Achieving Scalable and Efficient Video-on-Demand Over Multicast

Ramaprabhu Janakiraman Marcel Waldvogel Wei Deng Lihao Xu

***Abstract*— Server bandwidth has been identified as a major bottleneck in large Video-on-Demand (VoD) systems. Using multicast delivery to serve popular content helps increase scalability by making efficient use of server bandwidth. In addition, recent research has focused on *proactive* schemes in which the server periodically multicasts popular content without explicit requests from clients. Proactive schemes are attractive because they consume bounded server bandwidth irrespective of client arrival rate.**

In this work, we describe *Fuzzycast*, a scalable periodic multicast scheme that uses simple techniques to provide video on demand at reasonable client start-up times while consuming optimum server bandwidth. We start with a theoretically optimum scheme for providing scalable multicast video-on-demand and analyze its performance. We go on to consider a series of issues that are of both theoretical and real-world importance, including support for variable-bitrate (VBR) media and optimum transmission over multiple multicast groups.

***Index Terms*— Video on demand, media on demand, multicast, proactive, shuffling**

I. Introduction

The promise of universal broadband networks and fast cheap computation has triggered active research and popular interest in Video-on-Demand (VoD). However, experience with traditional VoD systems has revealed significant limiting factors: server bandwidth tends to become swamped by requests for popular videos, forcing providers to invest in expensive resources to ensure acceptable quality of service under peak load.

Earlier work [3] on requests in video rentals suggests that a variation of the well-known 80:20 rule might hold here as well: 80% of the requests are for the top 20 movies. Applying this knowledge to the design of VoD system, this fact suggests that multicast delivery can help significantly reduce server loads by concurrently serving popular content to multiple clients.

However, clients in a VoD system, unlike television broadcast audiences, choose their own schedules: plain broadcast alone will not suffice. On the other hand, dedicating a channel to each client quickly uses up server bandwidth. Many of the efficient VoD systems compromise by periodically rebroadcasting content to satisfy the different movie start times requested by clients.

The following metrics are important in assessing VoD performance: the first three are driven by user demand, the rest by technology limits.

MEDIA QUALITY: Users expect at least the media quality that they routinely get on cable television and rented videos.

PLAYOUT QUALITY: Playout should be reasonably free of glitches and skipped frames. This depends on the network connectivity and computational power available to the client.

This paper is a significant revision and extension of our results published at Globecom 2001 [1] and IEEE Infocom 2002 [2].

STARTUP DELAY: While VoD should strictly be instantaneous, most discussions on broadcast-based VoD systems allow for reasonable client wait times between requesting a video and commencement of playout.

BANDWIDTH USAGE: Bandwidth is multi-faceted, ranging from the requirements at the server or client attachment to the overall load on the Internet Service Provider (ISP) network used by the VoD system. From a scalability perspective, server bandwidth usage is the most critical metric, but the others aspects should not be neglected.

BUFFER SPACE: The most efficient VoD systems transmit video segments out of order. Clients need computers or set-top boxes with large amounts of buffer space to cache out-of-order segments from arrival until their playout and to smooth playout jitter introduced by the network. In some broadcast-based schemes, peak buffer requirements run to several megabytes.

With storage cost rapidly dropping, the crucial trade-off in building scalable VoD systems appears to be that of server bandwidth usage vs. client startup delay. Recent research has therefore focused on ways to minimize the server bandwidth required to achieve a given startup delay and vice versa.

Proactive multicast protocols [4] are especially attractive in terms of server bandwidth usage [5–9]. These protocols "push" popular content periodically without explicit requests from clients, so that server bandwidth usage remains bounded and is essentially independent of client demand. However, current proactive protocols have their own drawbacks: The most efficient protocols use a *fluid* model [8, 10] in which data is segmented and multicast in parallel over many constant-rate bit streams. This view is conceptually appealing but difficult to sustain in practice: video data consists of individual *frames* that are transmitted over the network in discrete packets. The complexity involved in overlaying multiple time-sensitive, constant bandwidth bit streams on a best-effort packet network will be a significant obstacle in deploying these protocols.

In this work, we discuss *Fuzzycast*—a proactive multicast scheme that takes an alternative, *discrete* frame-oriented approach to periodic multicast of video data. We demonstrate that using a discrete approach results in a feasible and practical VoD system without sacrificing the bandwidth efficiency of optimum but infeasible designs.

The remainder of this paper is organized as follows: In Section II, we introduce and analyze *harmonic broadcasting*, the theoretical ideal for proactive VoD schemes, and explain why its existing approximations either are infeasible in practice or inefficient in design. In Section III, we describe *Fuzzycast*, a practicable and efficient version of harmonic broadcasting, and evaluate its performance. In Section IV, we consider the impact of various effects such as limited client buffers and variable-bit-rate

Algorithm 1 IDEAL

```
1: for all frames f_j do
2:     λ ← j + w;
3:     for (t ← λ; t ≤ t_max; t+ = λ) do
4:         transmit (t, f_j);
```

(VBR) media on its performance and outline simple extensions to address these issues. In Section V, we propose the problem of optimally partitioning a transmission over a small number of multicast groups, and show that it is a special case of a commonly encountered resource tradeoff—one that we have labeled "Scottie's dilemma"—and solve the problem in its general form, before, in Section VI, applying it to Fuzzycast and analyzing performance. In Section VII, we briefly describe a prototype VoD system that was built using these techniques. In Section VIII we discuss alternative solutions before concluding in Section IX.

II. Towards Optimality

A. Definitions

This work applies to a VoD system that comprises a central server distributing digital media to clients over a network that supports a bandwidth-efficient broadcast primitive, such as satellite broadcast or Internet Protocol (IP) multicast. We use the terms "broadcast" and "multicast" interchangeably throughout this paper, except in Section V where we assume an ability to join and leave multicast groups.

The server stores a set of *movies* from which each client is free to choose. A movie comprises blocks of data (*frames*) which, for convenience of explanation and without loss of generality, are assumed to be transmitted atomically in network packets. Except in Section IV-B, which is devoted to VBR, we assume frames to be of fixed size.

Time is discrete and measured in *instants*; an instant is defined to be the playout time of a single frame. Bandwidth is measured in frames per instant. Clients arrive at times of their choosing, request the server for movies, and after a given initial waiting period of w instants, consume their movies from beginning to end, thus spending $w + n$ instants on a movie of n frames. We shall neglect client decoding time and network-introduced delay in our analysis, as they will be negligible compared to typical startup delays w. As a result, a frame transmitted at time t_0 will be available for playout at the beginning of $t_0 + 1$.

B. Harmonic Broadcasting

Consider the broadcast of a popular movie of n frames. Assume the frames are to be broadcast to satisfy the on-demand requirements of multiple clients with different join times. Now, a client with a join time of t and a wait time of w will require frame f at time t_f no later than during playout time $t+w+f-1$, i.e., $t \le t_f < t + w + f$. Thus each client has a window of $w + f$ instants in which to receive frame f. In the absence of client feedback, i.e., in a proactive system, on-demand delivery for each client is ensured by broadcasting frame f at least once every $w + f$ instants. Most of the existing work expands on this simple result, known as *Harmonic Broadcasting* [6].

This is formalized as algorithm IDEAL (Algorithm 1) below. The schedule generated by IDEAL (with $w = 1$) is plotted in Fig. 1(a), showing the frames transmitted during each instant and the receive windows for two clients joining at instants 1 and 4. In this example, we assume a *transmit* system call that schedules frame f for transmission at instant t using a transmission queue.

Theorem 1 On average, IDEAL consumes server bandwidth and client bandwidth of $\log \frac{n+w}{w}$ frames/instant.

Proof: Each frame f is scheduled once in $w + f$ instants and hence occupies an average bandwidth of $1/(w + f)$ frames/instant. Thus the average bandwidth for the entire movie is

$$B = \sum_{f=1}^{n} \frac{1}{w+f} \approx \log \frac{n+w}{w}, \tag{1}$$

where B is normalized to the playout bandwidth of the movie. In other words,

$$\text{Bandwidth (in frames/instant)} \approx \log \frac{\text{Movie length}}{\text{Initial delay}}$$

where the $\log$ function refers to the natural logarithm. ∎

In practical terms, serving a 2-h 300-kbps Real Media or MPEG-4 movie with a 5-min initial delay requires a server and client bandwidth of ≈1 Mbps. Thus, the system begins to be advantageous as soon as the number of clients exceeds 3. Fig. 1(b) shows the scaled bandwidth usage (relative to the bit rate of the movie) as a function of the initial delay (relative to the length of the movie).

Theorem 2 For a client with a waiting time w between arrival and playout, IDEAL

- delivers all data on time, and
- has the least server bandwidth for any pure proactive scheme.

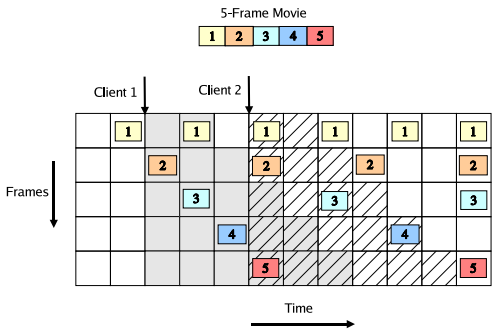
Proof: It is easy to prove that IDEAL is optimum in the sense that a frequency of $1/(w + f)$ instants for frame f is both necessary and sufficient for on-demand data delivery: necessary because an interval of $w + f$ instants without frame f beginning at time t would cause a client starting at t to miss f; sufficient because, in the absence of interactive functions such as *fast forwarding*, each client is guaranteed to play out frame f no earlier than $w + f$ instants after joining. ∎

Theorem 3 IDEAL requires a peak client buffer space of about $1/e \approx 37\%$ of the movie length, where e is the base of the natural logarithm.

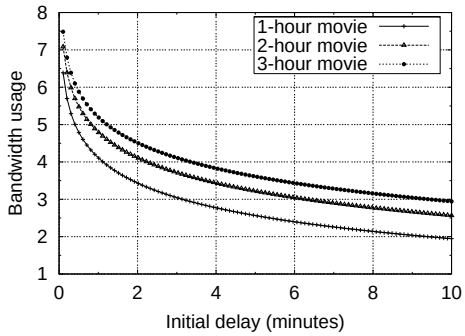
Proof: The probability $p(f, t)$ that frame f has reached the client by time t ($t < f + w$) relative to the start of the session is $\frac{t}{w+f}$. The transmit time of any frame is calculated without reference to any other frame, making the arrival probabilities independent. Expected buffer space at time t can thus be calculated as the cumulative probability $\sum_{f=t}^{n} p(f, t)$. Buffer requirements at the client side are therefore given by

$$B(t) = \begin{cases} t \log \frac{n+w}{w} & 1 \le t \le w, \\ t \log \frac{n+w}{t} & w \le t \le n + w \end{cases} .$$

This has a maximum $\max(B) = \frac{n+w}{e}$ at time $\frac{n+w}{e}$. As $w \ll n$, $\max(B) \approx 0.37n$. ∎



(a) Basic Transmission Pattern



(b) Bandwidth vs. Delay

Fig. 1. These figures show the scheduling and performance of an optimum multicast transmission scheme. Fig. 1(a) shows frames scheduled over time. Note that clients that join at any of the time instants shown will retrieve all the frames in time for playout, but not necessarily in order. Fig. 1(b) shows the critical bandwidth-delay tradeoff. Here the bandwidth is shown normalized to the rate of the movie, i.e., in frames per instant.

C. Existing Approaches

Although IDEAL is simple, elegant, and optimum, a fatal flaw renders it unusable in its original form. The number of frames scheduled for transmission at time t is the number of integers $w \leq i \leq w+n$ such that i divides t. This function is extremely spiky, varying from ≤ 2 for prime values of t to record highs when t is *highly composite* [11]. Owing to the resulting bandwidth spikiness, earlier research [8, 9] discounted IDEAL as a theoretical limit rather than as a practicable scheme.

Existing protocols, notably the harmonic broadcasting protocols [6, 7], have taken a stream-based approach to avoid the limitations of IDEAL. Stream-based protocols, rather than transmitting frame (or an arbitrary-sized *segment* of the movie) f every $f+w$ instants, transmit it continuously in a separate channel or stream of bandwidth $1/(f+w)$. This ensures uniform bandwidth usage, but also encounters difficulties: In stream-based protocols, the initial delay is a function of the segment size. Because user acceptance considerations dictate that initial delay be small compared with the movie length, these protocols transmit a movie over many concurrent streams. For example, Polyharmonic Broadcasting [10] transmits a single 2-h movie with a 5 min initial delay over 96 streams, with bandwidths varying from a few hundred kbps to a few hundred bps. However, this merely defers responsibility down the network stack because streams ultimately map to network packets. Because packets cannot be arbitrarily small, low-bandwidth streams will have to be aggregated, returning to the original problem of infeasible schedules.

Finally, unless error-correction techniques such as Forward Error Correction (FEC) [12] are used, stretching the transmission of a segment over an extended period and over multiple packets increases the probability that a frame becomes unusable due to partial loss or corruption in transit. This is further amplified by error propagation as part of the decompression process. A recent solution for this problem using *un-equal protection (UEP) codes* is discussed in [13].

Another stream-based protocol, Pagoda broadcasting [14], attempts to pack segments into a few fixed-rate channels deterministically, but sacrifices performance in the process because it has to settle for suboptimum schedules (Fig. 2(c)).

Algorithm 2 BASIC

```
1: B_est ← B_act ← 0;
2: for all frames f_j do
3:     λ ← j + w;
4:     B_est += 1/λ;
5:     for (t ← λ;  t ≤ t_max;  t += δ_t, B_act[t]++) do
6:         δ_t ← FIND-NEIGHBOR;
7:         transmit (t + δ_t, f_j);
```

III. Computing Feasible Frame Schedules

As a proactive scheme, the only flaw of IDEAL is that it results in non-uniform bandwidth usage. We rectify this as follows: Whenever a frame f has to be scheduled at an instant that has used up the bandwidth allotted to frames $1 \cdots f$, we allow it to 'drift' heuristically from its scheduled position to a neighboring time slot that can spare some of *its* allotted bandwidth. The aim is to spread out or 'smear' a bandwidth peak over time—flattening peaks and filling up troughs—without significantly changing the optimum schedule.[1]

This is formalized as algorithm BASIC (Algorithm 2). The crux of it is the FINDNEIGHBOR function, which finds an alternative placement in a neighboring time slot for frames that IDEAL would have scheduled in relatively 'crowded' time slots.

At this point, we pause to distinguish between advancing a frame and delaying it: advancing a frame wastes bandwidth locally by scheduling it before it is due, whereas delaying it potentially increases startup delay for all clients expecting it. The impact of both operations is a reduction of the startup—bandwidth efficiency. The actual impact depends on the frame shifted, but in contrasting ways: *Delaying* later frames *increases* the average initial delay more strongly, since more clients wait for these frames. *Advancing* later frames, however, is less harmful as its marginal effect on average bandwidth usage decreases with increasing gap between successive transmissions of a frame.

With this in mind, we define two parameters δ_a and δ_d, which together provide the boundaries for advancing or delaying of frame f out of time slot t between $t-(w+f)\delta_a$ and $t+\delta_d \times w$. Reasonable defaults are $\delta_a \approx 0.05$ and $\delta_d < 0.1$, but these val-

[1] This fuzziness of operation is the origin of the term "Fuzzycast."

Algorithm 3 FIND-NEIGHBOR function implementing BFSCAN

```
1: δ_t ← λ;
2: for (i ← λ; i > λ − left; i − −) do
3:    if (B_act[t + i] ≤ B_est) then
4:       δ_t ← i;
5:       break;
6:    else if (B_act[t + i] < B_act[t + δ_t]) then
7:       δ_t ← i;
8: for (i ← λ + 1; i < λ + right; i++) do
9:    /* Lines 3 through 7 */
10:
11: return δ_t;
```

Algorithm 4 Co-scheduling multiple movies

```
1: B_est ← B_act ← 0;
2: for all movies m_i do
3:    b_frame ← b_block ← 0;
4:    right ← w_i δ_d;
5:    for all frames f_j ∈ m_i do
6:       λ ← j + w_i;
7:       left ← λ δ_a;
8:       B_est+ = 1/λ;
9:       for (t ← λ; t ≤ t_max; t+ = δ_t, B_act[t] + +) do
10:         δ_t ← FIND-NEIGHBOR;
11:         transmit(t + δ_t, f_j);
```

ues can be tuned during system setup or at the configuration stage to take into account practical limits on server bandwidth and delay variability. For example, variability in startup delay can be forbidden by setting $\delta_d = 0$ so that a frame may only be advanced from its original slot.

Given these limits, there are many ways to implement a neighborhood search function, such as:

BFSCAN: Starting from t, scan first backward from t to $t - \delta_a(f + w)$ and then forward from t to $t + \delta_d \times w$, looking for time slots with available bandwidth.

FBSCAN: Similar to BFSCAN, but start by going forward first.

SPIRAL: Search along a spiral path alternatingly going backward and forward, so that $t - \delta_a(w + f)$ is evaluated just before $t + \delta_d \times w$. To accommodate asymmetric bounds, the spiral is appropriately distorted. For example, if the advancing limit is 6 frames and delay limit 3, the sequence of time slots that SPIRAL considers is

$$t,\ t-1,\ t-2,\ t+1,\ t-3,\ t-4,\ t+2,\ t-5,\ t-6,\ t+3 \quad .$$

It is possible that FIND-NEIGHBOR finds no neighbor that can accommodate frame f. As a fallback, if all instants in the search interval exceed their allotted bandwidth, these algorithms schedule f in the minimum bandwidth instant within this interval. But our simulations suggest that this will seldom happen for reasonable values of δ_a and δ_d as both the allotted bandwidth and the search interval size increase with the frame number, thus continuously increasing the degree of freedom.

As shown in Fig. 2(a), these strategies can be represented by paths from coordinate (t, t) to $(t - \delta_a(w + f), t + \delta_d \times w)$. For these paths, both coordinates are non-decreasing as the path progresses. For example, SPIRAL can be represented by a straight line between the two points, as mapped by Bresenham's line drawing algorithm [15].[2] Advancing horizontally or vertically by a "pixel" results in probing the next unprobed time slot in backward or forward direction, respectively; direction changes on the rectangle correspond to direction changes in the search. Extensive simulation over a wide range of parameters indicates that SPIRAL is a robust way to perform a neighborhood search. Because of its back-and-forth nature, SPIRAL generates feasible schedules while managing to place frames close to their original time slots.

[2]Our use of Bresenham's line drawing algorithm to distort the SPIRAL helped introduce the path visualization.

Algorithm 3 shows the implementation of the FINDNEIGHBOR function. For clarity, we have used the simpler BFSCAN algorithm instead of SPIRAL. Fig. 2(b) displays the *bandwidth spectrum*, i.e. the distribution of bandwidths over time, for transmitting a 30-frames-per-second (fps) 2-h movie with various initial delays.

IV. Specialized Scheduling

A. Peak versus Average Bandwidth

Our assumption of taking frames as indivisible units shows a drawback: if the theoretical server bandwidth requirement is, say, exactly 4.1 frames/instant, even the best possible algorithm will necessarily have to schedule (at least) 5 frames in some ($\approx 10\%$) time slots, so that peak bandwidth usage overshoots the average by more than 20%. The obvious remedy would be to divide frames into many smaller units, which would significantly increase server scheduling and client reordering complexities, as well as disk access times due to increased seeks.

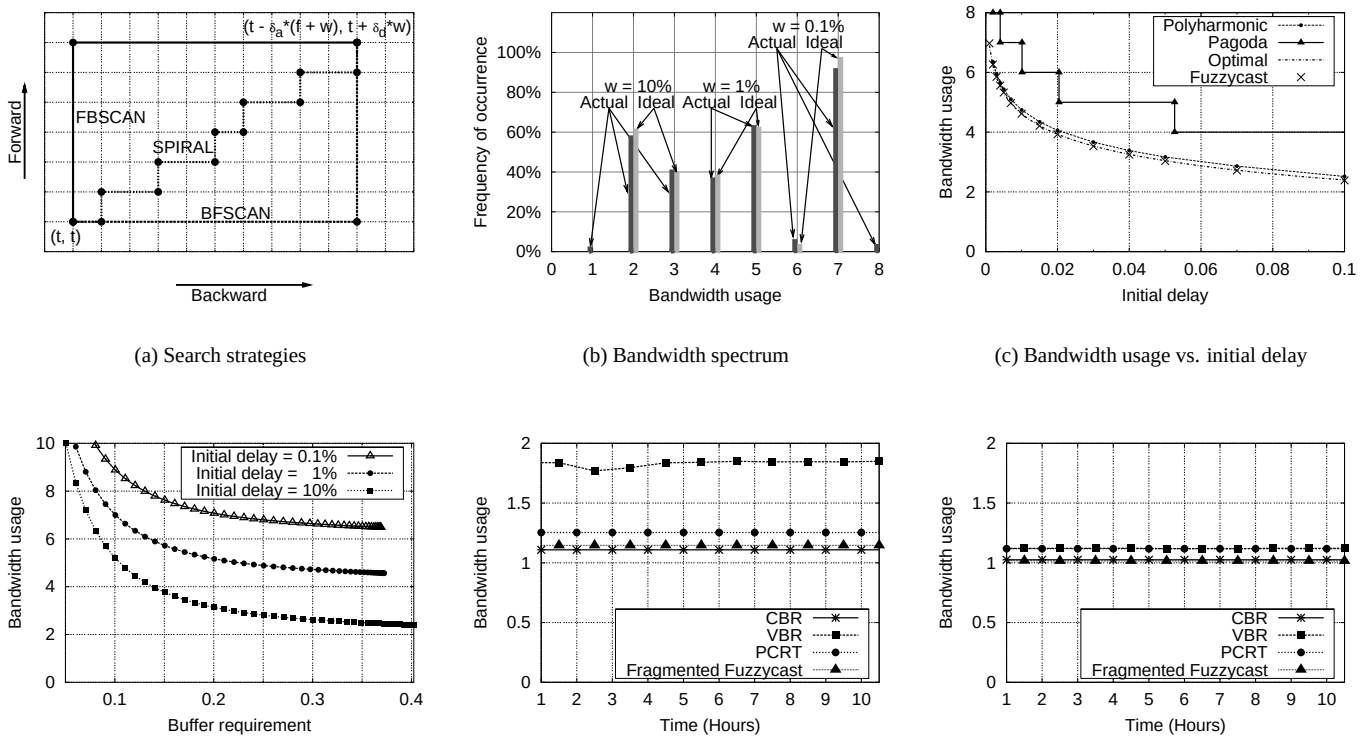
Instead, we observe that it is likely for any VoD system to broadcast multiple movies simultaneously. By modifying algorithm BASIC to be aware of both allotted and consumed global bandwidth when making scheduling decisions (Algorithm IV-A), it is possible to efficiently co-schedule multiple streams. We find the resulting co-scheduling to as few as 8 concurrent streams results in a peak bandwidth usage within 2% of the optimum (Fig. 2(f)).

B. Support for Variable-Bit-Rate Media

So far, we have made the simplification that the media are encoded at a constant bit rate (CBR). In practice, however, popular media encoding results in variable frame sizes and thus VBR. Algorithm BASIC can be used to schedule VBR frames, provided the frame sizes are incorporated into the bandwidth calculation. For an n-frame movie with frame sizes $f_1, f_2, \cdots, f_n$, estimated bandwidth for the first p frames is:

$$B_{\text{VBR}}(p) = \sum_{i=1}^{p} \frac{f_i}{w + i} \quad .$$

When combined with the global scheduling algorithm, this significantly smoothens bandwidth usage. For example, Fig. 2(f) shows the bandwidth usage (normalized bandwidth predicted according to Eq. (1)) of 1-h MPEG-4 movie streams, over a 10-h



(a) Search strategies

(b) Bandwidth spectrum

(c) Bandwidth usage vs. initial delay

(d) Bandwidth usage vs. buffer space requirement

(e) Peak Bandwidth usage: single VBR stream

(f) Peak Bandwidth usage: 8 VBR streams

Fig. 2. These graphs illustrate the techniques and performance of our heuristic scheduling schemes. Fig. 2(a) formulates the various neighborhood search strategies as paths along a rectangular grid. Fig. 2(b) shows the distribution of bandwidth usage over time, in comparison with the optimum distributions. Fig. 2(c) plots average bandwidth usage versus initial delay for various proactive video-on-demand schemes. Fig. 2(d) measures the average bandwidth usage of the Limited-buffer scheme against the buffer space requirement it entails. Figures 2(e) and 2(f) plot the bandwidth variability of various schemes while transmitting variable bit-rate content. In these last two graphs, the peak bandwidth usage is measured on an hourly basis and is normalized to the average for that period.

Algorithm 5 FRAGMENT function

```
1: b_block ← b_frame ← 0;
2: i ← 0;
3: for all frames f_j do
4:    while (b_block ≥ b_frame and j < n) do
5:       b_frame += size(f_j);
6:       j++;
7:    λ_i ← w + j;
8:    b_block += blocksize;
9:    i++;
10: return (λ_1, λ_2, ···);
```

Algorithm 6 Fragmented Fuzzycast (co-scheduled)

```
1: B_est ← B_act ← 0;
2: for all movies m_i do
3:    (λ_1, λ_2, ···) ← FRAGMENT;
4:    right ← w_i δ_d;
5:    for all blocks b_j ∈ m_i do
6:       B_est += 1/λ_j;
7:       left ← λ_j δ_a;
8:       for (t ← λ_j; t ≤ t_max; t+ = δ_t, B_act[t] + +) do
9:          δ_t ← FIND-NEIGHBOR;
10:         transmit(t + δ_t, f_j);
```

period. However, clients do not benefit from the smoothing effect of multiple streams; they still suffer from significant bandwidth variability. Variable-sized frames also complicate client buffer management.

Using a smoothing mechanism such as piecewise constant-rate transmission (PCRT) [16, 17] is an effective compromise. PCRT smoothens by dividing the media into a few variable-sized segments, which are then transmitted at constant rates. Initial delay and peak bandwidth usage strongly depend on how the movie is split up [16]. PCRT smoothens bandwidth variability effectively (Fig. 2(e)) but the additional initial delay incurred sometimes results in performance overheads that exceed 20%.

We now propose a simpler and more effective solution called *Fragmented Fuzzycast*, which is a straightforward extension of our original frame scheduling: Consider a VBR-encoded movie with a set of frames $F = \{f_1, f_2, \cdots, f_n\}$, split into a set of fixed-sized blocks $B = \{b_1, b_2, \cdots, b_m\}$. For each block b_i, there is a set $C(b_i) \subset F$ of frames that are either fully or partially contained in b_i. If the earliest frame in $C(b_i)$ is f_j, then transmit block b_i at frequency $1/(w + j)$.

Theorem 4 Fragmented Fuzzycast delivers all data on time.

Proof: Block b_i is scheduled such that the earliest frame in it reaches all clients on time. By fixing its transmission rate according to the frame with the most urgent requirement, we ensure that later frames in it also reach their destination on time. If the last frame in block b_i is truncated, transmitting block b_i only

Algorithm 7 LIMITED-BUFFER

```
1: for all frames f_j do
2:     λ ← min(j + w, v + w);
3:     for (t ← λ; t ≤ t_max; t+ = λ) do
4:         transmit (t, f_j);
```

guarantees on-time delivery of this fragment. But the rest of this frame, by virtue of being the earliest in block b_{i+1}, is delivered on time. As all the frames in the last block are delivered on time, so are all the frames in a movie. ∎

Fragmented Fuzzycast (Algorithm 5) is simple to implement: we maintain pointers to the end of the current block in b_{block} and the current frame in b_{frame}, which grow at rates $blocksize$ and size(j) respectively. Whenever the block pointer overtakes the frame pointer, the frame number is increased until this state is reversed. Fig. 2(f) shows that Fragmented Fuzzycast is effective in smoothing the rate variability of VBR traffic: the graph is a virtual replica of the CBR bandwidth usage in the same graph. Algorithm 6 shows a version of the Fuzzycast algorithm supporting co-scheduling multiple VBR streams.

C. Support for Limited Client Buffers

From Theorem 3, we know that the peak buffer requirements run to about 37% of the movie. This is not an issue as far as desktop clients are concerned: the most inexpensive hard disks today routinely come at tens of gigabytes. However, this could be too high a requirement for "dumb" clients that, in the absence of a hard disk, must buffer entirely in RAM or other, non-disk, media. In case a significant fraction of clients using these devices, there is a need to tailor transmission according to the required client buffer limit. As we have seen, possibly at the cost of increased network bandwidth usage.

One simple way to achieve this is to transmit frames more frequently than necessary, so that clients do not have to buffer later frames for too long before playout. Specifically, we define a limit v up to which the interframe distance increases (cf. Algorithm 7).

Using simple probabilistic analysis, we can show that the buffer space requirement over time follows

$$D(t) = \begin{cases} t\log(\frac{v}{t}) + \frac{t^2}{2v} & 0 < t < v, \\ \frac{v}{2} & v < t < n - v, \\ \frac{t}{2} - \frac{(t-n+v)^2}{2v} & n - v < t < n. \end{cases}$$

The increased bandwidth is given by

$$B = \log(\frac{v+w}{n+w}) + \frac{n-v}{v}. \qquad (2)$$

The key parameter here is v, which can be tuned according to the following rule: peak buffer space $= v/2$, with a commensurate increase in network bandwidth usage given by Eq. (2). Figure 2(d) shows the relation between bandwidth and buffer space requirements for various initial delays.

V. TRANSMITTING OVER MULTIPLE MULTICAST GROUPS

A. Problem Statement

Periodic broadcast schemes are attractive in terms of server bandwidth usage but tend to consume additional client and network bandwidth by continuously and redundantly transmitting data. While this is unavoidable in a purely broadcast-based system, e.g., a satellite- or cable-based distribution network, it is wasteful in a multicast situation where network support for subscribing to and unsubscribing from a multicast session is available. It is therefore desirable that each client explicitly deregister its interest in unwanted frames with the multicast infrastructure.

The transmission goal is to strictly avoid sending the same frame more than once to the same client. In a purely proactive system, this is possible only if a client could unsubscribe selectively from further transmissions of an arbitrary frame upon receiving it once. Although such high granularity of choice may be achieved in theory by dedicating one multicast group per frame, this solution is clearly infeasible due to the high network overheads it incurs in the form of group membership messages and multicast state information in routers.[3] For practical reasons, each movie should therefore be multicast over a small number of groups. Each client initially subscribes to all the groups of a movie and then proceeds to discard each group upon receiving every frame in it at least once. This is similar to, but different from existing techniques for receiver-driven congestion control [18] and efficient data distribution over layered multicast [19, 20].

Our problem, then, is simply stated: Given a movie of n frames, how to transmit it over α multicast groups in a way that minimizes total redundancy? As the first few frames use the most bandwidth in our scheduling scheme, it is intuitive to drop early and drop often. But there is a strict limit on the number of groups: a greedy assignment will merely exchange the bombardment of a few frequent frames for the slow torture of many infrequent ones.

In the remainder of this section, we show how this problem is actually a specific instance of a general optimization trade-off that we have encountered frequently enough to assign it a name – "Scottie's dilemma."

B. Scottie's Dilemma: When to Cut Costs?

In situations involving processes that have a constantly accruing cost but decreasing utility, we would like to cut costs as soon and as often as possible, rather than drag along excess baggage. However, practical constraints dictate that we aggregate such actions into a few distinct decision points rather than continuously improve the state of affairs. This dilemma is common in real life: psychologists speak of deferring instant gratification for long-term profit; rocket scientists have to decide when and how often their creations jettison used-up booster stages; file systems periodically synchronize with storage and discard modified buffers.

In general, situations of this kind can be represented by two simple functions: $\Theta(t)$, a weight function that defines how cost accrues over time, and $\Phi(t)$, a utility function that defines how utility decays with time. In the common case when costs add up linearly in time, $\Theta(t) = t$.

[3] Better solutions are possible if active networking technology were deployed in the network.

Given these two functions, the theoretical minimum cost, C_∞, is obtained by perfectly following the utility at each instant:

$$C_\infty = \int_0^T \Phi(t)\, d\Theta(t).$$

However, in practice, it is more realistic to assume that time consists of a number of distinct *epochs* (say, again, α of them), separated by decision points $t_0 = 0, t_1, \cdots, t_\alpha = T$. At each decision point, unwanted costs accumulated during the preceding epoch are eliminated. In this case the total cost is given by

$$\begin{aligned} C_\alpha &= \sum_{k=1}^{\alpha} \int_{t_{k-1}}^{t_k} \Phi(t)\, d\Theta(t) \\ &= \sum_{k=1}^{\alpha} \big\{\Theta(t_k) - \Theta(t_{k-1})\big\}\Phi(t_{k-1}), \end{aligned} \tag{3}$$

where $t_0 = 0$ and $t_\alpha = T$.

Thus the tradeoff is reduced to choosing an optimum set of decision points $(t_1^*, t_2^*, \cdots, t_{\alpha-1}^*)$ that minimizes cost $C_\alpha = C_\alpha^*$. Differentiating both sides of Eq. (3) w.r.t. t_k, we obtain

$$\begin{aligned} \tfrac{\partial C_\alpha}{\partial t_k} &= \big(\Theta(t_{k+1}) - \Theta(t_k)\big)\Phi'(t_k) - \\ &\quad \Theta'(t_k)\Phi(t_k) + \Theta'(t_k)\Phi(t_{k-1}) \\ &= 0 \qquad \text{(for minimum cost)}, \end{aligned}$$

or

$$\Theta(t_{k+1}^*) = \Theta(t_k^*) + \frac{\Theta'(t_k^*)}{\Phi'(t_k^*)}\big(\Phi(t_k^*) - \Phi(t_{k-1}^*)\big) \quad . \tag{4}$$

This recurrence can then be solved for specific cost and utility functions $\Theta(t)$ and $\Phi(t)$, to obtain optimum decision points. To assess performance, we define inefficiency as follows:

$$I(\alpha) = \frac{\text{optimum cost with } \alpha \text{ groups}}{\text{theoretical minimum cost}} = \frac{C_\alpha^*}{C_\infty} \quad . \tag{5}$$

C. Numerical Solutions

When closed-form expressions for the optimum boundaries cannot be obtained, numerical methods can be applied. Owing to the recursive nature of Eq. (4), finding the set of optimum decision points $(t_1^*, t_2^*, ...t_{\alpha-1}^*)$ reduces to finding the first point t_1^*. For a given candidate $t_1 = x$, we can define a recursive set of functions, $t_2(x), t_3(x), \cdots t_\alpha(x)$ that can be determined either analytically or numerically using Eq. (4). As it is always true that $t_\alpha(t_1^*) = t_\alpha^* = T$, finding t_1^* reduces to solving

$$t_\alpha(x) - T = 0 \quad .$$

This can be done numerically, e.g., using the Newton–Raphson iteration. Once t_1^* has been determined, all optimum decision points can readily be decided. In the cases of interest to us, time is measured using integers, and $t_k(x)$ are monotonically increasing functions of x, so that we may do a binary search on the time interval, resulting in an $O(\alpha \log T)$ algorithm to find α optimum boundaries over time T. When $\Theta(t)$ or $\Phi(t)$ is an arbitrary function defined over integer t, we can use a dynamic programming approach shown in Algorithm 8 to obtain an optimum solution in $O(\alpha T^2)$ time.

Algorithm 8 Dynamic program solving Scottie's dilemma

```
G_0 ← 0;
for i ← t_0 to t_α do
    G_i ← G_{i−1} + Φ(i);
    F_{1,i} ← G_i ∗ Θ(i);
for i ← 2 to α do
    for j ← t_0 to t_α do
        F_{i,j}, k_{i,j} ← F_{i−1,j}, j;
        for k ← t_0 to j do
            if F_{i−1,k} + (G_j − G_k) ∗ Θ(j) < F_{i,j} then
                F_{i,j}, k_{i,j} ← F_{i−1,k} + (G_j − G_k) ∗ Θ(j), k;
t*_α ← t_α;
for i ← α − 1 to 1 do
    t*_i ← k_{i,t*_{i+1}};
return t*_1, t*_2, ··· t*_{α−1};
```

VI. Fuzzycast Over Multiple Groups as an Instance of Scottie's Dilemma

In this section, we consider the specific problem of how to optimally partition a Fuzzycast transmission over multiple multicast groups. Given the discussion in the previous section, it is apparent that partitioning a transmission over multiple multicast groups is an instance of "Scottie's dilemma," where "epochs" correspond to distinct multicast groups. Given α decision points in which played-out frames can be dropped, we have to choose the points that minimize redundancy given various objectives. Let us consider two different objectives: minimizing client load and minimizing overall network load. (Recall that there is no need to minimize server load, as it is constant, independent of the number of multicast groups and the dropping points.)

A. Case 1: Minimizing Client Load

We first consider the partition that minimizes the total number of frames that each client receives. In this case, total cost is given by the number of frames received during the course of transmission. The utility of the transmission at any time is the portion of the frames received for the first time.[4] Thus, the weight and utility functions may be formulated as

$$\Theta(t) = t; \quad \Phi(t) = \int_t^T \frac{1}{t}\, dt = \log \frac{T}{t} \quad .$$

In this case, optimum values of drop boundaries are given by

$$t_{k+1}^* = t_k^* \left(1 + \log \frac{t_k^*}{t_{k-1}^*}\right) \quad . \tag{6}$$

Descending recursively, the first optimum drop point t_1^* is determined by

$$t_\alpha^* = t_1^* \underbrace{\left(1 + \log \frac{t_1^*}{t_0^*}\right)\left(1 + \log\left(1 + \log \frac{t_1^*}{t_0^*}\right)\right)\cdots}_{\alpha \text{ terms}} \quad , \tag{7}$$

where $t_\alpha^* = n + w$ and $t_0^* = w$. For convenience, we assumed time starts with w.

[4] Ignoring the negligible effect of scheduling jitter and the case of limited client buffers, each frame is received exactly once before playout. Therefore, this also matches the number of frames that have not yet been played out.

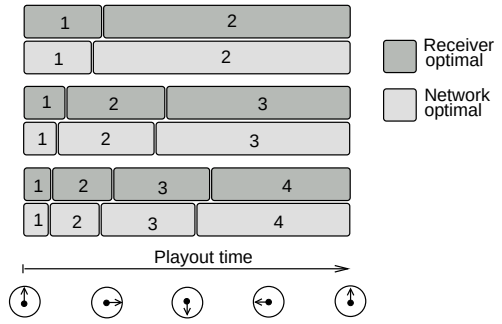


Fig. 3. Optimum partition (1-h movie, 36 s initial delay)

Using the method outlined in Section V-C, Eq. (7) can be solved numerically to obtain $t_1^*, \cdots, t_{\alpha-1}^*$. This set of boundaries is the one that minimizes the number of frames that each client receives. For example for a 1-h, 30-fps movie with $\alpha = 3$ and $w = 36$ s (1%), the optimum group boundaries are at 7:34, 26:46, and 60:36 min, leading to an average client bandwidth usage of 54 fps, as opposed to 165 fps without layering, roughly 67% reduction.

To measure the performance gain in this case, we use Eq. (5) to obtain the *Receiver inefficiency*:

$$C_\alpha^* = \sum_{k=1}^{\alpha} t_k^* \log \frac{t_k^*}{t_{k-1}^*}; \quad C_\infty = \int_w^{n+w} \log \frac{n+w}{t}\, dt$$
$$I_R(\alpha) = \frac{\text{\# frames received on average}}{\text{\# frames in movie}}$$
$$\approx \frac{1}{n} \sum_{k=1}^{\alpha} t_k^* \log \frac{t_k^*}{t_{k-1}^*}.$$

Fig. 4(a) plots the receiver inefficiency against α for various initial delays. Fig. 4(b) shows the values of inefficiency obtained through simulation. There shows excellent agreement between the predicted and the experimental values. We also find that there is a "sweet spot" at around 4–5 groups, where maximum gains are obtained; increasing α further does not result in significant performance gains.

B. Case 2: Minimizing Network Load

Another problem that might be more relevant from an ISP's viewpoint is to find the partition that minimizes overall network costs, i.e., we would like to minimize the number of frames in the network at any given time.

If the number of links in a delivery tree of m clients is $L(m)$ and the average client arrival rate is λ, then the number of clients subscribed to group k at any given time is $\approx \lambda t_k$. Throughout this section, we assume that clients are characterized by unique end routers. According to this definition, multiple end users on a single local network count as a single client.

A seminal result obtained by Chuang and Sirbu [21] states that for Internet multicast, $L(m)$ is fairly accurately approximated by a power law of the form, $L(m) \approx \hat{u} m^\rho$, where $\rho \approx 0.8$ and $\hat{u}$ is the average unicast path length (recall that m represents the number of unique end routers). m/m^ρ thus represents its network bandwidth advantage over multiple unicast, which has $L(m) = \hat{u}m$. This was subsequently verified by Phillips et al. [22].

Now, we can simply set up the weight function as the number of links in a group at time t:

$$\Theta(t) = \hat{u}(\lambda t)^\rho; \quad \Phi(t) = \log \frac{T}{t} \quad .$$

This results in the recurrence

$$t_{k+1}^* = t_k^* \left(1 + \rho \log \frac{t_k^*}{t_{k-1}^*}\right)^{\frac{1}{\rho}} \quad \text{i.e.,}$$
$$t_\alpha^* = t_1^* \underbrace{\left(1 + \rho \log \frac{t_1^*}{t_0^*}\right)^{\frac{1}{\rho}} \left(1 + \log\left(1 + \rho \log \frac{t_1^*}{t_0^*}\right)\right)^{\frac{1}{\rho}} \cdots}_{\alpha \text{ terms}} \quad . \tag{8}$$

Again, this equation can be numerically solved to get optimum $t_1 = t_1^*$.

To measure performance, we obtain the *Network inefficiency* from Eq. (5) as follows:

$$C_\alpha^* = \sum_{k=1}^{\alpha} (t_k^*)^\rho \log \frac{t_k^*}{t_{k-1}^*}; \quad C_\infty = \int_w^{n+w} \log \frac{n+w}{t}\ d(t^\rho)$$
$$I_N(\alpha) = \frac{\text{\# frames in network at any time}}{\text{minimum \# frames in network}}$$
$$\approx \frac{\rho}{(n+w)^\rho} \sum_{k=1}^{\alpha} (t_k^*)^\rho \log \frac{t_k^*}{t_{k-1}^*} \quad . \tag{9}$$

Fig. 4(d) shows the network inefficiency versus α for various w. Figures 4(e) and 4(f) shows the values obtained from simulation over realistic network topologies created using the GT-ITM [23] simulator and from traces obtained from the SCAN [24] project. Details about our simulation setup are given in Section VI-E. As the figure shows, there is excellent agreement between predicted and observed values, both for generated and real topologies. Again, there is a "sweet spot" at around 4–5 groups, beyond which increasing α does not seem to have much effect.

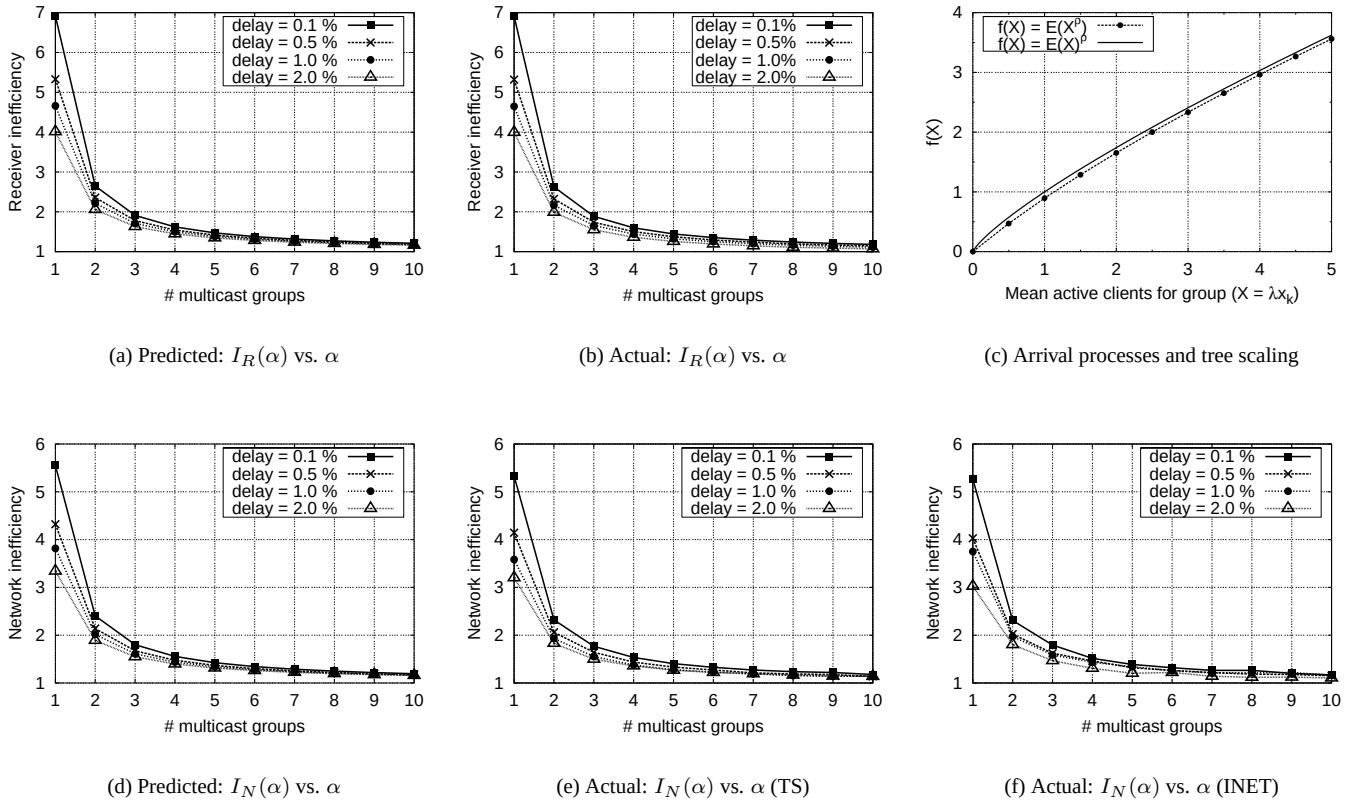
C. Comparing Receiver-Optimum and Network-Optimum Cases

In Fig. 3, we compare the partitions in the receiver-optimum case and the network-optimum case. It is apparent from the figure that the boundaries for the network-optimum case are earlier than the corresponding receiver-optimum boundaries. This is in fact always true and can easily be proved by letting $z_k = t_k/t_{k-1}$ in both cases, so that Eq. (6) and Eq. (8) both reduce to the form

$$z_{k+1} = (1 + \rho \log z_k)^{\frac{1}{\rho}} \quad ,$$

where $\rho = 1.0$ in the first case and 0.8 in the second (when using multicast on Internet topologies). This can be shown to be an decreasing function of ρ, from which the result immediately follows.

The intuition behind this result is that the sublinear dependence of the multicast tree size on the membership size "dilutes" the effect of large groups, so that when optimizing for network load, it is advantageous to drop the initial high-bandwidth frames sooner.



(a) Predicted: $I_R(\alpha)$ vs. α

(b) Actual: $I_R(\alpha)$ vs. α

(c) Arrival processes and tree scaling

(d) Predicted: $I_N(\alpha)$ vs. α

(e) Actual: $I_N(\alpha)$ vs. α (TS)

(f) Actual: $I_N(\alpha)$ vs. α (INET)

Fig. 4. This set of graphs plots the performance of the schemes for efficiently transmitting content over multiple multicast groups. Figures 4(a) and 4(b) plot the *receiver inefficiency* (= #frames received/#frames in movie) versus α, the number of groups used. The first shows predicted performance, the second performance measured in simulations. Fig. 4(c) relates to the approximation discussed in Section VI-D and provides a numerical justification for it. The last series of graphs Figures 4(d)–4(f) are from Section VI-B and measure the *network inefficiency* (= #frames in network/minimum #frames in network) as a function of the number of groups. The first shows the performance in theory; the next two plot performance measured in simulations involving two different topologies explained in Section VI-E.

D. Variable Arrival Rate

In Sections VI and especially VI-B, we have assumed that client arrivals are uniformly distributed. Specifically, we assumed that with an arrival rate λ, the number of clients in time t would be $\lambda \times t$. However, realistic client arrivals follow a distribution centered around a mean λ. For Poisson-distributed arrivals, the network cost is given by

$$N \approx K' \times \sum_{k=1}^{\alpha} E(X^{\rho}, \lambda x_k) \log \frac{x_k}{x_{k-1}} \quad ,$$

where $E(f(X), \lambda)$ is $\sum_{k=0}^{\infty} \frac{e^{-\lambda} \lambda^k f(k)}{k!}$. That is, in the preceding section we have implicitly assumed $E(X^{\rho}, \lambda x_k) \equiv E(X, \lambda x_k)^{\rho}$, which is not true in general. However, as the mean arrival rate for any given group exceeds 1, these two expressions converge rapidly (Fig. 4(c)). Because x_k is of the order of a few minutes and the content is popular, the approximation is justified, at least for the Poisson case.

E. Performance Analysis

We now study the performance of these techniques in realistic situations. Our simulation setup is as follows: For the topology generated using GT-ITM [23], we created a transit-stub graph containing $\approx 10,000$ nodes and $36,000$ edges. For the real network topology, we have used the merged traces of the SCAN project [24] and the Internet mapping project [25] at Bell Labs. To make this huge topology manageable, we have chosen to construct a subgraph by doing a traversal with maximum depth 8 starting from an arbitrary node.[5]

Having generated a graph from this data, we pick a random source S (In the GT-ITM model, this is a stub node). We pick unique receivers R_i located at random nodes n_i, select start times s_i with an average arrival rate of λ, and construct the distribution tree. At random times t_j, we use the rule that for each multicast group k, R_i is subscribed at time t_j if and only if $t_j \geq s_i$ and $t_j \leq s_i + x_k$ in order to calculate L_{kj}, the distinct links involved in group k at time t_j. The results are then averaged to obtain an estimate $\hat{L}_k$ of $L_k = L(\lambda x_k)$. The overall network bandwidth can then be estimated as $\sum_{k=1}^{\alpha} \hat{L}_k \log \frac{x_k}{x_{k-1}}$.

In Figures 4(e) and 4(f), we plot the performance predicted by Eq. (9) compared with values obtained by graph simulations. *TS* refers to the GT-ITM generated transit-stub graph, and *INET* refers to the Internet trace. As the figure shows, there is good agreement between estimated and empirical values.

VII. Implementation

Our system consists of an application-level, proactive Media-on-Demand server and multiple Fuzzycast clients, all of which

[5] These traces, along with programs for their manipulation, can be found at http://www.arl.wustl.edu/∼rama/traces/ .

are connected to a multicast-enabled 100-Mbps Ethernet. Multiple media streams can be served, but are limited by the network bandwidth and disk throughput available to the server. The current implementation is a proof-of-concept prototype written in about 1800 lines of *Java* code and organized into object-oriented modules to facilitate flexible plug-ins of different algorithms, such as server scheduling and client caching. Its architecture, shown in Fig. 5, consists of the following components:

SCHEDULER: The Scheduler incorporates most of the functionality described in the above sections. Given a list of metafiles describing multiple media files, it can set up concurrent playout schedules for these media items, relying on Algorithm 6. The Server Cache and the Dispatcher are regularly notified, triggering a chain of events that ultimately results in frames being transmitted as scheduled.

DISPATCHER: The Dispatcher is a multithreaded process that accepts frame data from various media streams and muticasts them according to a schedule over multiple multicast groups. It segregates transmissions of various movies into flows, each flow operating at its optimum bandwidth (plus about 2 to 3%) as determined by Eq. (1).

SERVER CACHE: The Server Cache is a circular memory buffer that caches frequently transmitted frames of each movie. The cache acts on requests for frames from the scheduler. It maintains a mapping between media frames and disk blocks. The buffer is shared with the Dispatcher.

PROGRAM GUIDE: The Program Guide Server keeps a profile for each media stream, which contains network information (multicast address and port), protocol information (stream length, packet size, initial delay, number of multicast groups), and media information (stream name, brief introduction, snapshot, media type). Each receiver connects to the server through unicast and downloads a program guide. An alternative implementation could multicast the program guide in its own well-known, low-bandwidth channel.

RECEIVER: The Receiver is responsible for subscribing to and receiving media data. The receiver 'tunes in' to the appropriate multicast channels according to the information provided by the directory service.

CLIENT CACHE: The Client Cache is filled by the receiver and is consumed by the displayer. In our scheme, the client is required to have sufficient buffer space for about 37% of the movie. For efficiency, the Client Cache buffers frames that are near playout in main memory. Efficient schemes for client side buffer management are described in [26].

DISPLAYER: The Displayer paces the data to the rate the media player desires and provides the media stream through standard HTTP streaming, ready for use by a local off-the-shelf media player. This increase the flexibility of integration with external media players for supporting various newer or proprietary media types not handled by our internal player. For example, we have successfully used RealPlayer to play RealVideo movies, without need for knowledge of the actual format.

CLIENT GUI: The client GUI interacts with the end user. As soon as the client connects to the system, it downloads a program guide from the directory service via a unicast channel, and shows the brief introductions and snapshots for each media stream. Whenever a media stream is selected, it will wait an initial delay according to the stream's profile, and starts the integrated player to play sequentially until the end of the stream or user interruption. We use Java Media Framework [27] to construct the integrated player, due to its platform independence.

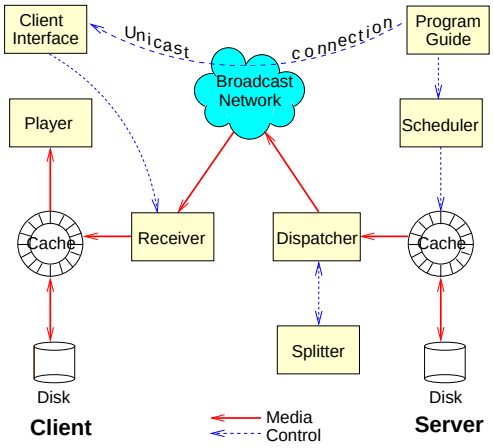


Fig. 5. System architecture

Fig. 6 shows the screen snapshots from a Linux-based server and two clients on Linux and Windows respectively. For comparison, we show the server side network bandwidth at different time points. The observed bandwidth remains approximately constant. The two clients join the system at different time and get the same media stream at the specified initial delay.

The plot results from the client windows also show that the theoretical results provided in the above sections are met. For example, the client network bandwidth and the client cache growth curves are in accordance with the results shown in Fig. 2(c) and Fig. 2(d), respectively.

VIII. RELATED WORK

Among the earliest proposals for bandwidth-efficient VoD was *Batching* [3], where the server aggregated requests that came close together in time. In subsequent years, progressively more efficient periodic broadcast methods have been proposed.

PROACTIVE TRANSMISSION SCHEMES: Recently, the *Harmonic Broadcasting* [6,7] family of protocols (discussed in Sections II-B and II-C) seem to be the most promising insofar as the bandwidth-delay tradeoff is concerned. Some lower bounds for the performance of such protocols were obtained in [8,9,28]. The impact of packet loss was evaluated and reduced in [29]. Support for interactive functions was introduced in [30].

PRE-PUSH: Several commercial pay-per-view networks are currently testing "on-demand" models, in which movies are downloaded ahead of time to consumer set-top boxes. With this technique, a single broadcast transmission suffices to preload all data. The downside is that enormous storage amounts are required to keep enough data so that an acceptable selection of movies can be offered. Moreover, while most demand at any given time is for a small set of movies, the composition of this set is a moving target, defeating attempts at any long-term client-side caching.

SMOOTHING VBR VIDEO: Although there is a large body of work on smoothing unicast transmission of VBR video [16, 17], the impact of VBR media on the performance of proactive multicast schemes has never been properly studied.

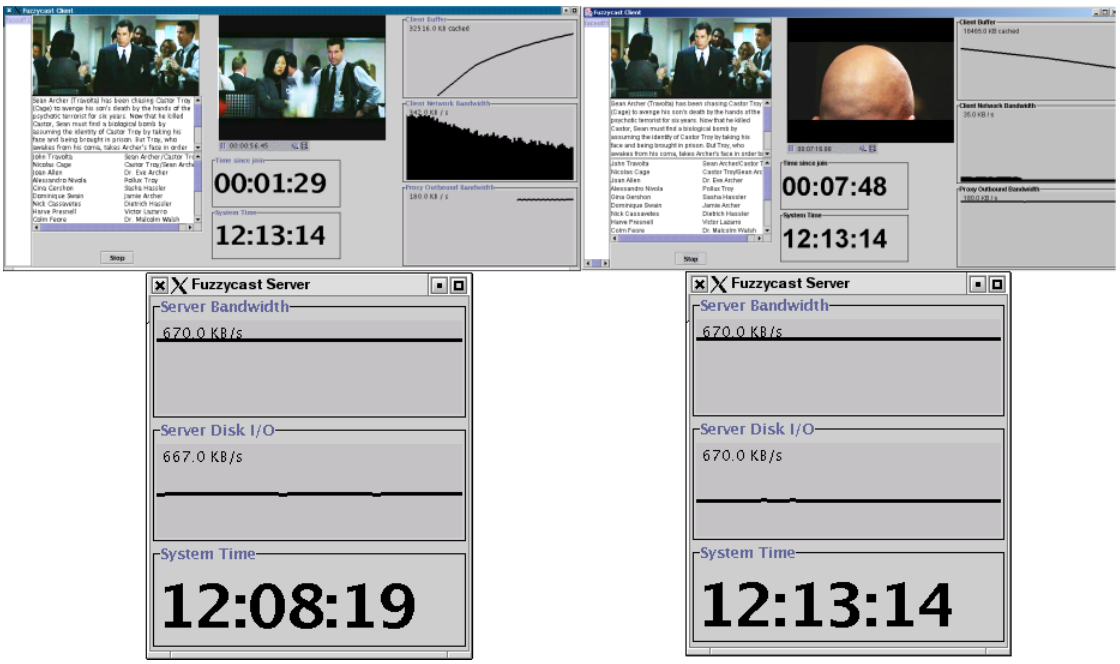


Fig. 6. Screen captures of clients and server in action. Top left and right show snapshots from two clients, running under Linux and Windows, respectively, that joined at different times. The bottom two shots show the server load before and after the second client joined. Client windows: Left is the movie guide; in the center the current movie as well as movie and system time; the graphs on the right show buffer size, bandwidth from the network, and bandwidth to the player.

LAYERING OVER MULTIPLE GROUPS: We introduced the problem of client and network optimum partitioning over multicast groups for Fuzzycast in [1]. To our knowledge, no prior work to quantifying the network impact of proactive VoD protocols or on optimally distributing content among multiple multicast groups exists. Bhattacharyya et al. [20] discuss optimum scheduling of data packets in a layered multicast transmission [18] to receivers with identical starting times.

CONTENT DISTRIBUTION NETWORKS: Content distribution networks (CDNs [31]) are an alternative way of providing VoD to many clients. For the most part, they are orthogonal to the work on harmonic broadcasting. For our purpose, CDNs just provide a way to trade investments in networks and routers for servers and storage. Combining bandwidth-efficient distribution strategies with cache hierarchies in a cost-effective manner currently is an area of active research.

BULK DATA DISTRIBUTION: Byers et al. proposed a digital fountain approach to data distribution [19], in which receivers download from a continuous data stream until they have received enough unique encoded data to reconstruct all of the original data. While this is an attractive solution for bulk data transfer in which data only needs to be reconstructed once at the end of transmission, it does not seem to be readily applicable to streaming media applications, which requires the first parts of the stream to be reconstructed early. An adaptation of digital fountains to VoD using UEP codes is described in [13]. Compared to Fuzzycast, the use of coding requires an additional processing step involving large amounts of main memory.

IX. CONCLUSIONS

The success of VoD systems depends on the provider's ability to offer a cost-effective service that is also attractive to end-users. Scalability and efficiency are critical for the former part, while functionality, ease of use, and quick response to user commands are needed to satisfy the latter aspect.

Proactive VoD protocols are attractive from the scalability point of view, because they use server bandwidth efficiently to serve media even under heavy demand. However, current proactive schemes have significant drawbacks in terms of practical implementation and deployment. Fuzzycast, by taking a pragmatic frame-oriented approach, uses near-optimum server bandwidth while remaining relatively simple to implement and maintain.

Although transmitting variable bitrate (VBR) media is a significant issue in the real world, most existing periodic multicast schemes do not handle VBR media very well. We proposed a simple extension to Fuzzycast, namely *Fragmented Fuzzycast*, and demonstrated that it was able to deliver VBR content over constant-rate channels with minimal performance loss or complexity overhead.

Finally, periodic multicast schemes place extra load on the network owing to redundant multicasts. We show how the problem of transmitting content over multiple multicast groups results in a fundamental resource tradeoff; by solving the general case, we obtain an optimum solution to our problem. We find that using even a few multicast groups results in significant reduction in overhead for both client and network. We note that the result obtained here is quite general and applicable to diverse situations, including networks that follow scaling properties that are very different from the Chuang–Sirbu law, in a straightforward manner.

Most importantly, we have shown that a simple and inexpensive heuristic scheduling can be used to outperform all known practical schemes, including many more complex schemes.

Using the optimum solution described here in other similar scenarios is an area of research we intend to pursue further. In addition, our current work involves extending these results to add support for more "user-friendly" options like interactive VCR-like functions.

REFERENCES

[1] Marcel Waldvogel and Ramaprabhu Janakiraman, "Efficient media-on-demand over multiple multicast groups," in *Proceedings of Globecom 2001*, San Antonio, Texas, USA, Nov. 2001.

[2] Ramaprabhu Janakiraman, Marcel Waldvogel, and Lihao Xu, "Fuzzycast: Efficient video-on-demand over multicast," in *Proceedings of INFOCOM*, New York, NY, USA, June 2002, pp. 920–929.

[3] Asit Dan, Dinkar Sitaram, and Perwez Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings ACM Multimedia '94*, Oct. 1994, pp. 391–398.

[4] Sridhar Ramesh, Injong Rhee, and Katherine Guo, "Multicast with cache (mcache): An adaptive zero-delay video-on-demand service," in *Proceedings of IEEE INFOCOM*, 2001, pp. 85–94.

[5] Kien A. Hua and Simon Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM SIGCOMM*, Sept. 1997, pp. 89–100.

[6] Li-Shen Juhn and Li-Meng Tseng, "Harmonic broadcasting for video-on-demand service," *IEEE Transactions on Broadcasting*, vol. 43, no. 3, pp. 268–271, Sept. 1997.

[7] Jehan-François Pâris, Steven W. Carter, and Darrel D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proceedings 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, July 1998, pp. 127–132.

[8] Subhabrata Sen, Lixin Gao, and Donald F. Towsley, "Frame-based periodic broadcast and fundamental resource tradeoffs," Tech. Rep. 99-78, University of Massachusetts, Amherst, 1999.

[9] Derek L. Eager, Mary K. Vernon, and John Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," in *Proceedings of Multimedia Information Systems Conference (MIS '99)*, Oct. 1999.

[10] Jehan-François Pâris, Steven W. Carter, and Darrel D. E. Long, "A low bandwidth broadcasting protocol for video on demand," in *Proceedings 7th International Conference on Computer Communications and Networks (IC3N'98)*, Oct. 1998, pp. 690–697.

[11] Srinivasa Ramanujan, "Highly composite numbers," *Proceedings of the London Mathematical Society*, vol. 14, pp. 347–409, 1915.

[12] Jörg Nonnenmacher, Ernst W. Biersack, and Donald F. Towsley, "Parity-based loss recovery for reliable multicast transmission," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 349–361, Aug. 1998.

[13] Lihao Xu, "Efficient and scalable on-demand data streaming using uep codes," in *Proceedings of ACM Multimedia 2001*, Ottawa, Canada, Sept.–Oct. 2002.

[14] Jehan-François Pâris, Steven W. Carter, and D. D. E Long, "A hybrid broadcasting protocol for video on demand," in *Proceedings of Multimedia Computing and Networking Conference 1999 (MMCN'99)*, 1999, pp. 317–326.

[15] Jack E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, Jan. 1965.

[16] James D. Salehi, Zhi-Li Zhang, James F. Kurose, and Donald F. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 397–410, Aug. 1998.

[17] Jean M. McManus and Keith W. Ross, "A dynamic programming methodology for managing prerecorded VBR sources in packet–switched networks," in *Proceedings SPIE, Performance and Control of Network Systems*, Nov. 1997, pp. 140–154.

[18] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-driven layered multicast," in *Proceedings of ACM SIGCOMM*, Aug. 1996, pp. 117–130.

[19] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashu Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of ACM SIGCOMM*, Vancouver, BC, Canada, Sept. 1999, pp. 56–67.

[20] Supratik Bhattacharyya, James F. Kurose, Donald F. Towsley, and Ramesh Nagarajan, "Efficient rate-controlled bulk data transfer using multiple multicast groups," in *Proceedings of IEEE INFOCOM*, June 1998, pp. 1172–1179.

[21] John C.-I. Chuang and Marvin A. Sirbu, "Pricing multicast communications: A cost based approach," in *Proceedings of INET*, 1998.

[22] Graham Phillips, Hongsuda Tangmunarunkit, and Scott Shenker, "Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law," in *Proceedings of ACM SIGCOMM*, Sept. 1999.

[23] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 770–783, 1997.

[24] "The Mercator Internet mapping project," http://www.isi.edu/scan/mercator/maps.html.

[25] "The Internet Mapping project," http://cm.bell-labs.com/who/ches/map/.

[26] Marcel Waldvogel, Wei Deng, and Ramaprabhu Janakiraman, "Efficient buffer management for scalable media-on-demand," in *SPIE Multimedia Computing and Networking (MMCN 2003)*, Santa Clara, CA, USA, Jan. 2003.

[27] "Java Media Framework," http://www.javasoft.com/products/java-media/jmf/.

[28] Yitzhak Birk and Ron Mondri, "Tailored transmissions for efficient near-video-on-demand service," in *IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, June 1999, pp. 9226–9231.

[29] Anirban Mahanti, Derek L. Eager, Mary K. Vernon, and David Sundaram-Stukel, "Scalable on-demand media streaming with packet loss recovery," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, CA, USA, June 2001, pp. 97–108.

[30] Ernst Biersack, Alain Jean-Marie, and Philippe Nain, "Open-loop video distribution with support of VCR functionality," *Performance Evaluation*, vol. 49, no. 1-4, pp. 411–428, Sept. 2002.

[31] Balachander Krishnamurthy, Craig Wills, and Yin Zhang, "On the use and performance of content distribution networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.