# Fuzzycast: Efficient Video-on-Demand over Multicast

Ramaprabhu Janakiraman
Applied Research Laboratory
Washington University in St. Louis
St. Louis, MO 63130-4899, USA
rama@arl.wustl.edu

Marcel Waldvogel
IBM Research
Zurich Research Laboratory
8803 Rüschlikon, Switzerland
mwl@zurich.ibm.com

Lihao Xu
Department of Computer Science
Washington University in St. Louis
St. Louis, MO 63130-4899, USA
lihao@cs.wustl.edu

***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 optimal server bandwidth. We present a theoretical analysis of its bandwidth and client buffer requirements and prove its optimality. We study the effect of variable bitrate (VBR) media on Fuzzycast performance and propose a simple extension to transmit VBR media over constant-rate channels. Finally, we solve the problem of partitioning a transmission over multiple multicast groups by considering it as a specific instance of a more widely encountered resource trade-off.**

## I. Introduction

The promise of universal broadband networks and fast cheap computation during the last decade 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 get swamped by requests for popular videos, forcing providers to invest in expensive resources to ensure acceptable quality of service during peak load.

Earlier work [1] on requests in video rentals suggests that an 80:20 rule might hold here too: 80% of requests are for the top 20 movies. In a VoD system, this suggests that multicast delivery can help reduce server loads by concurrently serving multiple requests for popular content.

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

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 reasonable free of glitches and skipped frames. This depends on the network connectivity and computational power available to the client.

Start-up delay: While VoD should strictly be instantaneous, most discussion on broadcast-based VoD systems admits of reasonable client wait times between requesting a video and commencement of playout.

Bandwidth usage: This might refer to server bandwidth or client bandwidth or overall load on the ISP network used by the VoD system. From a scalability perspective, server bandwidth usage is the most important metric.

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 till playout and to smoothe over playout jitter introduced by the network. In some broadcast-based schemes, peak buffer requirements can run to several megabytes.

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

*Proactive* multicast protocols [2] are especially attractive in terms of server bandwidth usage [3–7]. 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 [6, 8] 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 stumbling block in deploying these protocols.

In this work, we discuss *Fuzzycast*–a proactive multicast scheme that takes an alternate *discrete* frame-oriented approach to periodic multicasting of video data. We demonstrate that using a discrete approach results in a bandwidth-efficient system that is also easy to build and maintain.

The remainder of this paper is organized as follows: In § II, we introduce and analyze *harmonic broadcasting*, the holy grail for proactive VoD schemes, and explain why its existing approximations either are infeasible in practice or inefficient in design. In § III, we describe *Fuzzycast*, a practicable and efficient version of harmonic broadcasting, and evaluate its performance. In § IV, we consider the effect of variable bit rate (VBR) media on its performance and outline a simple extension for transmitting VBR media. In § V, we propose the problem of optimally partitioning a transmission over a few multicast groups, 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. We discuss related work in § VI and conclude in § VII.

## 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 § 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 of fixed-size blocks of data (*frames*) which– for convenience–are assumed to be transmitted atomically in network packets.

Time is 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 a total of $w + n$ instants on a movie of $n$ frames.

For simplicity, we initially assume that content is constant bit rate; VBR issues will be discussed later in § IV.

For convenience, we also assume that clients arrive synchronous with instants and are able to receive frames transmitted during the instant of their joining. This might lead to underestimating delays by at most one instant. In practice this is insignificant: an instant, by our definition, would correspond to about 33 milliseconds for 30 frames-per-second (fps) video. We shall also omit client decoding time and network setup delay from our analysis.

### 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 playout time $t + w + f - 1$, i.e., $t \leq t_f < t + w + f$. 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 work expands on this simple result, known as *Harmonic Broadcasting* [4].

This is formalized as algorithm IDEAL (Listing 1) below. The schedule generated by algorithm IDEAL (with $w = 0$) 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.

**Listing 1** Algorithm IDEAL

```
for all frames $f_j$ do
    $\lambda \leftarrow j + w$;
    for ($t \leftarrow \lambda; t \leq t_{max}; t += \lambda$) do
        transmit ($t, f_j$);
    end for
end for
```

**Theorem 1** On average, algorithm IDEAL (Listing 1) 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 $\frac{1}{w+f}$ frames/instant. Thus 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-hour 300 kbps Real Media or MPEG-4 movie with a 5-minute initial delay requires a server and client bandwidth of $\approx 1$ 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, algorithm IDEAL:

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

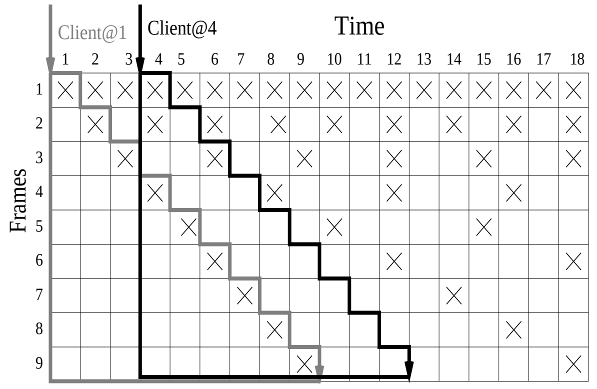
*Proof:* It is easy to prove that algorithm IDEAL is optimal in the sense that a frequency of $\frac{1}{w+f}$ instants for frame $f$ is both necessary and sufficient for on-demand data delivery: necessary because an interval of $w + f$ instants without frame $f$ beginning at time $t$ would cause a client starting at $t$ to just miss $f$; sufficient because, in the absence of interactive functions like *fast forwarding*, each client is guaranteed to play out frame $f$ no earlier than $w + f - 1$ instants after joining. ■

**Theorem 3** Algorithm 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.

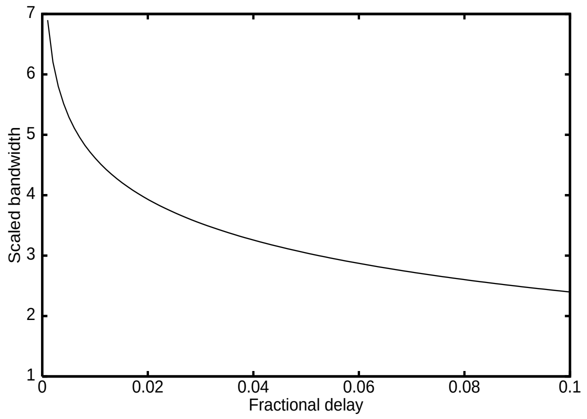
*Proof:* The probability $p(f, t)$ that frame $f$ has reached by time $t$ ($t < f + w$) into the movie is $\frac{t}{w+f}$. Expected buffer space at time $t$ is just the cumulative probability $\sum_{f=t}^{n} p(f, t)$. Thus buffer requirements at the client side are given by:

$$B(t) = \begin{cases} t \log \frac{n+w}{w} & 1 \leq t \leq w, \\ t \log \frac{n+w}{t} & w \leq t \leq n + w \end{cases} . \tag{2}$$

This has a maximum $B_{max} = \frac{n+w}{e}$ at time $\frac{n+w}{e}$. As $w << n$, $B_{max} \approx 0.37n$. ■



(a) Basic Transmission Pattern



(b) Bandwidth vs. Delay

Fig. 1. Fuzzycast Transmission

## C. Existing approaches

Although algorithm IDEAL is simple, elegant, and optimal, 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 $i \geq w$ such that $i$ divides $t$. This function is extremely spiky, varying from $\leq 2$ for prime values of $t$ to record highs for *highly composite* [9] $t$. It is due to this spikiness that earlier research has advanced algorithm IDEAL as a theoretical limit rather than as a practicable scheme [6, 7].

Existing protocols, notably the harmonic broadcasting protocols [4,5], have taken a stream-based approach to get around this limitation. Stream-based protocols, rather than transmit frame (or segment) $f$ every $f + w$ instants, transmit it continuously in a separate channel or stream of bandwidth $\frac{1}{f+w}$. This ensures uniform bandwidth usage, but also runs into difficulties:

In stream-based protocols, the initial delay is a function of segment size. Because user acceptance considerations dictate that initial delay be small compared to movie length, these protocols transmit a movie over many concurrent streams. For example, Polyharmonic Broadcasting [8] transmits a single 2-hour movie with a 5 minute initial delay over 96 streams, with bandwidths varying from a few hundred Kbps to a few hundred bps.

Moreover, 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, revisiting the original problem of infeasible schedules.

Finally, unless error correction techniques like Forward Error Correction (FEC [10]) are used, transmitting a segment over an extended period and over multiple packets increases the chances that a frame is unusable due to partial loss or corruption in transit.

Another stream based protocol, Pagoda broadcasting [11], attempts to pack segments into a few fixed-rate channels deterministically, but sacrifices performance in the process because it has to settle for suboptimal schedules (Fig. 2(d)).

## III. Computing Feasible Frame schedules

As a proactive scheme, the only flaw of algorithm 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 heuristically 'drift' 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 changing the optimal schedule significantly.[1]

**Listing 2** Algorithm BASIC

```
$B_{est} \leftarrow B_{act} \leftarrow 0$;
**for all** frames $f_j$ **do**
  $\lambda \leftarrow j + w$;
  $B_{est} += \frac{1}{\lambda}$;
  **for** $(t \leftarrow \lambda; t \leq t_{max}; t += \delta_t, B_{act}[t]++)$ **do**
    $\delta_t \leftarrow$ FINDNEIGHBOR;
    transmit $(t + \delta_t, f_j)$;
  **end for**
**end for**
```

This is formalized as algorithm BASIC (Listing 2). The crux of it is the FINDNEIGHBOR function, which finds an alternate neighboring time slot for frames that algorithm IDEAL originally schedules 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; Delaying it potentially increases start-up delay for all clients expecting it. The impact of both operations depends on the frame shifted, but in contrasting ways: *Delaying* later frames *increases* the average initial delay more, since more clients wait for these frames. *Advancing* later frames, however, is less harmful since 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 $\delta_a$ and $\delta_d$, which together limit the shifting of frame $f$ out of time slot $t$ between $t - \delta_a(w + f)$ and $t + \delta_d \times w$. Reasonable defaults are $\delta_a \approx 0.05$ and $\delta_d < 0.1$, but these values can be tuned during system setup or configuration taking into account practical limits on server bandwidth and delay variability. For example, variability in

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

**Listing 3** BFSCAN FINDNEIGHBOR FUNCTION

```
$\delta_t \leftarrow \lambda$;
for ($i \leftarrow \lambda$; $i > \lambda - left$; $i - -$) do
    if ($B_{act}[t+i] \le B_{est}$) then
        $\delta_t \leftarrow i$;
        break;
    else if ($B_{act}[t+i] < B_{act}[t+\delta_t]$) then
        $\delta_t \leftarrow i$;
    end if
end for

for ($i \leftarrow \lambda + 1$; $i < \lambda + right$; $i$++) do
    /* Lines 3 through 8 */
end for
return $\delta_t$;
```

**Listing 4** Co-scheduling multiple movies

```
$B_{est} \leftarrow B_{act} \leftarrow 0$;
for all movies $m_i$ do
    $b_{frame} \leftarrow b_{block} \leftarrow 0$;
    $right \leftarrow w_i \delta_d$;
    for all frames $f_j \in m_i$ do
        $\lambda \leftarrow j + w$;
        $left \leftarrow \lambda \delta_a$;
        $B_{est} += \frac{1}{\lambda}$;
        for ($t \leftarrow \lambda$; $t \le t_{max}$; $t += \delta_t, B_{act}[t]++$) do
            $\delta_t \leftarrow$ FINDNEIGHBOR;
            transmit($t + \delta_t, f_j$);
        end for
    end for
end for
```

start-up 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. Some examples are:

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

FBSCAN: Similar to BFSCAN, but start 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 accomodate asymmetric bounds, the spiral is correspondingly distorted. For example, if the advancing limit is 6 frames and delay limit is 3, the sequence of time slots that SPIRAL considers is:

$$(t, t-1, t-2, t+1, t-3, t-4, t+2, t-5, t-6, t+3).$$

It is possible that FINDNEIGHBOR finds no neighbour that can accomodate 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 this seldom happens for reasonable values of $\delta_a$ and $\delta_d$ since both allotted bandwidth and search interval size increase with frame number.

As in Fig. 2(a), these strategies can be represented by different paths from coordinate $(t, t)$ to $(t - \delta_a(w+f), t + \delta_d \times w)$. For example, SPIRAL can be represented by a straight line path between the two points, as mapped by Bresenham's line drawing algorithm [12]. Advancing horizontally or vertically by a "pixel" results in probing the next unprobed time slot in the 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 do neighborhood search. Because of its back-and-forth nature, SPIRAL generates feasible schedules while managing to place frames close to their original time slots.

Listing III shows the implementation of the FINDNEIGHBOR function. For clarity, we have used the simpler BFSCAN algorithm instead of SPIRAL. Fig. 2(c) displays the *bandwidth spectrum*–the distribution of bandwidths over time–for transmitting a 30 fps 2-hour movie with various initial delays.

### A. Co-scheduling multiple movies

Algorithm BASIC has a minor defect that is apparent from Fig. 2(b): by dealing with frames as indivisible units, if the theoretical server bandwidth requirement is even 4.1 frames/instant (say), it schedules 5 frames in some time slots[2], so that peak bandwidth usage overshoots the average by more than 20%. This is easily remedied: if multiple movies are broadcast simultaneously (as is likely in any VoD system), they could be co-scheduled by modifying algorithm BASIC to be aware of both allotted and consumed global bandwidth when making scheduling decisions (Listing III-A). We find that co-scheduling as few as 8 concurrent streams results in a peak bandwidth usage $\approx 2\%$ of optimal (Fig. 3(a)).

## IV. SUPPORT FOR VARIABLE BIT RATE MEDIA

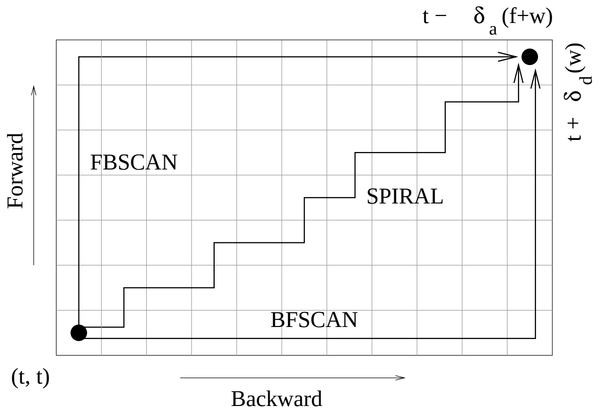
In § III, we made the simplifying assumption that media is encoded at a constant bit rate (CBR). In practice, however, popular media like MPEG-2 and MPEG-4 are VBR: frame sizes are not constant. Algorithm BASIC can be used to transmit VBR frames, provided frame sizes are incorporated into the bandwidth calculation. For a $n$-frame movie with frame sizes $f_1, f_2, \cdots, f_n$, estimated bandwidth for the first $p$ frames is:

$$B_{VBR}(p) = \sum_{i=1}^{p} \frac{f_i}{w+i} \quad . \tag{3}$$

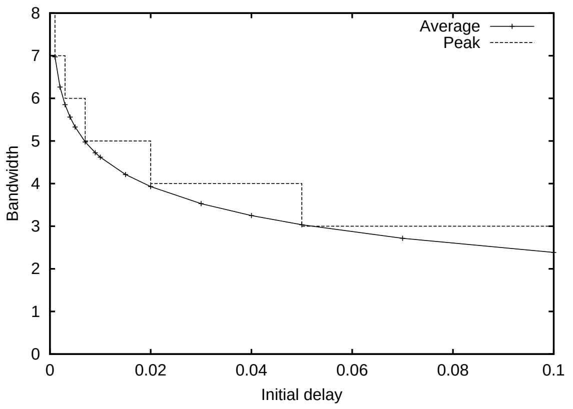
When combined with the global scheduling algorithm, this significantly smoothes bandwidth usage. For example, Fig. 3(b) shows the bandwidth usage (normalized bandwidth predicted according to Eq. (1)) of 1-hour MPEG-4 movie streams, over a 10-hour period. However, clients do not benefit from the smoothing effect of multiple streams; they still suffer from significant bandwidth variability. Variable sized frames also complicate buffer management.

Using a smoothing mechanism like piecewise constant rate transmission (PCRT) [13, 14] is an effective compromise. PCRT smoothes by dividing the media into a few variable-sized segments, which are then transmitted at constant rates. Initial delay and peak bandwidth usage depend heavily on how the movie

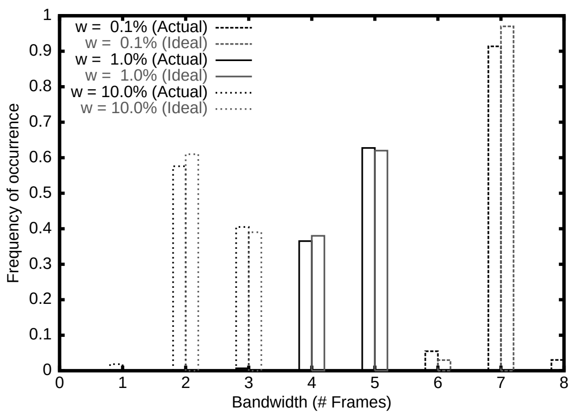
[2] $\approx$ 10% of them



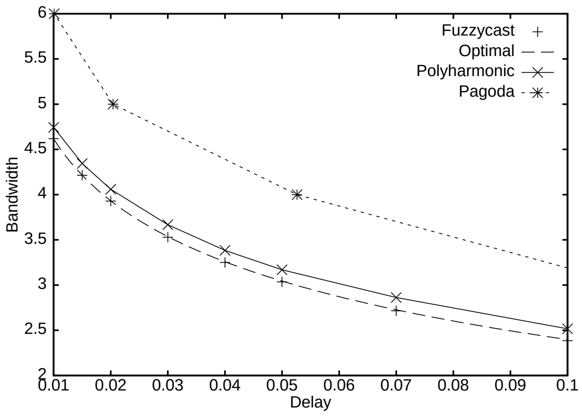
(a) Search strategies



(b) Peak and average bandwidths



(c) Bandwidth spectrum



(d) Performance comparison (Note: The Fuzzycast and optimal graphs are practically superposed)

Fig. 2. Effectiveness of heuristic scheduling

is split [13]. PCRT smooths bandwidth variability effectively (Fig. 3(c)) but the extra initial delay incurred sometimes results in performance overheads exceeding 20%.

We now propose a simpler and more effective solution called *Fragmented Fuzzycast*, which is a straight-forward 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 which 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 $\frac{1}{w+j}$.

**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 on time. If the last frame in block $b_i$ is truncated, transmitting block $b_i$ only guarantees on-time delivery of this fragment. But the rest of this frame, by virtue of being the earliest in block $b_{i+1}$, reaches on time. Since all frames in the last block reach on time, all frames in the movie do too. ■

Fragmented Fuzzycast (Listing 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 of affairs is reversed. Fig. 3(d) 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 graph in Fig. 3(a).

Listing 6 shows a version of the Fuzzycast algorithm supporting co-scheduling multiple streams and fragmenting VBR content.

## V. Multiple groups

Periodic broadcast schemes, while admittedly attractive in terms of server bandwidth usage, waste client and network bandwidth by redundantly transmitting data. While this is unavoidable in a pure broadcast-based system, e.g., a satellite-based distribution network, it is wasteful in a multicast situation where there is network support for subscribing to and unsubscribing from a multicast session. It is therefore desirable that each client explicitly deregister its interest in unwanted frames with the multicast infrastructure.

**Listing 5** FRAGMENT function

```
$b_{block} \leftarrow b_{frame} \leftarrow 0$;
$i \leftarrow 0$;

for all frames $f_j$ do
    while ($b_{block} \geq b_{frame}$ and $j < n$) do
        $b_{frame} += $ size($f_j$);
        $j{+}{+}$;
    end while
    $\lambda_i \leftarrow w + j$;
    $b_{block} += blocksize$;
    $i{+}{+}$;
end for
return $(\lambda_1, \lambda_2, \cdots)$;
```

**Listing 6** Fragmented Fuzzycast

```
$B_{est} \leftarrow B_{act} \leftarrow 0$;
for all movies $m_i$ do
    $(\lambda_1, \lambda_2, \cdots) \leftarrow$ FRAGMENT;
    $right \leftarrow w_i \delta_d$;
    for all blocks $b_j \in m_i$ do
        $B_{est} += \frac{1}{\lambda_j}$;
        $left \leftarrow \lambda_j \delta_a$;
        for ($t \leftarrow \lambda_j; t \leq t_{max}; t += \delta_t, B_{act}[t]{+}{+}$) do
            $\delta_t \leftarrow$ FINDNEIGHBOR;
            transmit($t + \delta_t, f_j$);
        end for
    end for
end for
```

While we would ideally like to transmit each frame in its own multicast group, this creates too much network overhead in the form of group membership messages and state information in the network routers. In practice, each movie is multicast over a small number of groups. Each client initially subscribes to all the groups of a movie and then proceeds to shed each group when it is done with it. This is somewhat similar to existing techniques for receiver-driven congestion-control [15] and efficient data distribution over layered multicast [16, 17].

Our problem, then, is simply stated: Given a movie of $n$ frames, how do we transmit it over $\alpha$ multicast groups in a way that minimizes total redundancy? Since the first few frames have the most bandwidth in our scheduling scheme, our first instinct is to drop early and drop often. On the other hand, we have only a few groups to play with: running out of groups too early will only have exchanged the bombardment of a few high-bandwidth frames for the slow torture of many low-bandwidth ones.

In [18], we consider the problem of partitioning a Fuzzycast transmission over multiple multicast groups. In this section, we show how this problem is really a specific instance of a more general problem that we have termed "Scottie's dilemma."

### A. Scottie's dilemma

In situations involving processes that have a constant 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. For example, psychologists speak of deferring instant gratification for long-term profit. Rocket scientists have to decide when and how often their creations jettison unwanted cargo. File systems periodically synchronize with storage and discard modified buffers.

In general, we find that 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, $\Theta(t) = t$.

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

$$C_\infty = \int_0^T \Phi(t)\, d\Theta(t). \tag{4}$$

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

$$\begin{aligned} C_\alpha &= \sum_{k=1}^{\alpha} \int_{t_{k-1}}^{t_k} \Phi(t)\, d\Theta(t) \\ &= \sum_{k=1}^{\alpha} \big\{\Theta(t_k) - \Theta(t_{k-1})\big\} \Phi(t_{k-1}), \end{aligned} \tag{5}$$

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

Thus the trade-off is reduced to choosing an optimal set of decision points $(t_1^*, t_2^*, \cdots, t_{\alpha-1}^*)$ that minimizes cost $C_\alpha = C_\alpha^*$. Differentiating both sides of Eq. (5) w.r.t $t_k$,

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

Or,

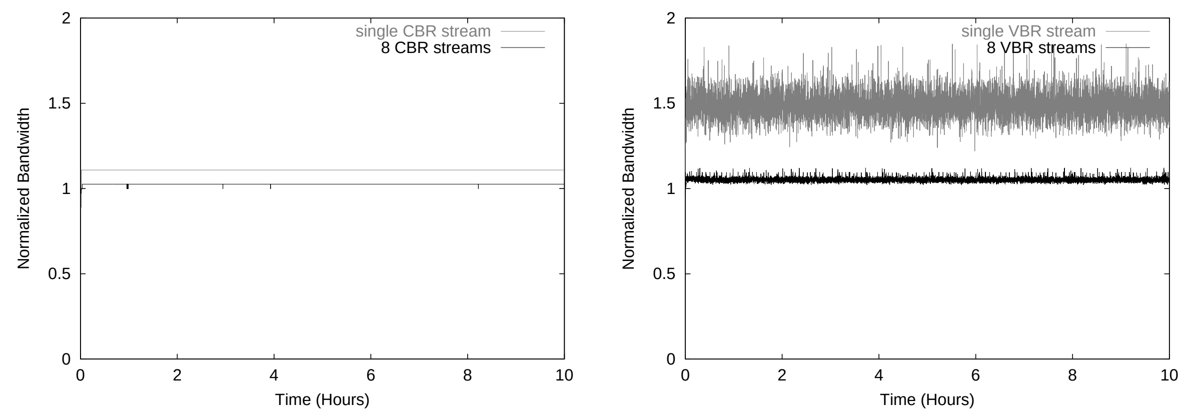
$$\Theta(t_{k+1}^*) = \Theta(t_k^*) + \frac{\Theta'(t_k^*)}{\Phi'(t_k^*)}\big\{\Phi(t_k^*) - \Phi(t_{k-1}^*)\big\}. \tag{6}$$

This recurrence can then be solved for particular cost and utility functions $\Theta(t)$ and $\Phi(t)$, to obtain optimal decision points. For measuring performance, we define inefficiency as follows:

$$I(\alpha) = \frac{\text{Optimal cost with } \alpha \text{ groups}}{\text{Theoretical minimum cost}} = \frac{C_\alpha^*}{C_\infty}. \tag{7}$$

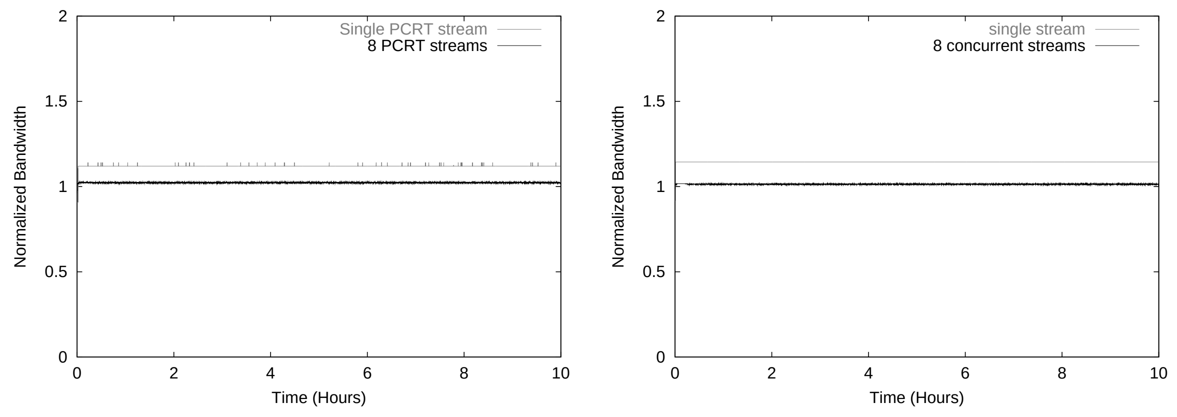
### B. Numerical solution

When closed-form expressions for the optimal boundaries cannot be obtained, we have to settle for obtaining these through numerical methods. Due to the recursive nature of Eq. (6), finding the set of optimal decision points $(t_1^*, t_2^*, ...t_{\alpha-1}^*)$ is reduced



(a) CBR

(b) VBR



(c) PCRT

(d) Fragmented Fuzzycast

Fig. 3. Peak bandwidth for Fuzzycast (one-second peaks)

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. (6). Since it is always true that $t_\alpha(t_1^*) = t_\alpha^* = T$, finding $t_1^*$ reduces to solving

$$t_\alpha(x) - T = 0. \tag{8}$$

This can be done numerically, e.g., using Newton-Raphson iteration. By determining $t_1^*$, we have determined all the optimal decision points.

## C. Case 1: Minimizing Client Load

In this section, we build upon our earlier work [18], where we restrict ourselves to 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 $\alpha$ decision points in which played-out frames can be dropped, we have to choose the points that minimize redundancy.

Let us first find 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 that has not been played out.

Thus, weight and utility functions may be formulated as:

$$\begin{aligned}\Theta(t) &= t \\ \Phi(t) &= \int_t^T \frac{1}{t}\,dt = \log\frac{T}{t} \quad .\end{aligned} \tag{9}$$

In this case, optimal values of drop boundaries are given by:

$$t_{k+1}^* = t_k^*\{1 + \log\frac{t_k^*}{t_{k-1}^*}\} \quad . \tag{10}$$

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

$$t_\alpha^* = t_1^* \underbrace{(1 + \log\frac{t_1^*}{t_0^*})\,(1 + \log(1 + \log\frac{t_1^*}{t_0^*}))\,\cdots}_{\alpha \text{ terms}}, \tag{11}$$

where $t_\alpha^* = n + w$ and $t_0^* = w$. For convenience, we assumed time starts from $w$. This result is identical to that obtained from first principles in [18].

Using the method outlined in § V-B, Eq. (11) can be solved numerically to obtain $t_1^*, \cdots t_{\alpha-1}^*$. This set of boundaries is the

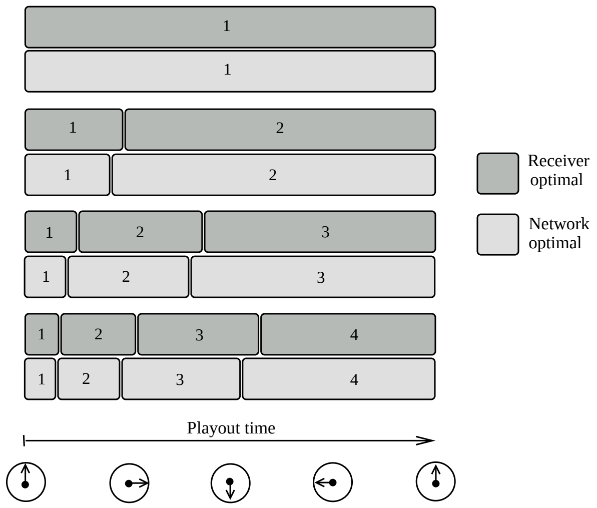


Fig. 4. Optimal partition (1–hour movie, 36 second delay)

one that minimizes the number of frames that each client receives. For example for a one-hour, 30 fps movie with $\alpha = 3$ and $w = 36$ seconds (1%), the optimal group boundaries are at 7:34, 26:46, and 60:36 minutes, leading to an average client bandwidth usage of 54 fps as opposed to 165 fps without layering, ($\approx 67\%$ reduction).

To measure performance gain in this case, we use Eq. (7) to get *Receiver inefficiency*:

$$C_\alpha^* = \sum_{k=1}^{\alpha} t_k^* \log \frac{t_k^*}{t_{k-1}^*}$$
$$C_\infty = \int_w^{n+w} \log \frac{n+w}{t}\, dt \tag{12}$$

$$I_R(\alpha) = \frac{\text{\# frames received on average}}{\text{\# frames in movie}} \approx \frac{1}{n}\sum_{k=1}^{\alpha} t_k^* \log \frac{t_k^*}{t_{k-1}^*}.$$

Fig. 5(a) plots receiver inefficiency against $\alpha$ for various initial delays. Fig. 5(b) shows the values of inefficiency obtained through simulation. There is excellent agreement between predicted and experimental values. We also find that there is a "sweet spot" at around 4-5 groups where maximum gains are obtained. Increasing $\alpha$ further does not increase performance much.

### D. 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. That is, we would like to minimize the number of frames on the network at any given time. This again is discussed specifically in [18].

If the number of links in a delivery tree of $m$ clients is $L(m)$ and the average client arrival rate is $\lambda$ 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, two end users on a single local network count as a single client.

A seminal result due to Chuang and Sirbu [19] 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.) This represents its network bandwidth advantage over multiple unicast, which has $L(m) = \hat{u}m$. This was subsequently verified by Phillips et. al. [20].

Now, we can set up the weight function as simply the number of links in a group of time $t$, i.e., for the network optimal case,

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

This results in the recurrence:

$$t_{k+1}^* = t_k^*(1 + \rho \log \frac{t_k^*}{t_{k-1}^*})^{\frac{1}{\rho}} \tag{13}$$

Or,

$$t_\alpha^* = t_1^* \underbrace{(1 + \rho \log \frac{t_1^*}{t_0^*})^{\frac{1}{\rho}} (1 + \log(1 + \rho \log \frac{t_1^*}{t_0^*}))^{\frac{1}{\rho}} \cdots}_{\alpha \text{ terms}}. \tag{14}$$

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

To measure performance, we obtain *Network inefficiency* from Eq. (7) as:

$$C_\alpha^* = \sum_{k=1}^{\alpha} (t_k^*)^\rho \log \frac{t_k^*}{t_{k-1}^*}$$
$$C_\infty = \int_w^{n+w} \log \frac{n+w}{t} d\{t^\rho\} \tag{15}$$

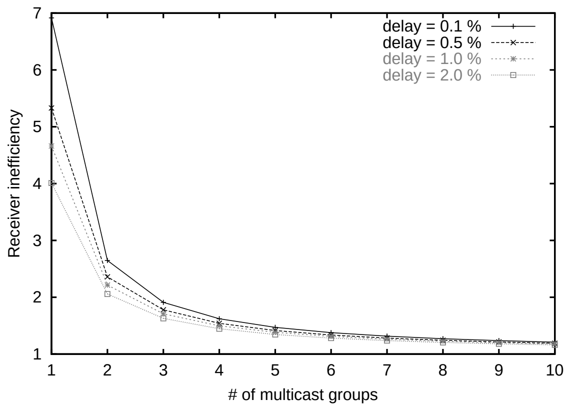
$$I_N(\alpha) = \frac{\text{\# frames in network at any time}}{\text{minimum \# frames in network}} \approx \frac{\rho}{(n+w)^\rho}\sum_{k=1}^{\alpha} (t_k^*)^\rho \log \frac{t_k^*}{t_{k-1}^*}.$$

Fig. 5(c) shows the network inefficiency versus $\alpha$ for various $w$. Fig. 5(d) shows the values obtained from simulation over realistic network topologies, using the GT-ITM [21] simulator and from traces obtained from the SCAN [22] project. Details about our simulation setup are given in [18]. 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 or 5 groups, beyond which increasing $\alpha$ does not seem to have much effect.

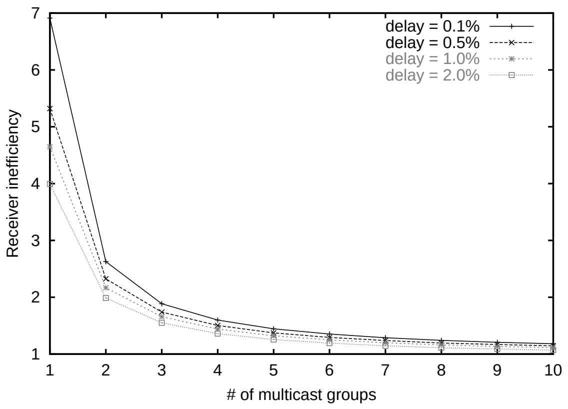
### E. Comparing receiver-optimal and network-optimal cases

In Fig. 4, we compare partitions in the receiver and network-optimal cases. It is apparent from the figure that the boundaries for the network-optimal case are earlier than the corresponding receiver-optimal boundaries. This is in fact always true and can be easily proved by letting $z_k = \frac{t_k}{t_{k-1}}$ in both cases, so that both Eq. (10) and Eq. (13) reduce to the form:

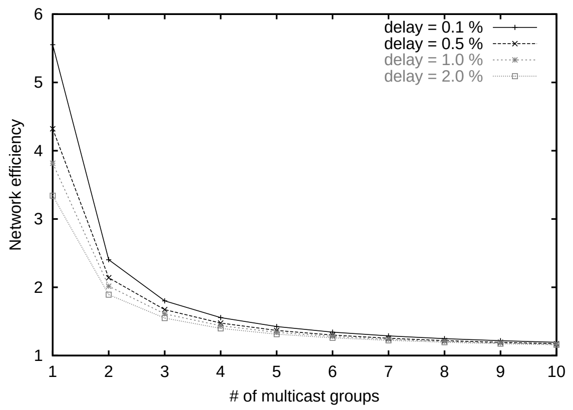
$$z_{k+1} = (1 + \rho \log z_k)^{\frac{1}{\rho}}$$



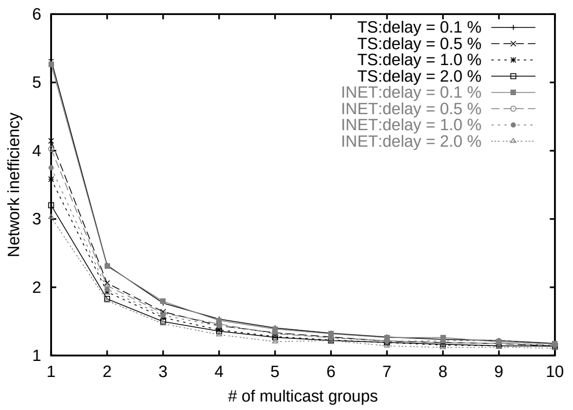
(a) Predicted: $I_R(\alpha)$ vs. $\alpha$



(b) Actual: $I_R(\alpha)$ vs. $\alpha$



(c) Predicted: $I_N(\alpha)$ vs. $\alpha$



(d) Actual: $I_N(\alpha)$ vs. $\alpha$

Fig. 5. Performance with multiple groups

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 increasing function of $\rho$, from which the result immediately follows.

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

### F. Variable arrival rate

The analysis in the previous section is not wholly accurate in that we have assumed uniformly distributed arrival rates, while in reality, clients arrive in a process centered around a mean $\lambda$. In [18], we show that switching to Poisson arrivals does not have a significant impact on these results.

## VI. Related Work

One of the earliest proposals for bandwidth-efficient VoD was *Batching* [1], 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* [4, 5] family of protocols (already discussed in § 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 have also been obtained in [6, 7].

Pre-push: Several commercial pay-per-view networks are currently testing "on-demand" models, where 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 amounts of storage are required to store enough data for a decent selection of movies to be offered. Moreover, while most demand at any given time is for a small set of movies, this set is a moving target, defeating attempts at any long-term client-side caching.

Smoothing VBR video: While there is a large body of work on smoothing unicast transmission of VBR video, the impact of VBR media on the performance of proactive multicast schemes has never been properly studied.

Layering over multiple groups: We discuss the specific problem of client and network optimal partitioning over multicast groups for Fuzzycast in [18]. As far as we know, there has not been work prior to this on quantifying the network impact of these VoD protocols or on optimally distributing content among multiple multicast groups. Bhattacharyya et al. [17] discuss optimal scheduling of data packets in a layered multicast [15] transmission to receivers with identical starting times.

CONTENT DISTRIBUTION NETWORKS: Content distribution networks (CDNs [23]) are an alternate 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 is currently an area of active research.

BULK DATA DISTRIBUTION: Byers et al. proposed a digital fountain approach to data distribution [16], where 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 where data only needs to be reconstructed all at once at the end of transmission, it does not seem to be readily applicable to streaming media applications where clients consume data piecemeal.

## VII. CONCLUSIONS

The success of Video-on-Demand (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 to the first part, while functionality, ease of use, and quick response to user commands are needed to satisfy the second.

Proactive VoD protocols are attractive from a scalability point of view, because they use server bandwidth efficiently to serve media in 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-optimal server bandwidth while remaining relatively simple to implement and maintain.

While 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, *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 due 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 optimal solution to our problem. We find that using even a few multicast groups results in significant reduction in overhead for both the client and the network. We note that the result obtained here is quite general and is capable of straightforward application to diverse situations, including networks that follow scaling properties very different from the Chuang-Sirbu law.

Using the optimal 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] 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.

[2] 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*, 2001.

[3] Kien A. Hua and Simon Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of SIGCOMM '97*, Sept. 1997, pp. 89–100.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-driven layered multicast," in *Proceedings of SIGCOMM '96*, Aug. 1996, pp. 117–130.

[16] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A digital fountain approach to reliable distribution of bulk data," in *SIGCOMM '98*, 1998, pp. 56–67.

[17] 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 '98*, June 1998, pp. 1172–1179.

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

[19] John C.-I. Chuang and Marvin A. Sirbu, "Pricing multicast communications: A cost based approach," in *Proceedings INET '98*, 1998.

[20] Graham Phillips, Hongsuda Tangmunarunkit, and Scott Shenker, "Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law," in *ACM SIGCOMM '99*, 1999.

[21] 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.

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

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