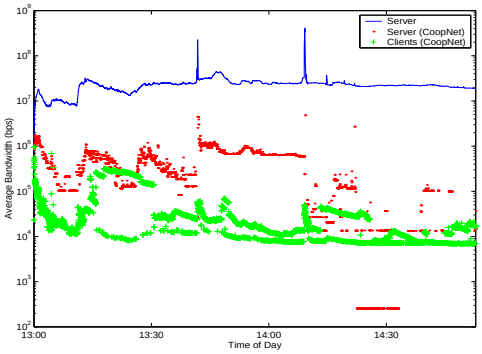
Figure 8: The average fraction of descriptions received for various repair times.

Figure 8 shows the average number of descriptions received as a function of time for four different settings of repair time: 1, 3, 6, and 10 seconds. With a repair time of 1 second, clients would receive 90% of the descriptions on average. With a 10 second repair time, the fraction drops to 30%. We believe that these results are encouraging since in practice tree repair can be done very quickly, especially given that our tree management algorithm is centralized (Section 3.1). Even a 1-second repair interval would permit multiple round-trips between the server and the nodes affected by the repair (e.g., the children of the departed node).

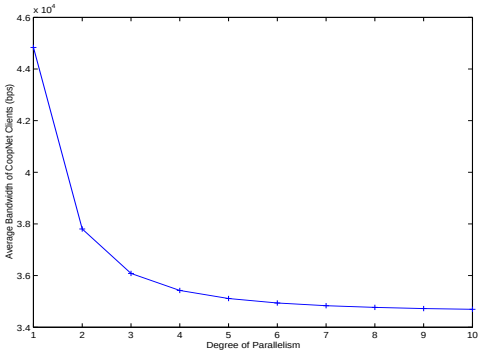
4.2 On-Demand Streaming

We now evaluate the potential of CoopNet in the case of on-demand streaming. The goals of our evaluation are to study the effects of client cooperation on load reduction at the server and additional load incurred by cooperating peers.

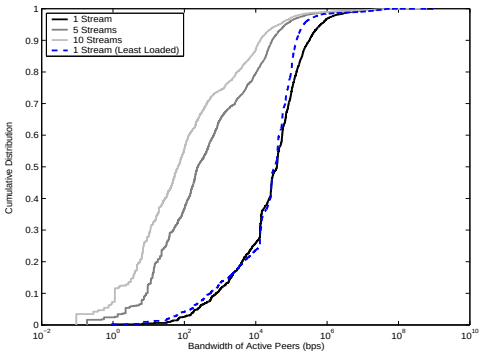
The cooperation protocol used in our simulations is based on server redirection as in [11]. The server maintains a fixed-size list of IP addresses (per URL) of CoopNet clients that have recently contacted it. To get content, a client initially sends a request to the server. If the client is willing to cooperate, the server redirects the request by returning a short list of IP addresses of other CoopNet clients who have recently requested the same file. In turn, the client contacts these other CoopNet peers and arranges to retrieve the content directly from them. Each peer may have a different portion of the file, so it may be necessary to contact multiple peers for content. In order to select a peer (or a set of peers when using distributed streaming) to download content from, peers run a greedy algorithm that picks out the peer(s) with the longest portion of the file from the list returned by the server. If a client cannot retrieve content through any peer, it retrieves the entire content from the server. Note that the server only provides the means for discovering other CoopNet peers. Peers independently decide who they cooperate with. The server maintains a list of 100 IP addresses per URL, and



(a) Average bandwidth at server and cooperating peers.



(b) Average bandwidth at peers when using distributed streaming.



(c) Distribution of bandwidth at active peers.

Figure 9: Performance of CoopNet for on-demand streaming.

returns a list of 10 IP addresses in the redirection messages in our simulations.

We use traces collected at MSNBC during the flash crowd of Sep 11, 2001 for our evaluation. The flash crowd started at around 1:00 pm GMT (9:00 am EDT) and persisted for the rest of the day. The peak request rate was three orders of magnitudes more than the average. We report simulation results for the beginning of the flash crowd, between 1:00 pm to 3:00 pm GMT. There were over 300,000 requests during the 2-hour period. However, only 6% or 18,000 requests were successfully served at an average rate of 20 Mbps with a mean session duration of 20 minutes. Unsuccessful requests were not used in the analysis because of the lack of content byte-range and session duration information.

4.2.1 *Bandwidth Load*

In our evaluation, load is measured as bandwidth usage. We do not model available bandwidth between peers. We assume that peers can support the full bit rate (56 kbps, 100 kbps) of each encoded stream. We also do not place a bound on the number of concurrent connections at each peer. In practice, finding peers with sufficient available bandwidth and not overloading any one peer are important considerations, and we are investigating these issues in ongoing work.

Figure 9(a) depicts the bandwidth usage during the 2-hour period for two systems: the traditional client-server system, and the CoopNet system. The vertical axis is average bandwidth and the horizontal axis is time. There are two peaks at around 1:40 pm and 2:10 pm, when two new streams were added to the server. In the client-server system, the server was distributing content at an average of 20 Mbps. However, client cooperation can reduce that bandwidth by orders of magnitude to an average of 300 kbps. As a result, the server is available to serve more client requests. The average bandwidth contribution that CoopNet clients need to make to the system is 45 kbps. Although the average bandwidth contribution is reasonably small, peers are not actively serving content all the time. We find that typically less than 10% of peers are active at any second. The average bandwidth contribution that *active* CoopNet peers need to make to the system is as high as 465 kbps, where average bandwidth of active peers is computed as the total number of bits served over the total length of peers' active periods.

To further reduce load at individual CoopNet clients, disjoint portions of the content can be retrieved in parallel from multiple peers using distributed streaming (Section 3.4). (The bandwidth requirement placed on each peer is correspondingly reduced.) Figure 9(b) depicts the average bandwidth contributed versus the degree of parallelism. The degree of parallelism is an upper-bound on the number of peers that can be used in parallel. For example, clients can retrieve content from up to 5 peers in parallel in a simulation with a degree of parallelism of 5. The actual number of peers used in parallel may be less than 5 depending on how many peers can provide content in the byte-range needed by the client. The load at each active peer is reduced as the degree of parallelism increases. When the degree of parallelism is 5, peers are serving content at only 35 kbps. However, the bandwidth of active peers (not depicted in this figure) is only slightly reduced to 400 kbps. This is because the large amount of bandwidth required to serve content during the two surges at 1:40 pm and 2:10 pm influence the average bandwidth.

The cumulative distribution of bandwidth contributed by active CoopNet peers, depicted in Figure 9(c), illustrates the impact of distributed streaming on bandwidth utilization. Each solid line represents the amount of bandwidth peers contribute when using 1, 5, and 10 degrees of parallelism. The median bandwidth requirement is 39 kbps when content is streamed from one peer, and only 66 bps for 10 degrees of parallelism. The bandwidth requirement imposed on each peer is reduced as the degree of parallelism increases. Although this reduction is significant, a small portion of peers still contribute more than 1 Mbps even when using 10 degrees of parallelism. We believe that the combination of the following two factors contribute to the wide range in bandwidth usage: the greedy algorithm a client uses to select peers and the algorithm the server uses to select a set of IP addresses to give to clients.

For better load distribution, the server can run a load-aware algorithm that redirects clients to recently seen peers that are the least loaded (in terms of network bandwidth usage). In order to implement this algorithm, the server needs to know the load at individual peers. Therefore, peers constantly re-

port their current load status to the server. We use a report interval of once every second in our simulations. Because the server caches a fixed-size list of IP addresses, only those peers currently in the server's list need to send status updates. Given this information, the server then selects the 10 least loaded peers that have recently accessed the same URL as the requesting client to return in its redirection message. This algorithm replaces the one described earlier in this section where the server redirects clients to peers that were recently seen. Clients, however, use the same greedy algorithm to select peers. We find that using this new algorithm, active clients serve content at 385 kbps. The dashed line in Figure 9(c) depicts the cumulative distribution of bandwidth contributed by CoopNet clients when the load-aware algorithm is used at the server. In this simulation, clients stream content from at most one other peer (degree of parallelism of 1). For the most part, the distribution is similar to the one observed when the server redirects the request to recently seen peers. The difference lies in the tail end of the distribution. About 6% of peers contributed more than 500 kbps of bandwidth when the server runs the original algorithm, compared to only 2% when the server runs the load-aware algorithm. In addition, the total number of active peers in the system doubles when the load-aware algorithm is used.

We find that client cooperation significantly reduces server load, freeing up bandwidth to support more client connections. In addition, the combination of distributed streaming and a load-aware algorithm used by the server further reduces the load on individual peers.

5. CONCLUSIONS

In this paper, we have presented CoopNet, a peer-to-peer content distribution scheme that helps servers tide over crisis situations such as flash crowds. We have focussed on the application of CoopNet to the distribution of streaming median content, both live and on-demand. One challenge is that clients may not participate in CoopNet for an extended length of time. CoopNet employs distributed streaming and multiple description coding to improve the robustness of the distributed streaming content in face of client departures.

We have evaluated the feasibility and potential performance of CoopNet using traces gathered at MSNBC during the flash crowd that occurred on September 11, 2001. This was an extreme event even by flash crowd standards, so using these traces helps us stress test the CoopNet design. Our results suggest that CoopNet is able to reduce server load significantly without placing an unreasonable burden on clients. For live streams, using multiple independent distribution trees coupled with MDC improves robustness significantly.

We are currently building a prototype implementation of CoopNet for streaming media distribution. We are also investigating distributed algorithms for tree management.

Acknowledgements

We are grateful to Steven Lautenschlager, Ted McConville, and Dave Roth for providing us the MSNBC streaming media logs from September 11. We would also like to thank the anonymous NOSSDAV reviewers for their feedback.

6. REFERENCES

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

[2] Y. Chawathe. "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service", Ph.D. Dissertation, University of California at Berkeley, Dec 2000.

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

[4] 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*, Denver, CO, August 1996. SPIE.

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina. "Streaming Live Media over a Peer-to-Peer Network", Technical Report, Stanford University, August 2001. *http://dbpubs.stanford.edu:8090/pub/2001-31*

[6] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, pages 74–93, September 2001.

[7] 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, Redondo Beach, CA, December 1998. IEEE.

[8] 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.

[9] T. P. Nguyen and A. Zakhor. "Distributed Video Streaming over the Internet", *Multimedia Computing and Networking*, January 2002.

[10] T. S. E. Ng and H. Zhang. "Towards Global Network Positioning", *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[11] V. N. Padmanabhan and K. Sripanidkulchai. "The Case for Cooperative Networking", *IPTPS*, March 2002.

[12] V. N. Padmanabhan and L. Subramanian. "An Investigation of Geographic Mapping Techniques for Internet Hosts", *ACM SIGCOMM*, August 2001.

[13] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. "Distributing Streaming Media Content Using Cooperative Networking", Microsoft Research Technical Report, MSR-TR-2002-37, April 2002.

[14] D. Pendarkis, S. Shi, D. Verma, and M. Waldvogel. "ALMI: An Application Level Multicast Infrastructure", *USITS*, March 2001.

[15] 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.

[16] R. Puri and K. Ramchandran. Multiple description source coding through forward error correction codes. In *Proc. Asilomar Conference on Signals, Systems, and Computers*, Asilomar, CA, October 1999. IEEE.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network", *ACM SIGCOMM*, August 2001.

[18] A. Rowstron and P. Druschel. "Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility", *ACM SOSP*, October 2001.

[19] S. Saroiu, P. K. Gummadi, and S. D. Gribble. "A Measurement Study of Peer-to-Peer File Sharing Systems", *Multimedia Computing and Networking*, January 2002.

[20] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications", *ACM SIGCOMM*, August 2001.

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

[22] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing", Technical Report UCB/CSD-01-1141, University of California at Berkeley, April 2001.

[23] Allcast. *http://www.allcast.com/*

[24] vTrails. *http://www.vtrails.com/*