Optimal Patching Schemes for Efficient Multimedia Streaming [1]

Subhabrata Sen[1], Lixin Gao[2], Jennifer Rexford[3], and Don Towsley[1]

[1] Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
{sen,towsley}@cs.umass.edu

[2] Dept. of Computer Science
Smith College
Northampton, MA 01060
gao@cs.smith.edu

[3] Networking & Distributed Systems
AT&T Labs – Research
Florham Park, NJ 07932
jrex@research.att.com

Abstract— **Multimedia streaming applications consume a significant amount of server and network resources due to the high bandwidth and long duration of audio and video clips. Making streaming services economically viable requires techniques for minimizing the incremental cost of serving a new client, particularly for popular content. Patching [1] reduces server and network overhead by allowing a client to receive (part of) a multimedia stream by listening to an ongoing transmission of the same clip, without increasing client playback delay. However, existing patching schemes [1–3] do not fully exploit the client buffer space or the ability to listen to more than one ongoing transmission, for reducing bandwidth overheads. In this paper, we first introduce** *Periodic Buffer Reuse* **(PBR) patching that maximizes the amount of data that a client can retrieve from the ongoing transmission. Similar to the existing schemes, PBR can employ a threshold to determine when to start a new complete transmission of the stream. We derive a closed-form expression for the transmission bandwidth requirements for PBR patching, and show how to determine the optimal threshold value. Our performance comparison demonstrates that PBR can significantly outperform existing patching schemes. We then present** *Greedy Buffer Reuse* **(GBR), an algorithm that allows clients to patch to multiple ongoing transmissions. We show that this algorithm provably minimizes the server and network transmission bandwidth requirements. Simulation experiments demonstrate that GBR patching offers a sizeable reduction in transmission overhead over any of the threshold-based schemes, and rarely requires the client to listen to more than three simultaneous transmissions, for the scenarios we examine.**

I. Introduction

A pervasive high-speed networking infrastructure and a mature digital video technology has led to the emergence of several networked multimedia applications which include streaming video as an integral component. Examples of such applications include live video broadcasts, distance learning, corporate telecasts, narrowcasts, and streaming of Web video clips. Video streams typically have high bandwidth requirements even when compressed (e.g., $4-6$ Mbps for MPEG-2), and such flows can be relatively long lived, making it expensive to deliver multimedia content. In addition, many applications have asynchronous clients that may request a video stream at different times. Still, particularly for popular clips, the client requests may arrive close together in time relative to the duration of the stream. Making high-volume video services economically viable requires effective techniques that minimize the incremental cost of serving a new client, while also limiting the client start-up latency and the likelihood of rejecting requests due to resource constraints.

For popular video streams, the server and network resources can be significantly reduced by allowing multiple clients to receive all, or part of, a single transmission [4–9]. For example, the server could *batch* requests that arrive close together in time [4], and multicast the stream to the set of clients. In addition to reducing server load, batching also lowers network overheads, particularly when clients share one or more links in common. However, batching must trade off the client playback latency against the ability to aggregate successive requests. Other approaches exploit the client's buffer space and the ability to listen to multiple simultaneous transmissions, either by passively listening on a shared medium or by joining/leaving multiple multicast groups. In *periodic broadcast* schemes, the server continuously broadcasts segments of the video on a collection of transmission channels [6–9]. Clients can listen to multiple channels at the same time and store segments for later playback. To limit start-up latency, the short initial segments are repeatedly more frequently than later, longer ones. Hence, achieving a low start-up latency requires a larger number of channels, which increases the load on the server and the network.

This paper explores a technique for reducing server and network transmission bandwidth usage for disseminating a video stream to multiple asynchronous clients, without introducing any client startup delay. Known as *patching* [1], this involves using client-side *workahead buffering* to allow a new client to receive (part of) its future playback data requirement by listening in to an *existing* ongoing transmission of the same video, with the server transmitting afresh only the remaining required frames. For example, suppose a multimedia server begins streaming a two-hour video clip to a requesting client, and a second client requests the same video ten minutes later.

[1]The work at the University of Massachusetts was supported in part under National Science Foundation grants NCR-9523807, NCR-9508274, CDA-9502639 and ANI-9805185. Lixin Gao was supported by National Science Foundation grant NCR-9729084 and NSF CAREER Grant ANI-9875513. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

Rather than transmitting the entire video a second time, the server could stream just the first ten minutes of the clip to the second client, while at the same time have this client retrieve and store (for a short period of time) the remaining video frames from the ongoing transmission of the *complete* video to the first client. This capitalizes on buffer space at the client site to store a ten-minute sliding window of frames from the ongoing transmission, and sufficient client I/O bandwidth for listening to two simultaneous transmissions for the first ten minutes. As a result, fewer server and network resources are required to satisfy the clients. Unlike batching, patching allows a client to begin playback immediately by receiving the initial video frames directly from the server. Similar to periodic broadcast schemes, patching exploits the client buffer space to store future frames from other video transmissions. Unlike periodic broadcasting, the server transmits video data only on-demand, when new clients arrive.

Existing patching schemes [1–3] do not fully exploit the client buffer space or the ability of the client to listen to more than one ongoing transmission, for reducing the transmission bandwidth requirements. We refer to these earlier algorithms as Restricted Buffer Reuse (RBR) schemes. In Section II, we discuss these algorithms in more detail, and introduce a general model for designing patching services. Then, we propose and analyze two new patching algorithms that capitalize on the client buffer space to maximize the portion of the video that can be received from ongoing transmissions of the same video. First, in Section III, we present the Periodic Buffer Reuse (PBR) algorithm that maximizes the amount of data that a subsequent client can receive from the existing *complete* transmission, even if the client buffer is not large enough to store the entire sliding window of frames. As with earlier patching schemes, PBR can be used in conjunction with a thresholding policy that determines when a client request should trigger a new complete transmission of the video stream. We derive a closed-form expression for the transmission overheads under our proposed patching scheme, and show how to compute the threshold value that minimizes the expected aggregate transmission bandwidth required to satisfy each client. Based on this analysis, we compare our approach to an RBR patching policy that uses an optimal threshold [2].

Although PBR maximizes the amount of data that a client can receive from an existing *complete* transmission, it does not exploit the potential of receiving data from more than one ongoing transmission. In Section IV, we present an *optimal* patching algorithm, *Greedy Buffer Reuse* (GBR) that allows the client to receive frames from multiple ongoing transmissions. The algorithm is optimal in that no other patching scheme can satisfy any given sequence of client requests with a lower transmission bandwidth usage. We present a proof of optimality, and compare the performance of the optimal algorithm to PBR. The simulation experiments show that GBR patching offers a sizeable reduction in server and network overhead, and rarely requires the client to listen to more than three ongoing transmissions, for the configurations we examine. This optimal algorithm provides a lower bound on the achievable transmission overhead, and can thus serve as the basis of new patching algorithms with lower computational complexity, and simpler implementation. In Section V, we describe our ongoing research on extending and evaluating both PBR and GBR patching. Section VI concludes the paper with a brief summary of the contributions of the work.

II. Patching Model

In this section, we first present related work on patching algorithms. We then describe a practical setting for video patching services, based on the capabilities of today's personal computers and network support for multicast. Finally, we provide a brief summary of our patching model, which serves as the basis of the new patching algorithms presented in Sections III and IV. The new schemes can fully capitalize on the available client buffer space, as well as the ability of the client to switch between different multicast groups, to reduce the aggregate transmission bandwidth required to serve a new client.

A. Related Work

A patching scheme dictates which video frames can be retrieved from an ongoing transmission, and when the patching server should initiate a new complete transmission of the entire video. Existing patching schemes [1–3] limit the client to listen to a contiguous set of frames from a *single* ongoing transmission; the remainder of the frames must be retrieved from the content server. For example, assume a discrete-time model at the granularity of one frame time (e.g., 33 msec for a 30-frame/second video), where the client has a B-frame buffer. Suppose that a client arrives t frame times after the beginning of the most recently started complete transmission of an N-frame video, initiated at the request of another client. Under existing patching policies, if $t \leq B$, the initial t frames are sent directly by the server, and the remainder of the frames are retrieved from this ongoing complete transmission. If $t > B$, then the client receives only the last B consecutive frames of the video from the existing transmission, and the server must supply the rest of the frames to the new client. Thus, in this latter case, only the final B frames of the video are shared by the two clients.

Existing patching schemes include a second critical component – a threshold value $T \in \{0, 1, \ldots, N-1\}$ is used to determine when to initiate a new complete transmission of the entire video to satisfy a client request. The server does not start a new complete transmission unless the client request arrives more than T frame times after the beginning of the most recently initiated complete transmission. All other clients with $t \leq T$ patch onto the earlier complete transmission. Existing patching schemes differ in the particular threshold chosen. Under *Greedy* patching [1, 3] a client patches to an ongoing transmission whenever possible (i.e., $T = N - 1$); hence, at any time, there is at most one ongoing complete transmission of the video from the server. In contrast, *Grace* patching [1, 3] starts a new complete transmission whenever a client arrives more than B time units after the start of the last complete transmission (i.e., $T = B$), rather than having the client patch only to the last B frames of the earlier ongoing complete transmission. Therefore, there may be multiple ongoing complete transmissions at any time for *Grace*. Grace patching typically outperforms greedy patching, by allowing future client requests to benefit from the start of a new transmission. Generalizing the idea of threshold-based patching, it is possible to determine the value of T that minimizes the average transmission required to serve a client, as a function of the request arrival rate λ, the client buffer size B, and the length of the video N. For a Poisson arrival process, it is possible to derive a closed-form expression for the optimal value of T [2].

B. *Practical Setting*

Central to any effective patching scheme is the ability of the client to listen to multiple transmission channels simultaneously, and to store frames ahead of their playout time. Patching operates well within the buffer space and I/O bandwidth of today's end systems. Both per-byte storage cost and access latencies for main memory and disk are decreasing dramatically. In addition, system bus speeds are also increasing. Commodity PCs already offer 100 MHz system bus and 64–128 megabytes of main memory, as well as several gigabytes of disk storage. These trends suggest that a significant segment of client stations have enough high-bandwidth storage space to accommodate several minutes worth of high-quality streaming video. These clients also have sufficient I/O and disk bandwidth to listen to multiple transmission channels simultaneously. For example, the ubiquitous Ultra ATA IDE disk interfaces offer about 33 Mbps. Newer PCs can support transfer rates of 40–100 Mbps with Ultra SCSI or Fiber Channel I/O interfaces.

Transmission of a video to a set of clients is coordinated by a *patching server*, located at the multimedia source or at a proxy inside the network. Employing patching functionality at a proxy is useful in various situations, e.g., to achieve patching gains while streaming video from a conventional content server which does not employ patching itself. Proxy-based patching is also useful if multicasting capability is not available on an end-end basis from the content server to clients. For example, in a heterogeneous internetworking environment, a proxy server in a domain close to the clients may receive the video on a unicast connection from the content server, and multicast the stream to downstream clients. Performing patching at the proxy reduces the bandwidth consumed on the path from the multimedia source to the proxy and from the proxy to the clients. In some cases, the proxy can provide a patching service without requiring the cooperation of the server (and hence any modifications to existing server sites) by issuing requests for the appropriate frames of the video (e.g., using a protocol such as RTSP [10]). This is advantageous and suggests that proxy patching services can be deployed incrementally in the network. Figure 1 shows a patching server connecting to a collection of asynchronous clients over a multicast-capable network.

The selective acceptance of different video frames by the client can be achieved in several ways. In one approach, the patching server transmits video frames in various multicast groups, with clients joining and leaving the groups to receive the appropriate frames. Alternatively, the client can listen to all server transmissions of the video, and use a local filter to decide which frames to keep. This model is particularly appropriate for clients on a shared media, such as an Ethernet or a cable access network. In the general case, when the clients are not on a shared media, proxies inside the network can filter the transmission to avoid sending unnecessary frames to the downstream clients. In this paper, we assume that join and leave latencies are small, or that patching is performed by a proxy that transmits frames to clients on a shared media. Section V considers extensions that incorporate join and leave latencies.

C. *System Model*

We next provide a formal model of the patching system, and introduce notation and key concepts that will be used in the rest of the paper. Without loss of generality, we consider a discrete-time system at the granularity of a frame time (e.g., 33 msec for a 30-frame/second video). We focus on patching for a single N-frame video, since requests for different streams do not interact. Suppose that client i requests the video stream at time t_i, and plays frame j at time $t_i + j$, where $j = 1, 2, \ldots, N$. Lossless, starvation-free playback is guaranteed if frame j is received at the client(s) by time $t_i + j$. Frames that arrive before their playback time are stored in the client's B-frame workahead buffer. When multiple clients arrive simultaneously, the requests are served as a single batch. Without loss of gener-

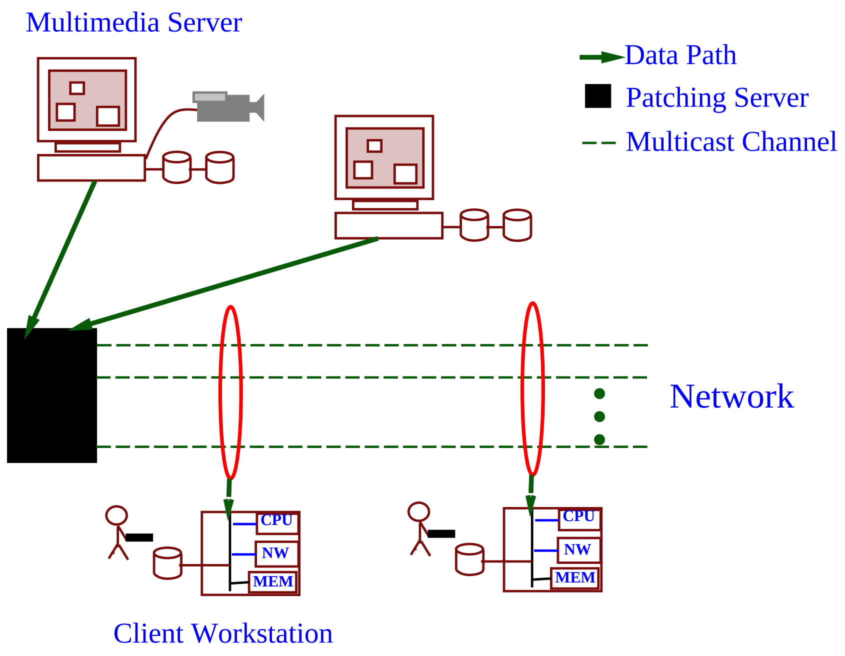


Fig. 1. **Video Patching Service:** Video streams originate at a multimedia server, and travel through the network, to multiple asynchronous clients. Patching is performed with the help of a patching server located at the source, or at a proxy server inside the network.

ality, we number the client batches in increasing order of the request time (i.e., $t_1 < t_2 < t_3 \ldots$).

We define the *most-recent complete* (MRC) transmission for a client i as the *most recently initiated* existing transmission of the entire video at the time of client i's arrival. Existing patching schemes limit the client to listen to a contiguous set of frames from *one* ongoing transmission – its MRC transmission. A new multicast-capable transmission channel is created to transmit whatever frames client i cannot receive from existing transmissions. In our model, a *transmission channel* is a logical entity that may intermittently transmit different segments of the video. Multiple channels can therefore be potentially time-multiplexed over the same underlying network bandwidth.

When a new client i arrives, the patching server computes a *reception schedule* $\mathbf{RS}_i$ for that client, and a *channel transmission schedule* $\mathbf{CS}_i$ corresponding to a new channel started for the client, using some patching algorithm. The transmission schedule for channel C_i specifies which frames are transmitted on that channel, and the scheduled transmission times. Depending on the service model, these frames are either requested directly by the client, or scheduled for transmission by the server. For example, consider a client request that arrives $t \leq B$ time units after the start of the MRC transmission under grace or greedy patching. Then, the server transmits frame j at time $t_i + j$ on channel i, resulting in $CS_{i,j} = j$ for $j \leq t$. The server does not transmit the remaining frames to client i, resulting in $CS_{i,j} = 0$ (idle slot) for $j \in \{t+1, t+2, \ldots, N\}$.

The server (or the client) also computes a *reception schedule* $\mathbf{RS}_i$ for each client request i, to specify what frame(s) the client should receive during each time slot. Each entry $RS_{i,j}$ is a set of one or more pairs (k, l), indicating that frame k should be retrieved from channel C_l at time $t_i + j$, for $j = 1, 2, \ldots, N$. For example, suppose channel m is the MRC transmission for client i. Then, under grace or greedy patching with $t \leq B$, we have $RS_{i,j} = \{(j,i), (j+t, m)\}$ for $j \leq t$, since the client must receive the first t frames directly from the server on channel C_i, and simultaneously receive the next t frames from the MRC transmission on channel m. The remaining frames are retrieved from the MRC transmission, resulting in $RS_{i,j} = \{(j+t, m)\}$ for $j > t$. Note that $RS_{i,j}$ has at most two entries for these two algorithms – from the MRC channel and/or the transmission channel C_i, and that the client retrieves a single contiguous set of frames from each channel.

Existing patching algorithms [1–3] do not fully exploit the presence of the client workahead buffer to reduce the additional transmission from the server, particularly when $t > B$. As such, we refer to the existing threshold-based patching algorithm using optimal thresholding [2] as RBR (Restricted Buffer Reuse) in the remainder of the paper. We now present new patching algorithms that exploit the client buffer space, and the ability to listen to multiple multicast groups, for more effective patching. In the next section, we propose a new patching algorithm that can retrieve multiple contiguous sets of frames from the MRC channel and/or the transmission channel C_i. Then, in Section IV, we present another patching algorithm that allows a client to retrieve frames from any of the existing active channels.

III. Periodic Buffer Reuse (PBR) Patching

In this section, we present a new threshold-based patching scheme that shares the following restriction with the existing schemes – a new client is restricted to patch to only its MRC transmission. The new scheme, called *Periodic Buffer Reuse* (PBR) maximizes the number of frames that a new client can retrieve from this most recently initiated ongoing complete transmission, by exploiting client-side buffering more effectively. After describing the algorithm, we derive an analytic expression for the transmission bandwidth requirements for streaming the video to a set of requesting clients. Based on these results, we compare our proposed patching scheme to RBR patching.

A. Periodic Buffer Reuse with Thresholding

When a client arrives more than B time units after the start of the most-recently initiated complete transmission of a video, existing patching schemes buffer at most the last B frames of the ongoing transmission. This may be too conservative, in that the client buffer space may remain empty for a long time until the existing channel starts transmitting the last B frames of the video. Existing patching policies always retrieve a contiguous set of frames from the MRC transmission, and require the client to receive a separate transmission of all remaining frames. As such, these schemes do not fully exploit the client buffer to reduce the amount of new transmission required from the server. We now present a new patching scheme that maximizes the number of frames that are retrieved from the existing complete transmission, even when the client arrives more than B time units after the complete transmission began. In this scheme, the client retrieves a frame from the earlier transmission whenever buffer space is available. Whenever the client must receive parts of the video from the server, these frames are retrieved *as late as possible*, just before their playback times.

A reception schedule and corresponding channel transmission schedule must be computed based on the client's buffer size B, and arrival time t *relative to the beginning* of its MRC transmission. When $t \leq B$, the client receives the first t frames from the server, and the remaining $N-t$ frames from the MRC transmission, as in RBR patching. When $t > B$, the client still must receive the first t frames directly from the server. Simultaneously, during the first B time units after the request is made, the next B frames are received from the MRC transmission, and buffered at the client. At time $t+B$, the client buffer is full, since the client is still playing frames retrieved directly from the server. After $t-B$ additional time units, the client can start draining its buffer contents, and receive additional frames from the MRC transmission. Frames which are transmitted on the MRC channel between time $t+B$ and $2t$ cannot be received by the client due to lack of workahead buffer space, and are therefore fetched directly from the server, just before their respective playback times at the client. The process repeats in a periodic fashion, with the client receiving frames $it+1,\ldots,it+B$ from the ongoing transmission, and frames $it+B+1,\ldots,(i+1)t$ directly from the server, for $i = 1, 2, \ldots$, as shown in Figure 2.

As in RBR, our proposed scheme includes a threshold T. The patching service initiates a new complete transmission to serve the new client whenever $t > T$. Later in the section, we show how to compute the optimal value of T for this algorithm.

B. Transmission Overhead for a Client

Intuitively, PBR patching attempts to keep the client buffer full at all times. Since the client is t time units behind its MRC transmission, a frame must reside in the buffer for t time units before it is consumed, freeing the space for storing another frame. To quantify the benefits of PBR patching, we derive an expression for $D(t)$, the number of frames transmitted by the server to a client that arrives t time units *after* the initiation of its MRC transmission. PBR has the same performance as RBR for very large and very small values of t. In particular, if $t \leq B$, the server transmits only the first t frames, resulting in $D(t) = t$. Similarly, when $t \in \{N-B+1, N-B+2, \ldots, N-1\}$, the client can receive the last $N-t$ frames from the ongoing MRC transmission, and $D(t) = t$. Finally, if the client buffer can store at least half of the stream (i.e., $B \geq N/2$), then $D(t) = t$ even if $t > B$, since the client buffer is large enough to store all remaining $N-t$ frames of its MRC transmission. Hence, when $t \leq B$, $t \in \{N-B+1, N-B+2, \ldots, N-1\}$, or $B \geq N/2$, we have $D(t) = t$, and the client buffers a single contiguous set of frames from its MRC transmission, as in RBR patching.

The difference between PBR and RBR patching arises when $t \in \{B+1, B+2, \ldots, N-B\}$ and $B < N/2$, when RBR can only buffer the last B frames from the ongoing stream. Under PBR patching, the remainder of the ongoing transmission is effectively divided into periods of length t, where B frames are buffered from the ongoing transmission in each interval. Following the first t frames, the remainder of the stream consists of $\lfloor \frac{N-t}{t} \rfloor$ complete intervals of length t. Then, the end of the stream consists of a partial interval of length $(N-t) \bmod t$. During this partial interval, the client can buffer up to B frames from the ongoing transmission, and must retrieve any remain-

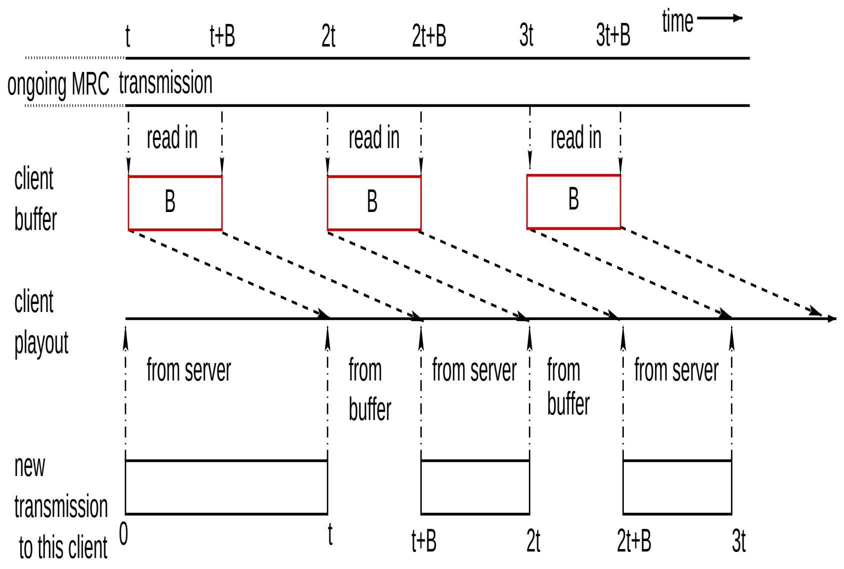


Fig. 2. **PBR Patching:** This figure shows how PBR patching operates for a client with a B-frame workahead buffer arriving $t > B$ time units after the start of its MRC transmission. Frames are retrieved either from the MRC transmission, or from a new server transmission channel.

ing frames directly from the server. Consequently,

$$D(t) = \begin{cases} t, & \text{if} \quad B \geq N/2, \quad t \leq B, \\ & \text{or} \quad N - B < t < N, \\ N - (\lfloor \frac{N-t}{t} \rfloor B + \min\{(N-t) \bmod t, B\}), & \\ & \text{if} \quad B < N/2 \quad \text{and} \quad B < t \leq N - B \end{cases}$$

For long video clips (large N), the contribution of the $\min\{(N-t) \bmod t, B\}$ term is small, and the difference between $\lfloor \frac{N-t}{t} \rfloor$ and $\frac{N-t}{t}$ is insignificant. Hence, we approximate $D(t)$ with $N - B(N-t)/t$ for $B < N/2$ and $B < t \leq N - B$. This simplifies to $N + B - NB/t$, resulting in

$$\tilde{D}(t) = \begin{cases} t, & \text{if} \quad B \geq N/2, \quad t \leq B, \\ & \text{or} \quad N - B < t < N, \\ N + B - NB/t, & \\ & \text{if} \quad B < N/2 \quad \text{and} \quad B < t \leq N - B \end{cases}$$

We use the expression for $\tilde{D}(t)$ to simplify the analysis in the next subsection.

C. Average Transmission Requirement for a Client

To evaluate the PBR patching policy, we derive a closed-form expression for the transmission bandwidth requirements as a function of the client buffer size (B), the length of the video (N), the threshold (T), and the request arrival distribution. The performance metric we consider is $\overline{W_c}$, the average amount of data transmitted to each client using the PBR policy.

The analysis models a discrete-time system, where $\{X_j\}_{j=0}^{\infty}$ are discrete random variables denoting the number of client arrivals at times $0, 1, \ldots$ respectively. Let $\{k_i\}_{i=0}^{\infty}$ denote the times at which the server schedules a transmission of the complete N-frame video. Under PBR, k_0 is the time of the first request for the video (i.e., $k_0 = \min\{\tau | (\tau \geq 0) \text{and} (X_\tau > 0)\}$). A new complete transmission of the video is initiated in response to the first client request that occurs more than T time units after the previous complete transmission (i.e., $k_{i+1} = \min\{\tau | (\tau > k_i + T) \text{and} (X_\tau > 0)\}$, for $i = 0, 1, \ldots$). For analytic tractability, we assume that the X_is are independent and identically distributed. Note that under PBR patching, clients arriving at times $k_i, k_i + 1, \ldots, k_i + T$ patch to a different complete transmission than clients arriving at times $k_{i+1}, k_{i+1} + 1, \ldots, k_{i+1} + T$. In this sense, the behavior of the patching system after time k_i is independent of the behavior before time k_i. This fact and the *i.i.d.* assumptions allow us to model the patching system as a renewal process with renewal points at $\{k_i\}_{i=0}^{\infty}$. The analysis focuses on the first complete transmission of the video at time k_0 (which is initiated for the batch of clients arriving at time k_0), and the subsequent client requests that patch to this transmission.

To aid in deriving an expression for $\overline{W_c}$, we introduce random variables A and W for the total number of client arrivals within this renewal interval, and the total number of frames scheduled for transmission by the server to satisfy these clients, respectively. For a mean request arrival rate of λ, the average number of clients is $E[A] = 1 + \lambda T$. Since $\overline{W_c} = E[W]/E[A]$, it remains to derive an expression for $E[W]$. Since simultaneous client requests are served as a batch, the transmission bandwidth requirement depends only on the likelihood that at least one request arrives in a single time interval. Let p be the probability that $X_j > 0$. Note that the probability distribution of client arrivals only influences the expression for $E[W]$ through the mean λ and the probability p. The analysis therefore applies

Parameter	Definition
B	Client buffer size (in units of frame times)
N	Length of the video (number of frames)
T	Threshold for starting a new transmission (in frame times)
p	Likelihood of at least one arrival in a single time unit
λ	Mean request arrival rate (requests per frame time)
$\overline{W_c}$	Average transmission per client (number of frames)
W	Total data transmitted by the server (number of frames)
A	Total request load satisfied by server (number of requests)

Fig. 3. **PBR Patching Model:** This table summarizes the parameters of the analytic model of PBR patching.

to a wide range of client arrival processes.

To compute $E[W]$, note that every renewal interval includes exactly one complete transmission of the N-frame video, and a partial transmission of $D(t)$ frames for every time slot t beyond the start of this complete transmission that has one or more client arrivals. Hence,

$$E[W] = N + p\sum_{t=1}^{T} D(t)$$

To simplify the analysis, we employ the approximate expression for $\tilde{D}(t)$. As in the expressions for $D(t)$ and $\tilde{D}(t)$, the derivation of $E[W]$ considers three cases:

- $T \le B$ or $B \ge N/2$: In this case, $D(t) = t$ for $t = 1, 2, \ldots, T$, resulting in

$$E[W] = N + p\frac{T(T+1)}{2}$$

  On average, after the first batch arrives at k_0, pT batches arrive (over the time interval $\{k_0+1, k_0+2, \ldots, k_0+T\}$) within the renewal interval, requiring the server to transmit an average of $(T+1)/2$ frames to each set of clients.

- $B < T \le N - B$ and $B < N/2$: In this case, $D(t) = t$ for $t = 1, 2, \ldots, B$, resulting in

$$\begin{aligned} E[W] &= N + p\frac{B(B+1)}{2} + p\sum_{t=B+1}^{T} D(t) \\ &\approx N + p\frac{B(B+1)}{2} + p\sum_{t=B+1}^{T}\left(N + B - \frac{NB}{t}\right) \\ &\approx N + p\frac{B(B+1)}{2} + \\ &\quad p\left\{(N+B)(T-B) - NB\ln\left(\frac{T}{B}\right)\right\} \end{aligned}$$

  The first two terms are for batch arrivals over time $\{k_0, k_0+1, \ldots, k_0+B\}$. The third term is for batch arrivals over time $\{k_0+B+1, k_0+B+2, \ldots, k_0+T\}$. The first approximation in this term stems from using the expression for $\tilde{D}(t)$. In the second approximation, the $\ln(T/B)$ term stems from $\sum_{t=B+1}^{T} 1/t \approx \int_{t=B}^{T} dt/t = \ln(T/B)$.

- $N - B < T < N$ and $B < N/2$: In this case, $D(t) = t$ for $t = 1, 2, \ldots, B$ and for $t = N - B + 1, \ldots, T$, resulting in

$$\begin{aligned} E[W] &= N + p\frac{B(B+1)}{2} + p\sum_{t=B+1}^{N-B} D(t) \\ &\quad + p\frac{(T-N+B)(T-N+B+1)}{2} \\ &\approx N + p\frac{B(B+1)}{2} + p\sum_{t=B+1}^{N-B}\left(N + B - \frac{NB}{t}\right) \\ &\quad + p\frac{(T-N+B)(N-B+T+1)}{2} \\ &\approx N + p\frac{B(B+1)}{2} + \\ &\quad p\left\{(N+B)(N-2B) - NB\ln\left(\frac{N-B}{B}\right)\right\} \\ &\quad + p\frac{(T-N+B)(N-B+T+1)}{2} \end{aligned}$$

  The first three terms are for batch arrivals over time $\{k_0, k_0+1, \ldots, k_0+N-B\}$. The fourth term is for batch arrivals over time $\{k_0+N-B+1, k_0+N-B+2, \ldots, k_0+T\}$. As before, the approximations stem from using $\tilde{D}(t)$ instead of $D(t)$, and replacing $\sum_{t=B+1}^{N-B} 1/t$ with $\int_{t=B}^{N-B} dt/t = \ln((N-B)/B)$.

Based on the expressions for $E[W]$, it is now possible to compute $\overline{W_c} = E[W]/(1+\lambda T)$. Then, the optimal threshold value can be computed, where

$$T_{opt} = \{T \mid \overline{W_c}(T) \le \overline{W_c}(j), j = 0, 1, \ldots, N-1\}.$$

For given values for B, N, λ, and p, the value of T_{opt} can be computed by differentiating the expression for $\overline{W_c}$ with respect to T and using numerical methods to determine when the derivative is zero. Alternatively, the minimum can be found by performing a binary search over the curve for $\overline{W_c}$ versus T. A patching server could compute T_{opt} offline, for different values of the arrival rate λ, and use the appropriate threshold online, based on the actual arrival rate.

The analysis of the patching algorithm in [2] assumes a continuous time model, Poisson arrivals, and no batching of clients. The RBR algorithm we consider in this paper is the discrete time equivalent of that algorithm, and like PBR, uses batching to serve multiple simultaneous clients. An approach similar to the analysis used for PBR can be used to analyze this RBR scheme for a range of client arrival processes.

D. Performance Comparison

As an initial comparison between PBR and RBR, we assume that clients arrive according to a Poisson process with rate λ, and all clients arriving within the same frame time are batched and served together. This results in $p = 1 - e^{-\lambda}$. First, in Figure 4, we investigate how the transmission bandwidth requirements of PBR vary as a function of T, using both the exact (using $D(t)$) and approximate (using $\tilde{D}(t)$) expressions for $\overline{W_c}$. The experiment considers a 1-hour long, 6 Mbps video stream with average request inter-arrival times of 5 and 10 minutes, and clients with 225 MBytes of buffer space (enough to store up to five minutes of the video). Note that the approximation for $\overline{W_c}$ is virtually indistinguishable from the exact expression for small and moderate values of T (including the regime where $\overline{W_c}$ is minimum), and is a moderately *pessimistic* estimate for very large T (the error is within $2 - 4\%$ of the exact value). This behavior is consistent across a range of buffer sizes, and arrival rates, suggesting that for practical purposes, it is sufficient to use the approximate expression for computing the optimal threshold.

Small values of T result in high bandwidth requirements, since most clients cannot take advantage of ongoing transmissions of the same video. As T increases, $\overline{W_c}$ decreases rapidly, as more clients have an opportunity to exploit their buffer space by patching to an earlier transmission. However, the finite client buffer size limits the benefit of patching beyond a certain point. When a client arrives long after the start of its MRC transmission, most of the video frames for the client must be retrieved directly from the server. Because of this, the transmission requirements eventually increase for larger threshold values, since a large number of clients retrieve almost all of their frames from the server. As a result, the curves have a cup-like shape, allowing a simple binary search procedure to locate

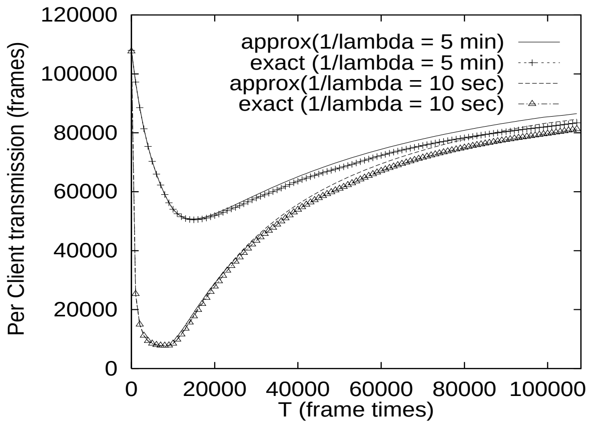


Fig. 4. **Transmission Bandwidth vs. Threshold T for PBR Patching:** The graph plots the exact (using $D(t)$) and approximate (using $\tilde{D}(t)$) expressions for $\overline{W_c}$ as a function of T for PBR patching. The experiment considers Poisson arrivals with $1/\lambda$ = 5 and 10 minutes. The video is one hour long, and the client buffer is 225 MBytes.

the optimal threshold value T_{opt}.

The optimal threshold value T_{opt} depends on the client buffer size, as shown in Figure 5(a). For small to moderate buffer sizes, T_{opt} is larger than B for both RBR and PBR patching. RBR patching has a smaller optimal threshold, since the algorithm only patches to the last B frames of the ongoing transmission when $t > B$. The larger threshold, along with the fact that a new client retrieves less frames from the server under PBR than under RBR, both contribute to the superior performance of PBR. This is shown in Figure 5(b), which plots the average transmission requirement per client $\overline{W_c}$ as a function of the client buffer size when each algorithm uses its respective optimal threshold. Small buffer sizes offer limited opportunities for patching, resulting in similar performance for PBR and RBR. As the client buffer size increases, $\overline{W_c}$ decreases more dramatically for PBR. For example, for a 45-MByte client buffer (which can store one minute of the video), PBR requires the server to transmit 242 MBytes less per client than RBR, a saving of 15%.

Eventually, increasing the client buffer size offers diminishing returns for both algorithms, and the performance difference between RBR and PBR starts to decrease, as shown in right side of Figure 5(b). A large buffer allows more a later client to receive more frames from its MRC transmission, thereby, pushing the optimal threshold higher. However, a higher threshold allows a longer time gap between a complete transmission and the clients that patch to it. This in turn requires the server to transmit a larger number of frames to clients which arrive a long time after the start of the complete transmission. In many cases, the server could reduce its overhead by starting a new complete transmission. As a result, the optimal threshold T_{opt} eventually reaches a maximum value, independent of

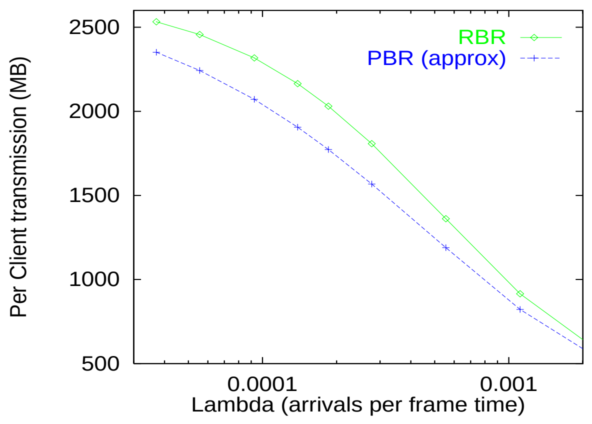


Fig. 6. **Influence of Client Arrival Rate on PBR and RBR Patching:** This graph plots the transmission bandwidth requirement $\overline{W_c}$ as function of mean arrival rate λ. The experiment considers a one-hour, 6-Mbps video, and a 45-MByte client buffer that can store up to 1-minute of the video.

the client buffer size, as shown in Figure 5(a). These results suggest that most of the benefits of patching can be achieved with a moderate amount of client buffer space, even for relatively long videos. In a practical patching system, the patching server could precompute buffer sizes for which $\overline{W_c}$ is within $x\%$ of the optimal, for different arrival rates.

The arrival rate has a significant influence on the performance of the patching algorithms. Figure 6 plots the average transmission $\overline{W_c}$ required to satisfy each client, as a function of the arrival rate λ. The experiment considers a 45-MByte client buffer that is capable of storing one minute of the one-hour, 6-Mbps video. Patching offers limited benefits for small arrival rates, since client requests do not typically arrive sufficiently close together in time. As λ increases, patching is increasingly effective, and $\overline{W_c}$ decreases. In this region, PBR requires substantially less bandwidth to serve each client. For example, $\overline{W_c}$ is generally about $10 - 14\%$ smaller under PBR than under RBR. In addition, when the number of client requests is larger, the reduction in the incremental overhead of serving each request is even more significant. The difference between PBR and RBR decreases for larger arrival rates, where many clients arrive close together in time, and can almost completely avoid retrieving frames directly from the server. In this region, both algorithms have an optimal threshold that is smaller than the client buffer size B, resulting in very similar performance.

IV. Optimal Patching

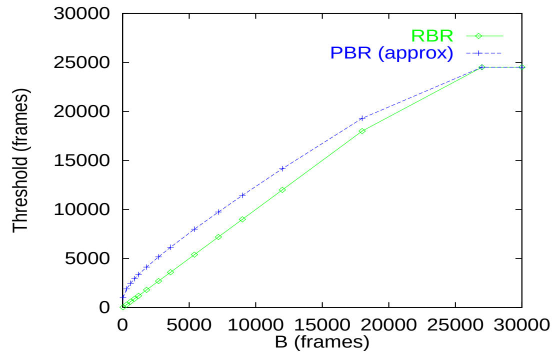
The effectiveness of RBR and PBR patching depends on selecting an appropriate threshold T, based on the request arrival process and the system parameters (buffer size B and video duration N). However, the arrival rate is not always known in advance, and may fluctuate across time. In this section, we present an optimal patching algorithm, *Greedy Buffer Reuse* (GBR), that does not depend on knowledge of the client arrival process, and does not use thresholding. The algorithm allows a client to subscribe to *multiple* ongoing transmissions to maximize the benefit of patching. The algorithm is incremental in that it decides the schedule for each client in the order of client arrival time. Therefore, it does not assume any future arrivals or arrival patterns. The strategy is greedy in that each client always fetches from on-going multicast channels as long as the client buffer does not overflow. Therefore, it reduces the total number of frames each client has to fetch from the server directly, given the client buffer size.

This GBR algorithm provides a lower bound on the transmission bandwidth required to serve client requests, and can serve as the basis of new patching algorithms that have lower computational complexity. After presenting the algorithm and establishing the optimality properties, we compare the performance with PBR.

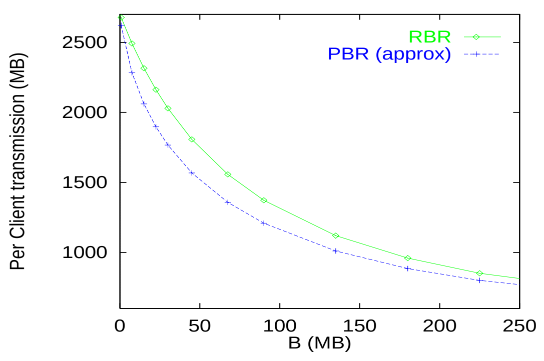
A. GBR Algorithm

The GBR algorithm sequences through the frames of the video and schedules the client to receive a frame *as late as possible* – from an ongoing transmission (if possible), or directly from the server. A frame is sent directly by the server in two cases – when no ongoing transmission includes this frame, and when the client does not have enough buffer space to store the frame from an earlier transmission. Figure 7 presents the pseudocode for the algorithm, which computes a reception schedule $\mathbf{RS}_i$ and a channel schedule $\mathbf{CS}_i$ for client i, arriving at time t_i. As discussed in Section II-C, the channel schedule is a vector, where $CS_{i,j}$, $j = 1, 2, \ldots, N$, is either 0 (idle slot) or a frame number k, indicating that the server should transmit frame k on channel C_i at time $d_j = t_i + j$. The reception schedule is a vector that specifies the time and the channel from which the client should receive each video frame. Entry $RS_{i,j}$, $j = 1, 2, \ldots, N$, is a set of pairs (k, l), indicating that frame k should be retrieved from channel C_l at time $t_i + j$.

The algorithm schedules frame reads from existing channels into the client buffer *as late as possible*. The vectors $\mathbf{LT}$ and $\mathbf{LC}$ respectively keep track of the latest time and the channel on which a particular frame will be sent. LT_j is the latest time that frame j is transmitted, considering all of the existing channels; LC_j indicates which channel is responsible for this transmission. The channel transmission schedule, latest arrival time, and latest arrival channel vectors are all maintained at the patching server. To receive the different frames at the correct times, a client should be aware of its own reception schedule. The patching server might transmit the computed reception schedule to the client. Alternatively, it might transmit



(a) Optimal threshold T_{opt}



(b) Server bandwidth $\overline{W_c}$

Fig. 5. **Influence of Client Buffer Size on PBR and RBR Patching:** The left graph shows how the optimal threshold T_{opt} depends on the client buffer size B for both PBR and RBR patching. Then, using the optimal threshold T_{opt}, the right graph considers the influence of B on the transmission bandwidth requirement $\overline{W_c}$. Both experiments consider a one-hour, 6-Mbps stream with an average request inter-arrival time of $1/\lambda = 2$ minutes.

```
 GBR_schedule ($B_i, t_i$, LT, LC)
for $j = 1, \ldots, N$ (consider frame $j$)
  if ($LT_j \geq t_i$) and (client $i$ can store an additional frame for times $LT_j, \ldots, d_j - 1$))
      (i) Add frame $j$ to the reception schedule (i.e., add tuple $(j, LC_j)$ to $RS_{i,LT_j-t_i}$).
      (ii) Update buffer occupancy to store an additional frame for times $LT_j, \ldots, d_j - 1$.
      (iii) Do not schedule the frame for transmission on channel $C_i$ (i.e., $CS_{i,j} = 0$).
  else
      (i) Schedule frame $j$ for transmission on channel $C_i$ at time $d_j$ (i.e., $CS_{i,j} = j$).
      (ii) Add frame $j$ to the reception schedule (i.e., add tuple $(j, C_i)$ to $RS_{i,d_j-t_i}$).
```

Fig. 7. **GBR Patching Algorithm:** This figure presents the pseudocode for processing the request for client i (client buffer size B_i), arriving at time i. The greedy algorithm produces a channel schedule $\mathbf{CS}_i$ for the server, and a reception schedule $\mathbf{RS}_i$ for the client.

sufficient information (in this case, the latest arrival time and latest arrival channel vectors) which could then be used by the client to compute its reception schedule.

The key intuition behind the algorithm is that frames that are received by the client closer to their consumption deadlines occupy the workahead buffer for a shorter time and, hence, maximize the chances of other frames using the same space. If frame j is already scheduled for transmission by the server between time t_i and $t_i + j$, the client tries to receive the copy that is transmitted last in that interval. If receiving this frame would overflow the client buffer, the server schedules a new transmission of frame j at time $t_i + j$, on channel C_i. We refer to the resulting schedule as the greedy reception schedule for client i. The computational complexity of the algorithm is $O(BN)$ per client batch, for client buffer size B.

B. Optimality

The GBR algorithm in Figure 7 is an optimal patching algorithm, in that it minimizes the transmission bandwidth required to satisfy a collection of clients. Consider M client arrivals to the server, where our greedy algorithm computes a channel and reception schedule for client i at its arrival time t_i, based on the existing channel schedules. Let the greedy schedule S at the server determine the set of reception and channel schedules of all the clients, in the order of client arrival times.

Theorem 1: Given a client buffer size B, the greedy schedule is optimal in the sense that no other schedule requires fewer frames from the server.

The proof of the theorem follows from proving the following claim.

Claim: *We can convert any valid schedule S to the greedy schedule via a series of transformations, each of which does not increase the number of frames that the server delivers.*

Proof: We work in the order of client arrival times. For each client, find the first frame j that is received in a non-greedy fashion. We will transform the schedule without increasing the number of frames that the server delivers so that the client receives frame j in a greedy fashion:

- Case 1: *Frame j is multicast between t_i and $t_i + j$ and is received before the last multicast of the frame in schedule S.*

  In this case, we can transform schedule S so that frame j is received at the last multicast of that frame. The resulting schedule S' will not require more frames from the server and the client will not overflow its buffer since the frame is received later.

- Case 2: *Frame j is scheduled to be multicast between t_i and $t_i + j$ and the client buffer will not overflow if frame j is received at the last scheduled multicast of the frame. However, in schedule S, frame j is transmitted afresh to this client at time $t_i + j$.*
  We can construct another schedule S' as follows. Schedule S' lets the client receive frame j at the last multicast of that frame. If there is another client that receives frame j at time $t_i + j$, we multicast frame j at the time that the earliest arriving such client needs to playback frame j. Also, other such clients retrieve frame j at this time. If no such client exists, the rest of schedule remains the same. Schedule S' is still valid, since each client still receives the frame before its playback time. Moreover, no buffer will overflow, since we only let some clients receive data later. Finally, note that schedule S' does not transmit more frames than schedule S.

Note that a consequence of Theorem 1 is that the GBR patching algorithm is also locally *optimal*. That is, this algorithm results in the server transmitting the minimum number of frames afresh to any new client.

C. Performance Evaluation

In Figure 8(a), we compare the optimal patching algorithm (GBR) against PBR patching using the optimal PBR threshold, as a function of the client arrival rate λ. The experiment considers a one-hour, 6-Mbps video stream. Each point in the graph is the average over six independent simulation experiments, each consisting of 720 distinct batch (of one or more clients) arrivals. The individual client arrivals are generated by a Poisson process, and multiple clients arriving within the same time unit (frame time) are batched and served together.

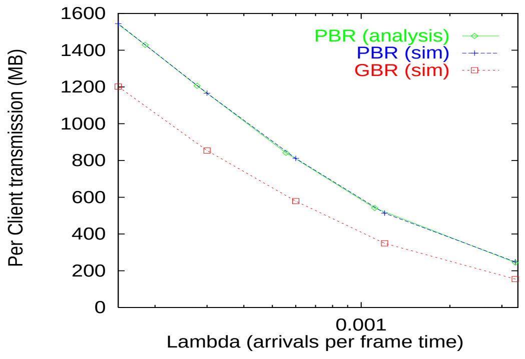
For a high arrival rate with $1/\lambda = 10$ seconds, the server overhead per client for PBR is 60% higher than that for GBR, a difference of 93 MBytes. As λ decreases, clients arrive further apart from each other, reducing the effectiveness of patching, resulting in higher values of $\overline{W_c}$ for both algorithms. For $1/\lambda = 2$ minutes, PBR results in a value for $\overline{W_c}$ that is 36% higher than GBR; the corresponding difference between PBR and GBR is 312 MBytes. These trends indicate that there is substantial potential for improving the performance of patching beyond that of the algorithms in literature.

Although the optimal patching algorithm does not explicitly limit the number of simultaneous channels for each client, the results in Figure 8(b) show that a client rarely listens to more than three channels at a time. The graph plots the proportion of time that a client listens to less than or equal to x channels, across several values of λ. Small values of λ offer limited opportunities for patching, and the client spends most of its time listening to just one channel. For example, when $1/\lambda = 3.5$ minutes, the client spends 17% of the time idle, 67% of the time listening to one channel, and about 1% of the time listening to three or more channels. For a high arrival rate, clients receive more frames from existing transmissions. However, even for small client inter-arrival times, the number of simultaneous channels is still quite low. For example, for $1/\lambda = 30$ seconds, 40%, 37%, and 14% of the total time is spent idle, listening to one and two channels respectively. Less than 1% of the time is spent listening to more than 5 channels. This indicates that most of the savings in transmission can be accrued if clients are able to listen to three or four channels simultaneously. The results also suggest that introducing an explicit constraint on the number of simultaneous channels (i.e., number of tuples in $RS_{i,j}$) should not degrade performance substantially.

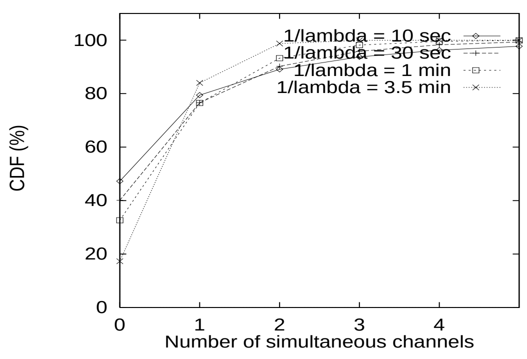
V. Ongoing Work

As ongoing work, we are pursuing a number of extensions to PBR and optimal GBR patching:

- **Performance evaluation of PBR:** Extending the work in Section III, we are comparing PBR and RBR patching under different client arrival processes. These experiments involve varying the values of p and λ in the analytic model, allowing us to study the impact of bursty arrivals on the relative performance of the two threshold-based patching schemes. In addition, we are extending the analysis to support batching of client requests that arrive within τ time units of each other. Finally, we are considering more flexible thresholding policies that incorporate information about the *number* of client requests that have arrived since the start of the previous complete transmission, to allow PBR's threshold value to adjust to changes in the arrival process across time.
- **Computational complexity of GBR algorithm:** The GBR algorithm as presented in Section IV has a complexity of $O(BN)$ per client batch. We are investigating another algorithm for computing the *same* channel and reception schedules that reduces the influence of the client buffer size on the algorithmic complexity. To further reduce the computational overhead, we are evaluating a generalization of GBR that divides a video into consecutive g-frame chunks, and applies the same scheduling decision to all frames within a chunk. This effectively reduces the client buffer size to B/g and the video length to N/g, and also reduces the number of client batches that must be considered, at the expense of restricting the opportunities for patching. Still, the technique should be effective for a range of small g values.



(a) Server bandwidth $\overline{W_c}$



(b) CDF

Fig. 8. **Performance of GBR Patching:** The left graph plots $\overline{W_c}$ as a function of λ for the PBR and GBR algorithms. The right graph plots the proportion of time that a client subscribes to less than or equal to x channels, under GBR patching. The client has a 90-MByte buffer that can store two minutes of the video.

- **Limited client I/O bandwidth under GBR:** The *last mile* network bandwidth constraints, or disk or memory bandwidth constraints, may keep the client from retrieving data from more than K simultaneous transmission channels. The results in Figure 8(b) show that GBR rarely requires more than three simultaneous transmissions to the same client. To avoid violating any constraints on K, we consider a simple heuristic extension to the optimal GBR algorithm. If retrieving a frame from an ongoing transmission would exceed the client I/O bandwidth, then a separate copy of the frame is scheduled for transmission from the server at the frame's playback time. Repeating the process across the sequence of N playback times removes any violations of the constraint K.
- **Dynamic join/leave from multicast groups:** If the patching server is not connected to the set of clients via a shared medium, the overhead of joining and leaving multicast groups would limit the client's ability to switch between transmission channels at the frame level. Dividing the video stream into g-frame chunks, as discussed above, would impose a minimum time between joining and leaving multicast groups. As an alternate approach, the client could subscribe to a set of K transmission channels for the duration of the transfer, and apply local filtering to store only the necessary video frames. This approach avoids dynamic joining and leaving of multicast groups, at the expense of consuming additional network bandwidth to transfer frames that the client does not need (or the use of network proxies to perform the filtering). We are investigating variants of GBR that generate schedules that apply a *single* set of K transmission channels to each client.
- **Blocking service model:** The model in Section II assumes that the server and the network have sufficient resources to satisfy all client requests. This approach is a reasonable way to compare the various patching algorithms, and to determine the amount of resources required for a well-provisioned service. However, in a real system, the patching service should be able to block a client request, when necessary. The simplest approach applies the existing patching algorithms to compute the reception and channel transmission schedules for the new client, and rejects the request if the server or the network cannot satisfy the resource requirements of the schedule. Alternatively, the patching server could compute schedules that reduce the likelihood of blocking future client requests. Extensions of the GBR patching algorithm can exploit any available latitude in scheduling frame transmissions for the new client. For example, the server could transmit a frame early (subject to the client buffer constraint) to reduce the total number of frames that must be sent in any given time slot.

VI. Conclusions

The high transmission bandwidth requirements of streaming video makes it a challenging problem to provision network resources for delivering such media to remote clients. In this paper, we examined *patching*, a technique for reducing the transmission to a client, by allowing sharing of data from existing, ongoing transmissions of the same video. We introduced *Periodic Buffer Reuse* (PBR), a new patching scheme that maximizes the amount of data that a client can retrieve from the most recently initiated complete transmission. Similar to existing patching schemes, PBR employs a threshold to determine when to start a new complete transmission of the stream. We analytically derived a closed-form expression for the transmission bandwidth requirements for PBR patching, and showed how to determine the optimal threshold value. Our performance comparison showed that PBR can significantly outperform existing patching schemes. Then, we presented *Greedy Buffer Reuse* (GBR), a patching algorithm that provably minimizes the server transmission bandwidth requirements. Simulation experiments showed that GBR patching offers a sizeable reduction in transmission overhead over any of the threshold-based schemes, and rarely requires the client to listen to more than three simultaneous transmissions. This algorithm can serve as the basis of new patching algorithms that have lower computational complexity, and simpler implementation.

VII. Acknowledgements

The authors thank Jim Kurose for many helpful and insightful discussions and suggestions which were crucial to this work.

References

[1] K. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, September 1998.

[2] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE International Conference on Multimedia Computing and Systems*, 1999.

[3] Y. Cai, K. Hua, and K. Vu, "Optimizing patching performance," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1999.

[4] C. Aggarwal, J. Wolf, and P. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proc. IEEE International Conference on Multimedia Computing and Systems*, June 1996.

[5] L. Golubchik, J. Lui, and R. Muntz, "Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers," *ACM Multimedia Systems Journal*, vol. 4, no. 3, 1996.

[6] K. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proc. ACM SIGCOMM*, September 1997.

[7] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proc. Inter. Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

[8] D. Eager and M. Vernon, "Dynamic skyscraper broadcasts for video-on-demand," in *Proc. Inter. Workshop on Advances in Multimedia Information Systems, Lecture Notes in Computer Science 1508*, September 1998.

[9] D. Eager, M. Ferris, and M. Vernon, "Optimized regional caching for on-demand data delivery," in *Proc. Multimedia Computing and Networking*, January 1999.

[10] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP), request for comments 2326," April 1998. `ftp://ftp.isi.edu/in-notes/rfc2326.txt`.