Cost-Based Scheduling for Scalable Media Streaming

Bashar Qudah and Nabil J. Sarhan
The Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI, 48202 USA
e-mail: {bqudah,nabil}@wayne.edu

Abstract

The number of media streams that can be supported concurrently is highly constrained by the stringent requirements of real-time playback and high transfer rates. To address this problem, media delivery techniques, such as Batching and Stream Merging, utilize the multicast facility to increase resource sharing. The achieved resource sharing depends greatly on how the waiting requests are scheduled for service. Scheduling has been studied extensively when Batching is applied, but up to our knowledge, it has not been investigated in the context of stream merging techniques, which achieve much better resource sharing. In this study, we analyze scheduling when stream merging is employed and propose a simple, yet highly effective scheduling policy, called *Minimum Cost First* (MCF). MCF exploits the wide variation in stream lengths by favoring the requests that require the least cost. We present two alternative implementations of MCF: *MCF-T* and *MCF-P*. We compare various scheduling policies through extensive simulation and show that MCF achieves significant performance benefits in terms of *both* the number of requests that can be serviced concurrently and the average waiting time for service.

Keywords: Scheduling, stream merging, video-on-demand (VOD), video streaming.

0.1 Introduction

Recently, the interest in media streaming has grown dramatically over the Internet, cellular networks, and Cable TV. Currently, the main applications over the web include Live Webcasting, Web Conferencing, Video-on-Demand (VOD), Distance Learning, Employee Training, Collaborations, Product Announcements, Surveillance/Security, and many more. Unfortunately, the distribution of streaming media faces a significant scalability challenge due to the high server and network requirements. Hence, numerous techniques have been proposed to deal with this challenge, especially in the areas of *media delivery* (also called *resource sharing*) and *request scheduling*. This study focuses on video streaming of recorded/stored content. Figure 1 shows a simplified functional description of a video streaming server and its connectivities to a storage subsystem and clients.

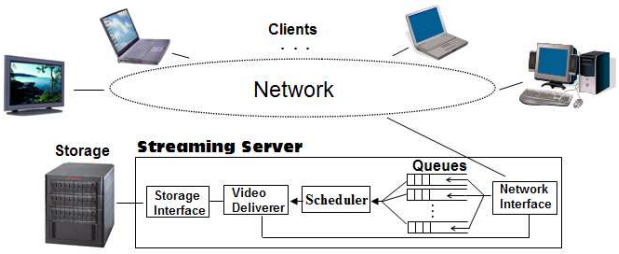


Figure 1: Simplified Video Streaming Environment

Video delivery can be done in a *client-pull* or a *server-push* fashion, depending on whether the channels are allocated on demand or reserved in advance. The simplest client-pull technique is *unicast*, whereby each request is serviced using a full stream. The cost of each request with this technique is very expensive in terms of both server and network resources. *Batching* [10, 9, 26, 2] increases resource sharing by accumulating the requests for the same videos and servicing them together using multicast streams. Stream merging techniques [15, 5, 11, 23, 19, 22] reduce the cost further by aggregating clients into larger groups that share the same multicast streams. These techniques include *Stream-Tapping/Patching* [8, 15, 6], *Transition Patching*

[5, 7], and *Earliest Reachable Merge Target* (ERMT) [11, 13, 12]. Server-push techniques (also called *Periodic Broadcasting* techniques) [16, 18, 20, 4, 17] divide each video into multiple segments and broadcast each segment periodically on dedicated server channels. Thus, they can be used only for the most popular videos and require the clients to wait until the next broadcast time of the first segment. This paper considers the stream merging approach.

The degrees of resource sharing achieved by video delivery techniques depend greatly on how the waiting requests are scheduled for service. Scheduling has been studied extensively with Batching [10, 26, 2, 25, 24], and the main policies include *First Come First Serve* (FCFS) [10], *Maximum Queue Length* (MQL) [10], and *Maximum Factored Queue Length* (MFQL) [2]. Despite the many proposed stream merging techniques and the numerous possible variations, the issue of scheduling has not been investigated in the context of these scalable techniques. Our analysis here stresses that the choice of a scheduling policy can be as important as or even more important than the choice of a stream merging technique, especially when the server is loaded. Prior work on stream merging [15, 5, 11] employed the scheduling policies proposed for Batching, and the decisions as to what policy to choose among the many available candidates were based on the performance results in the Batching environment. For example, [15] and [5] used MFQL, since it performs well with Batching. The Batching and stream merging requirements, however, differ significantly. With Batching, all streams are equal in length to the duration of the video, whereas stream merging techniques lead to streams that vary dramatically in length. Hence, the results in the case of Batching cannot be generalized for stream merging.

This paper investigates scheduling in servers employing stream merging techniques and proposes a simple, yet highly efficient scheduling policy, called *Minimum Cost First* (MCF). This policy captures the varying stream lengths with stream merging techniques by selecting the requests requiring the least cost in terms of the required stream length. We propose two alternative implementations of MCF: *MCF-T* and *MCF-P*. MCF-T selects the requests in the video queue that requires the minimum total cost (i.e., for all requests), whereas MCF-P selects the requests in the video queue with the least cost per request. For Patching and Transition Patching, we present three MCF-P variants based on how the cost is determined for regular and transition streams.

We analyze, through extensive simulation, various scheduling policies in servers employing stream merging and demonstrate and compare the effec-

tiveness of the proposed cost-based schemes. The study considers three main stream merging techniques, offering different levels of performance and implementation complexity: Patching, Transition Patching, and ERMT. The analyzed performance metrics include the overall customer defection (turnaway) probability, the average request waiting time, and unfairness against unpopular videos. The defection probability is the most important metric because it translates to the number of customers serviced concurrently and to server throughput. The average waiting time comes next in importance. Moreover, we evaluate the impacts of server capacity, customer waiting tolerance, arrival rate, number of videos, and video lengths on the results. Furthermore, we compare various stream merging techniques when applied with the proposed schemes and examine a dynamic implementation to find key design parameters in Patching and Transition Patching.

The main simulation results can be summarized as follows.

- The results with stream merging differ significantly from prior results in the case of Batching. For example, MFQL performs poorly compared with MQL and FCFS, when stream merging is used.
- The proposed MCF policy provides outstanding performance benefits with all studied stream merging techniques. Interestingly, it performs significantly better than existing policies in terms of *both* the defection probability and the average request waiting time (the two most important performance metrics). Moreover, it performs better in unfairness within the most important regions of operation than the best performer overall among existing policies.
- The performance benefits of MCF increase with more scalable stream merging (i.e., the improvements are better with ERMT than Transition Patching and with Transition Patching than Patching).
- The implementation MCF-P offers the best overall performance although it performs somewhat worse than MCF-T in unfairness, which is far less important than the other two metrics.
- The improvements achieved by MCF-P compared with the best performer among existing policies can be up to 50% in *all* three metrics, simultaneously. This demonstrates its remarkable performance benefits.

The rest of the paper is organized as follows. Section 0.2 discusses and provides preliminary analysis of stream merging and scheduling policies. Then, Section 0.3 presents the proposed scheduling policies and Section

0.4 discusses the simulation platform, the workload characteristics, and the main performance evaluation results. Finally, conclusions are drawn in the last section.

0.2 Background Information and Preliminary Analysis

0.2.1 Stream Merging

Stream merging techniques aggregate clients into larger groups that share the same multicast streams. In this subsection, we discuss three main stream merging techniques: Patching, Transition Patching, and ERMT.

With Patching, a new request joins immediately the latest multicast stream for the object and receives the missing portion as a *patch* using a unicast stream. When the playback of the patch is completed, the client continues the playback of the remaining portion using the data received from the multicast stream and already buffered locally. Since patch streams are not sharable with later requests and their cost increases with the temporal skew to the latest multicast stream, it is more cost-effective to start new full multicast stream (also called *regular stream*) after some time. Thus, when the patch stream length exceeds a certain value called *regular window* or Wr, a new regular stream is initiated instead. Figure 2 further explains the concept. D is the video length in seconds. Typically, Wr is much smaller than D, but the figure is meant only for a simple clarification. Initially, one regular stream is delivered, followed by three patch streams to service new requests. Then, another regular stream starts, and the cycle is repeated.

Transition Patching allows some patch streams to be sharable by extending their lengths. It introduces another multicast stream, called *transition patch*. The threshold to start a regular stream is Wr as in Patching, and the threshold to start a transition patch is called the *transition window* (Wt). The transition patch length is equal to the difference between the starting times of the patch stream and the last regular stream plus $2Wt$, whereas the length of the patch stream is equal to the difference between the starting times of the patch stream and the latest transition stream or regular stream. Therefore, the maximum possible patch length is Wt, and the maximum possible transition patch length is $Wr + 2Wt$. Figure 3 further illustrates the concept. A possible scenario for a client is to start listening to its own

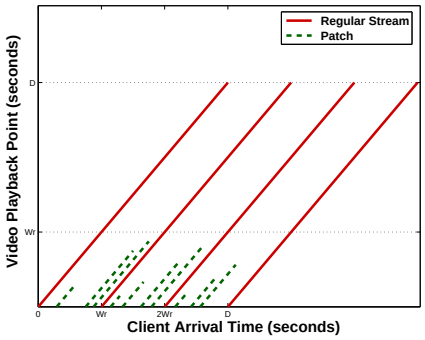


Figure 2: Patching

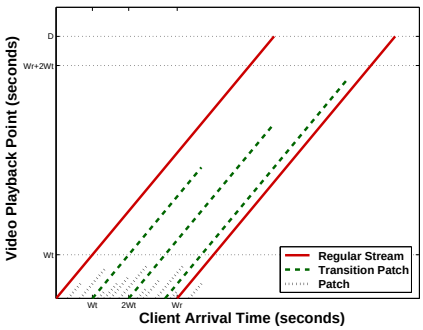


Figure 3: Transition Patching

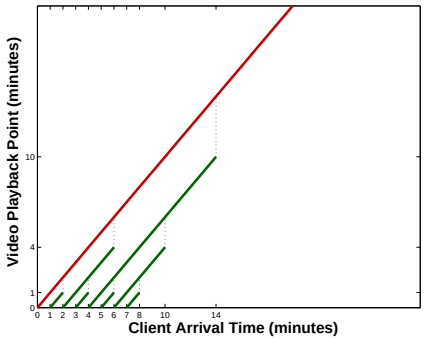


Figure 4: ERMT

patch stream and the transition patch, and when its patch is completed, it starts listening to the regular stream.

ERMT is a near optimal hierarchical stream merging technique. Basically, a new client or a newly merged group of clients snoops on the closest stream that it can merge with if no later arrivals preemptively catch them [11]. The target stream can later get extended to satisfy the needs of the new client (only after the merging stream finishes and merges with the target), and this extension can affect its own merge target. For example, in Figure 4, the third stream length got extended from two minutes to four minutes after the fourth stream had merged with it. ERMT performs better than other hierarchical stream merging alternatives and close to the optimal solution, which assumes that all request arrival times are known in advance [11, 13, 3]. It is used by *Bandwidth Skimming* [12] to work when the client download bandwidth is less than double the playback rate.

These three stream merging techniques differ in complexity and performance. Patching is the simplest to implement since it allows only one merge during the client's service period and allows only regular streams to be shared. Hence, it enables the client to know the streams it will listen to in advance. Transition Patching also informs the client about all the streams to listen to in advance, but it allows up to two merges per client. ERMT is the most complex because it allows any number of merges that can help in maximizing resource sharing. The client needs to be continuously informed about all previous streams for the same video, and the client (or the server) needs to perform frequent calculations to decide on the next merge target when a merge occurs. Consequently, selecting the most appropriate stream merging technique depends on a tradeoff between the required implementation complexity and the achieved performance. Both the implementation complexity and performance increase from Patching to Transition Patching to ERMT.

0.2.2 Request Scheduling

To facilitate scheduling, the server maintains a waiting queue for every video, routes incoming requests to their corresponding queues, and applies a scheduling policy to select an appropriate queue for service whenever it has an available *channel*. A channel is a set of resources (network bandwidth, disk I/O bandwidth, etc.) needed to deliver a multimedia stream. All requests in the selected queue can be serviced using only one channel. The number of channels is referred to as *server capacity*.

All scheduling policies are guided by one or more of the following primary objectives.

1. Minimize the overall customer defection (turn-away) probability.
2. Minimize the average request waiting time.
3. Minimize unfairness.

The defection probability is the probability that a new customer leaves the server without being serviced because of a waiting time exceeding the user's tolerance. It is the most important metric because it translates directly to the number of customers that can be serviced concurrently and to server throughput. The second and the third objectives are indicators of customer-perceived quality of service (QoS). It is usually desirable that the servers treat equally the requests for all videos. Unfairness measures the bias of a policy

against cold (i.e., unpopular) videos and can be obtained by the following equation: $unfairness = \sqrt{\sum_{i=1}^{N_v}(r_i - \bar{r})^2/(N_v - 1)}$, where r_i is the reneging probability for the waiting queue i, $\bar{r}$ is the mean reneging probability across all waiting queues, and N_v is the number of waiting queues (and number of videos as well).

Let us now discuss the main scheduling policies proposed for servers employing Batching.

- *First Come First Serve* (FCFS) [10] - This policy acts fairly by selecting the queue with the oldest request.
- *Maximum Queue Length* (MQL) [10] - This policy maximizes the number of request that can be serviced at any time by selecting the queue with the largest number of requests.
- *Maximum Factored Queue Length* (MFQL) [2] - This policy attempts to minimize the mean request waiting time by selecting the queue with the *largest factored length*. The factored length of a queue is defined as its length divided by the square root of the relative access frequency of its corresponding video. MFQL reduces the average waiting time optimally only if the server is fully loaded and customers always wait until they receive service (i.e., no defections).
- *FCFS-n* [10] - With this policy, the server broadcasts periodically the n most common videos on dedicated channels and schedules the requests for the other videos on a FCFS basis. When no request is waiting for the playback of any one of the n most common videos, the server uses the corresponding dedicated channel for the playback of one of the other videos.
- *Group-Guaranteed Server Capacity* (GGSC) [26] - This policy preassigns server channel capacity to groups of requests in order to optimize the mean request waiting time. It groups objects that have nearly equal expected batch sizes and schedules requests in each group on a FCFS basis on the collective channels assigned to each group.

This study considers only FCFS, MQL, and MFQL because they can directly be used in the stream merging environment. GGSC and FCFS-n are not directly applicable. Moreover, GGSC does not perform as well as FCFS in high-end servers [26] and FCFS-n performs either as well as or worse in certain situations than FCFS [26].

0.3 Proposed Minimum Cost Scheduling

In this paper, we analyze scheduling in servers applying stream merging techniques and propose a simple, yet highly efficient scheduling policy, called *Minimum Cost First* (MCF). The idea of MCF is to exploit the variations in stream lengths caused by stream merging techniques and give preference to the requests in the queue requiring the least cost. The length of the stream (in time) is directly proportional to the cost of servicing that stream since the server allocates a channel for the entire time the stream is active. By performing scheduling based on the cost, the server is expected to support a larger number of concurrent customers. Moreover, it can reduce the average request waiting time because server channels become available sooner for servicing new requests. Furthermore, MCF-based scheduling is easy to implement and incurs only little implementation overhead.

The cost can be computed in different ways. We propose the following two alternative implementations of MCF.

- *MCF-T* (T for "Total") - This implementation selects the queue requiring the least total cost for servicing all its requests. Thus, MCF-T aims at reducing the following objective function: $F(i) = L_i \times R_i$, where L_i is the required stream length for the requests in queue i, and R_i is the (average) data rate for the requested video.
- *MCF-P* (P for "Per") - This implementation selects the queue with the least cost per request. The objective function is $F(i) = \frac{L_i \times R_i}{N_i}$, where N_i is the number of waiting requests in queue i. By incorporating the MQL objective, MCF-P has the advantage of combining the benefits of MCF-T and MQL.

In certain situations, it may be desirable to modify the objective functions slightly because they introduce other types of unfairness. In particular, with these functions, MCF-T and MCF-P can be biased against long videos as well as high-quality videos. To remove this possible shortcoming, the objective functions can be modified respectively as follows: $F(i) = \frac{L_i}{D_i}$ and $F(i) = \frac{L_i}{D_i \times N_i}$, where D_i is the length of video i. In this study, we do not seek to exploit the variations in video lengths or video data rates and thus these modified objectives are used.

MCF works with any stream merging technique. In the case of Patching, MCF favors patches over regular streams and shorter patches over longer ones. With Transition Patching, MCF also tends to favor patches over tran-

sition patches (but this is not always the case). ERMT does not have the concepts of regular streams and transition patches. With ERMT, some streams may be extended but these extensions cannot be considered by scheduling since they happen after the scheduling decisions have been made. When the server is loaded, regardless of which delivery technique is used, MCF delays the services of long streams more than short ones and thus long streams tend to benefit more from the batching effect.

In Patching and Transition Patching, regular streams and transition patches are longer than patch streams but they are shared by all subsequent requests in Wr or Wt duration, respectively. The question arises as to whether they can benefit from special treatment so as not to be delayed excessively. Therefore, we present three variants of MCF-P, which differ in how regular streams and transition streams are treated.

- *RAF (Regular as Full)* - This variant simply uses the full length of regular streams and transition patches in determining the cost. Thus, for a regular stream j for video i, $F_j(i) = \frac{D_i}{D_i \times N_i} = \frac{1}{N_i}$. For a transition stream j for video i, $F_j(i) = \frac{2 \times Wt_i + s_j}{D_i \times N_i}$, where s_j is the skewness of the transition patch from the latest regular stream, and Wt_i is the transition window (discussed in Subsection 0.2.1) for video i.
- *RAP (Regular as Patches)* - This variant computes the lengths of regular streams and transition patches as if they were patches. Thus, for a regular stream or transition patch j for video i, $F_j(i) = \frac{s_j}{D_i \times N_i}$, where s_j is the skewness from the latest regular stream or transition patch.
- *RAS (Regular as Shared)* - This variant amortizes the cost of regular streams and transition patches considering that all subsequently serviced requests until the next regular stream or transition patch utilize them. Hence, for a regular stream j for video i, $F_j(i) = \frac{D_i}{D_i \times Nr_i} = \frac{1}{Nr_i}$ and for a transition stream j for video i, $F_j(i) = \frac{2 \times Wt_i + s_j}{D_i \times Nt_i}$, where Nr_i and Nt_i are the number of subsequently serviced requests until the next regular stream or transition patch, respectively.

In RAS, Nr_i and Nt_i can be found dynamically through smoothing the results of recent history. They can also be approximated by the expected number of arriving requests within Wr or Wt duration, respectively. Thus, $Nr_i = Wr_i \times \lambda_i$ and $Nt_i = Wt_i \times \lambda_i$, where λ_i is the request arrival rate for video i. By substituting these values in the objective functions and using the optimal values of Wr and Wt derived in [14] and [21], it can be shown that

for regular stream j for video i in Patching, $F_j(i) \approx Wr_i/2$. With Transition Patching, for regular stream j for video i, $F_j(i) \approx Wt_i/2$, and for transition patch j, $F_j(i) \approx (2 + k_i)/\lambda_i$, where k_j is the number of transition patches, including transition patch j, since the latest regular stream.

The three MCF-P variants have close implementation complexities, which are also comparable to MCF-T, assuming that RAS is implemented using the derived equations. Otherwise, RAS incurs a little more overhead in updating Nr_i and Nt_i regularly.

0.4 Performance Evaluation

0.4.1 Simulation Platform

We have developed a simulator for a media server that supports various video delivery techniques and scheduling policies. We have validated the simulator by reproducing several graphs in previous studies. The simulation stops after a steady state analysis with 95% confidence interval is reached.

0.4.2 Workload Characteristics

Table 1 summarizes the workload characteristics used. Like most prior studies, we assume that the arrival of the requests to the server follows a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. We also assume, as in previous work, that the access to videos is highly localized and follows a Zipf-like distribution. With this distribution, the probability of choosing the n^{th} most popular video is $C/n^{1-\theta}$ with a parameter θ and a normalized constant C. The parameter θ controls the skewness of video access. Note that the skewness reaches its peak when $\theta = 0$, and that the access becomes uniformly distributed when $\theta = 1$. In accordance with prior studies, we assume that $\theta = 0.271$. We characterize the waiting tolerance of customers by an exponential distribution with a mean of 0.5, 1.5, or 2.5 minutes.

We generally study a server with 120 videos, each of which is 120-minute long. We examine the server at different loads by fixing the request arrival rate at 40 requests per minute and varying the number of channels (server capacity) generally from 350 to 800. We also study the impacts of arrival rate, number of videos, and video length. MCF is expected to perform better

in workloads containing videos of varying lengths (time durations) and data bit rates (if the original unmodified objective functions are used). In this paper, we do not seek to exploit these variations because they lead to bias against long videos and videos with high quality encodings. Interactive operations can be supported using contingency channels [1]. Thus, the relative performance of various scheduling policies in terms of defection probability, waiting times, and unfairness does not depend on these operations as long as the fraction of server channels used for these operations is kept the same. To isolate the impact of these operations, the reported server capacity in this study includes only the number regular (i.e., non-contingency) channels.

Table 1: Summary of Workload Characteristics

Parameter	**Model/Value(s)**
Request Arrival	Poisson Process
Request Arrival Rate	Variable, Default = 40 Requests/minute
Server Capacity	350 to 800 channels
Video Access	Zipf-Like with Skewness Parameter $\theta = 0.271$
Number of Videos	Variable, Default = 120
Video Length	Variable, Default = 120 minutes
Waiting Tolerance Model	Exponential Distribution
Mean Waiting Tolerance (μ_{tol})	0.5, 1.5, and 2.5 minutes

0.4.3 Result Presentation and Analysis

General Analysis

Let us start with the results in the case of ERMT, the most efficient and most complex of the considered stream merging techniques. Figure 5 compares various scheduling policies in terms of customer defection probability (which translates to the number of customers that can be serviced concurrently), waiting times, and unfairness when the waiting tolerance of customers has a mean value of 0.5 minute. These results demonstrate that MCF-T and MCF-P significantly outperform other policies in terms of *both* the number of customers that can be serviced concurrently and the average request waiting time. They also perform better than MQL and MFQL in fairness (towards unpopular videos) in the most important region of the curve (when the defection probability is less than 30%). In contrast with prior studies on Batching,

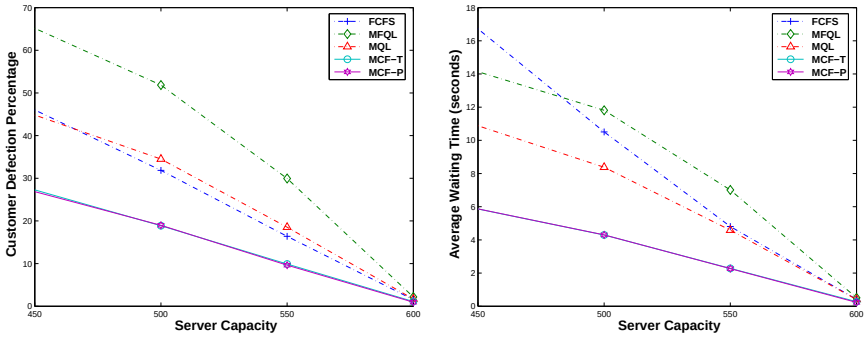
MFQL performs the worst in both the defection probability and the waiting times (which are the two most important metrics). As expected, FCFS always leads to the best fairness (which is the least important metric). FCFS and MQL produce close results in defection probability, but MQL achieves shorter waiting times, especially for low server capacities. MQL achieves the best overall performance among the previously proposed policies and thus can be used as a reference for comparison with our newly proposed schemes.

Figures 6 and 7 depict the results when Transition Patching and Patching are used, respectively. The results exhibit similar behavior as those for ERMT. Comparing the three sets of results demonstrate that the benefits of MCF-T and MCF-P in both the defection rate and waiting times generally are larger with more efficient stream merging. Figure 8 summarizes the percentage improvements achieved by MCF-P versus MQL under the three stream merging techniques. Note that the improvements achieved by MCF-P compared with MQL are up to 50% in *all* metrics when ERMT is used. Of course, when the number of server channels is over-provisioned, the server already can support all incoming requests immediately and thus scheduling becomes irrelevant. This explains why all policies perform the same beyond a certain point.

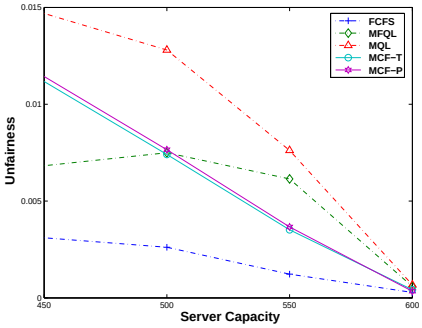
With Batching, FCFS can provide time of service guarantees and thus can encourage customers to wait, trading off waiting time for throughput [26]. One can certainly borrow the same idea in the case of some stream merging techniques, such as Patching and Transition Patching. However, with hierarchical stream merging techniques (such as ERMT), which are more scalable, FCFS cannot provide time of service guarantees since stream lengths change dynamically. Moreover, providing time of service guarantees achieves good throughput only when assuming a large mean for the customer waiting tolerance (μ_{tol}), such as 5 minutes, which may not be realistic these days because of the increasing expectations of customers for prompt service. Under the studied values of μ_{tol}, FCFS with time of service guarantees does not perform well, and thus the results are not shown to save space.

The conclusion here is that with proper weighing of performance metrics, MCF-P performs the best among all policies, followed by MCF-T. MCF-T generally has the advantage of somewhat better fairness since it does not favor longer queues, whereas MCF-P considers queue lengths. MCF achieves significant performance improvements in *both* of the most important metrics, compared with all other policies. It also improves the fairness significantly, compared with the best performer among existing policies, when the server

capacity is not too low.



(a) Defection Rate (b) Average Waiting Time

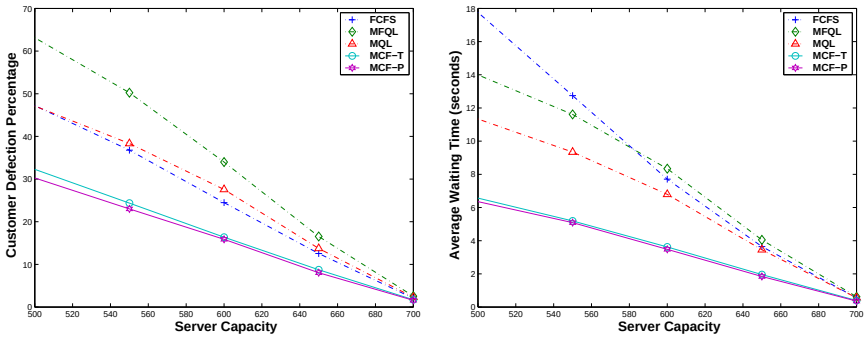


(c) Unfairness

Figure 5: Comparisons among Various Scheduling Policies [$ERMT$, $\mu_{tol} = 0.5$ minute]

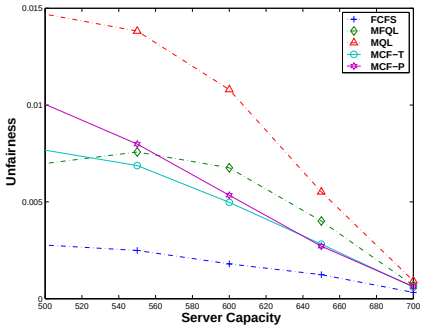
Impact of Customer Waiting Tolerance

Let us now compare the performance of the two implementations of MCF (MCF-T and MCF-P) under different levels of customer waiting tolerance. Only the results for ERMT are shown here to save space. The results for Patching and Transition Patching are similar. Figure 9 plots the comparisons for three values of the mean time of the waiting tolerance: 0.5 minute, 1.5 minutes, and 2.5 minutes. Smaller values correspond to higher customer expectations and services closer to true Video-on-Demand. As expected, the defection rate decreases with the waiting tolerance, but the average waiting



(a) Defection Rate

(b) Average Waiting Time



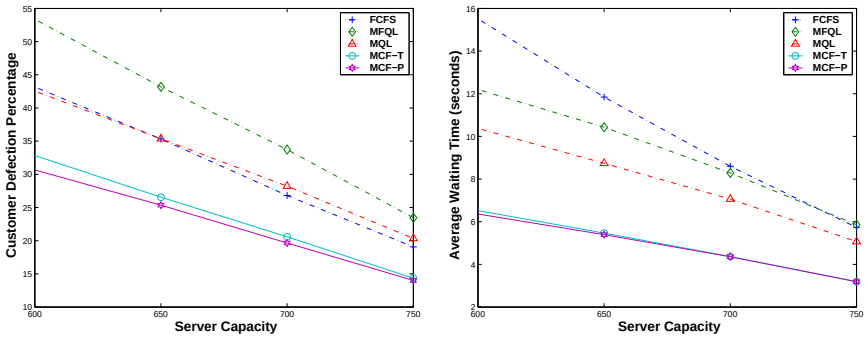
(c) Unfairness

Figure 6: Comparisons among Various Scheduling Policies [*Transition Patching*, $\mu_{tol} = 0.5$ minute]

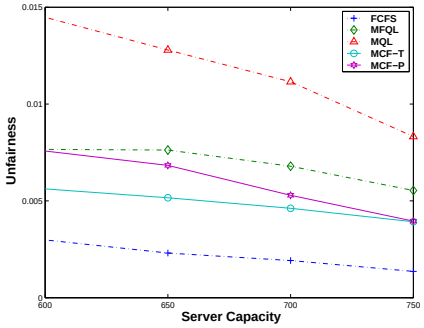
time increases. Also as expected, MCF-P performs better than MCF-T in the defection rate and waiting times, but MCF-T is more fair.

The performance gaps between MCF-T and MCF-P widen with the waiting tolerance. This happens because requests can wait longer and thus the variation in queue lengths increases. One might have expected the performance gaps to be much wider since MCF-P computes the cost per request and considers both stream lengths and queue lengths. There is, however, a correlation between the two factors that brings MCF-P close to MCF-T because more popular videos tend to have longer queues and shorter streams.

Let us now discuss the effectiveness of the best implementation of MCF (MCF-P) in comparison with the (generally) best performer among the previously proposed policies (MQL). Figure 10 compares these two policies under



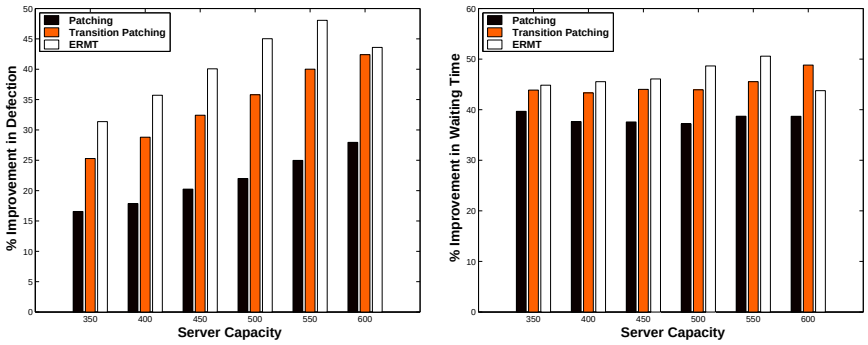
(a) Defection Rate (b) Average Waiting Time



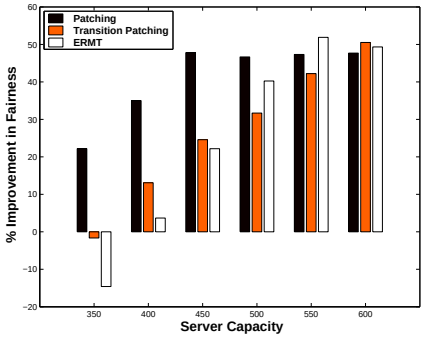
(c) Unfairness

Figure 7: Comparisons among Various Scheduling Policies [*Patching*, $\mu_{tol} =$ 0.5 minute]

three levels of waiting tolerance and shows that outstanding performance benefits achieved by MCF-P. The performance improvements achieved by MCF-P in the defection rate and waiting times generally increase with lower values of waiting tolerance. The reason is that the higher the tolerance is, the longer the waiting queues become for the more popular videos. Hence, the different queues vary more significantly in length and the queue-length factor plays a more important role, bringing MCF-P a little closer in behavior to MQL (although the performance gap remains significant).



(a) Defection Rate (b) Average Waiting Time

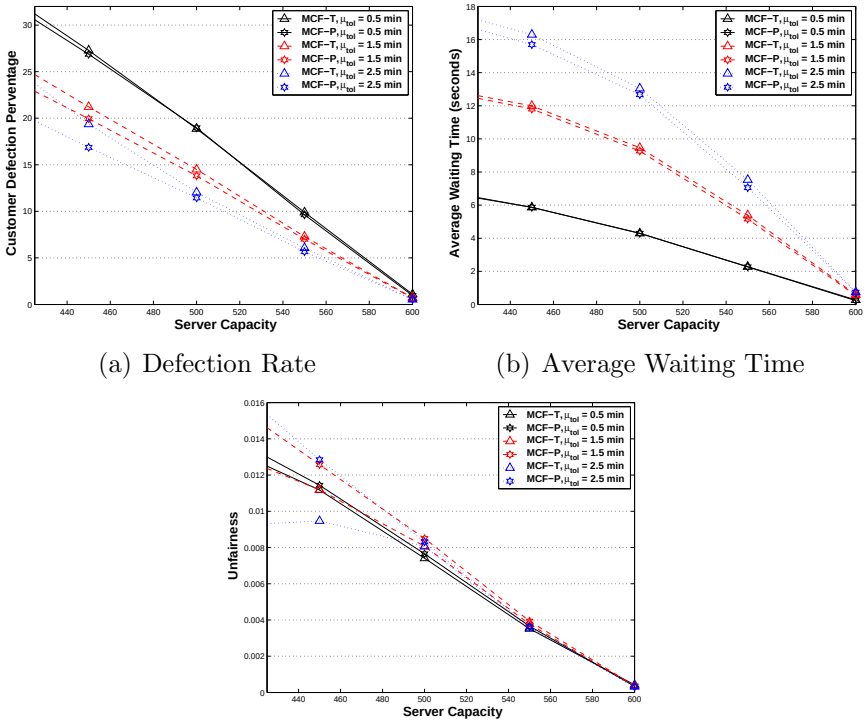


(c) Unfairness

Figure 8: Percentage Improvements of MCF-P vs MQL [$\mu_{tol} = 0.5$ minute]

Impacts of Other Parameters: Request Arrival Rate, Number of Videos, and Video Length

Figures 11, 12, and 13 depict the impacts of request arrival rate, number of videos, and video length, respectively. Server capacity is fixed at 500 channels. Only the results for Transition Patching are shown here for space limitation. The results for other stream merging techniques are similar. The results confirm that MCF-P consistently performs significantly better than previous policies in terms of the two most important metrics. MCF-T performs close to MCF-P in waiting times but the gaps between the two in defection probability are somewhat wide since the mean waiting tolerance in these sets of figures is 1.5 minute. When the arrival rate exceeds 45 requests/minute or the number of videos exceeds 160, MQL achieves lower



(a) Defection Rate

(b) Average Waiting Time

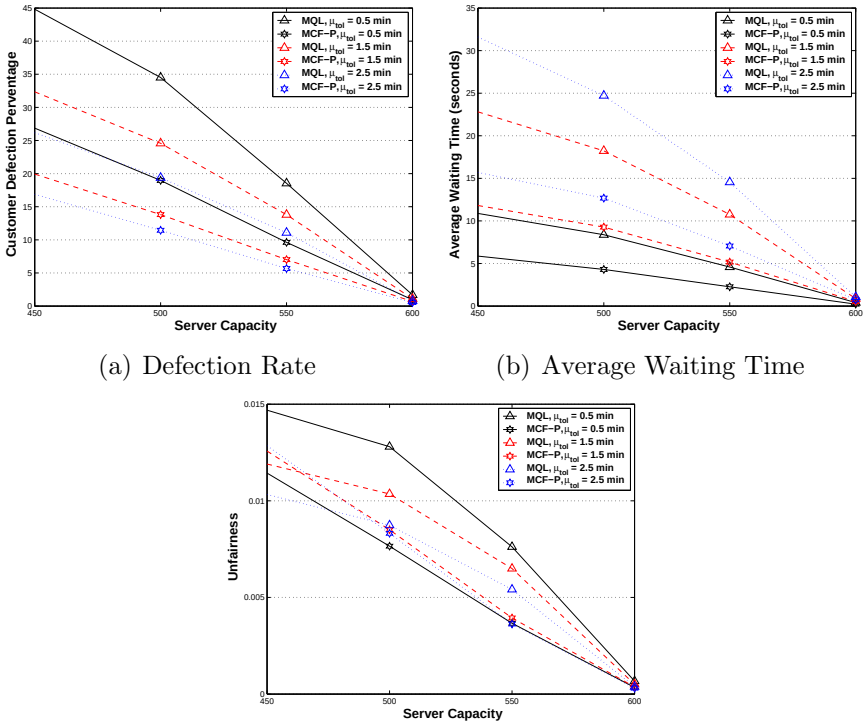
(c) Unfairness

Figure 9: Comparisons between MCF-T and MCF-P under Variable Customer Waiting Tolerance [*ERMT*]

defection probability than MCF-T. The unfairness with MCF-P and MCF-T grows at relatively fast rates with the arrival rate. MQL starts to be more fair than MCF-T when the arrival rate exceeds approximately 49 requests/minute and more fair than MCF-P when the arrival rate exceeds 58 request/minute. This behavior suggests that the impact of favoring short streams (which are likely for hot videos) in unfairness becomes more dominant than the impact of selecting longer queues as the request rate increases.

Analysis of Design Parameters: Wr and Wt

Let us now discuss the impact of scheduling on Wr and Wt. From this point on, we modify the optimal equations of these design parameters by replacing the arrival rates with the actual service rates, which are to be computed



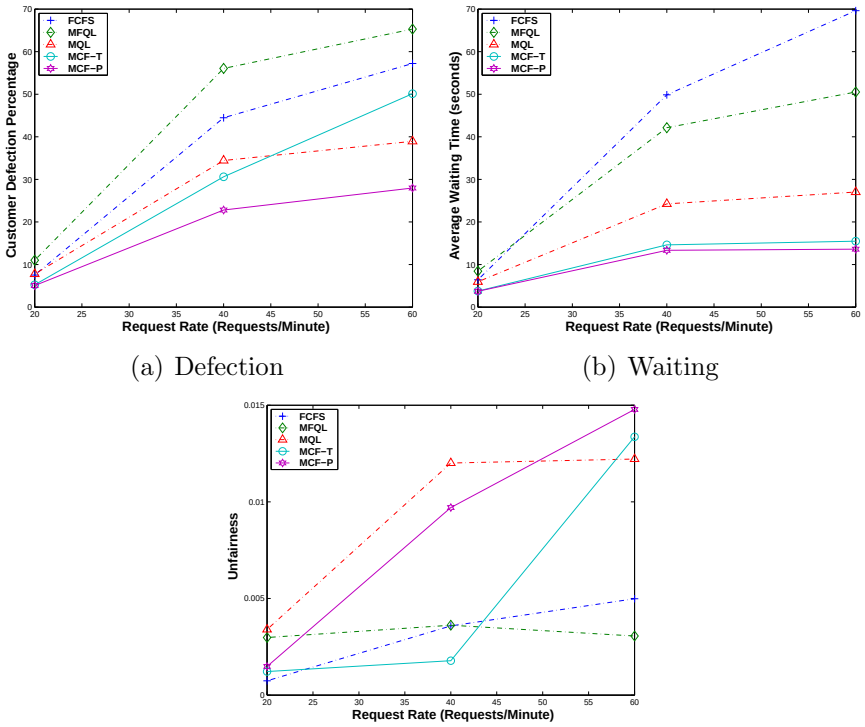
(a) Defection Rate (b) Average Waiting Time

(c) Unfairness

Figure 10: Comparisons between MQL and MCF-P under Variable Customer Waiting Tolerance [*ERMT*]

dynamically. The original equations were derived based on the assumptions of immediate service and no defections (i.e., True Video-on-Demand). For existing scheduling policies, the dynamic approach improves the performance of Patching and Transition Patching when the server is loaded. The improvements become negligible with MCF-T and MCF-P, which tend to be more resilient to such inaccuracies in computing Wr and Wt.

Figure 14 compares the actual values of Wr and Wt with FCFS, MQL, and the three variants of MCF-P (RAF, RAP, and RAS) for Videos 1 and 11 when Transition Patching is used. The results for Patching are similar and thus not shown. (Note that the videos are numbered in decreasing order of popularity.) RAS is implemented by updating Nr_i and Nt_i dynamically. The horizontal lines depict the values of Wr and Wt, computed based on the

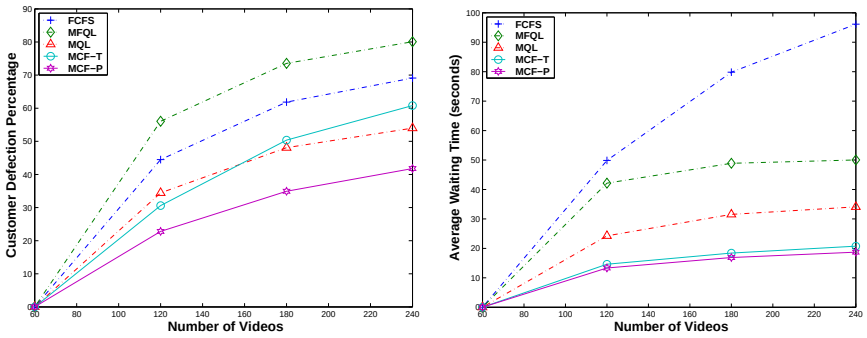


(a) Defection (b) Waiting

(c) Unfairness

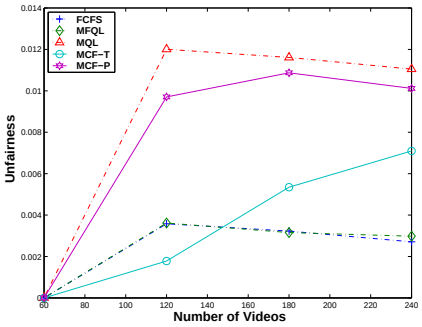
Figure 11: Impact of Request Arrival Rate [*Transition Patching*, 500 Server Channels, $\mu_{tol} = 1.5$ minute]

original equations (using the request arrival rates). To keep the figures clear, the results for MCF-T are dropped since it delays regular streams significantly and produces relatively large values. As expected, the actual values approach those of the original equations as the server capacity increases and the server becomes less loaded. Although RAF favors smaller streams, it produces smaller Wr than FCFS and MQL for Video 1 because of its high throughput and larger buildup of requests in that video's queue. For Video 11, however, which tends to have smaller number of requests, RAF has larger Wr, especially for low to moderate server capacities. By assigning lower cost to regular and transition patches, RAP and RAS produce the smallest Wr and Wt.



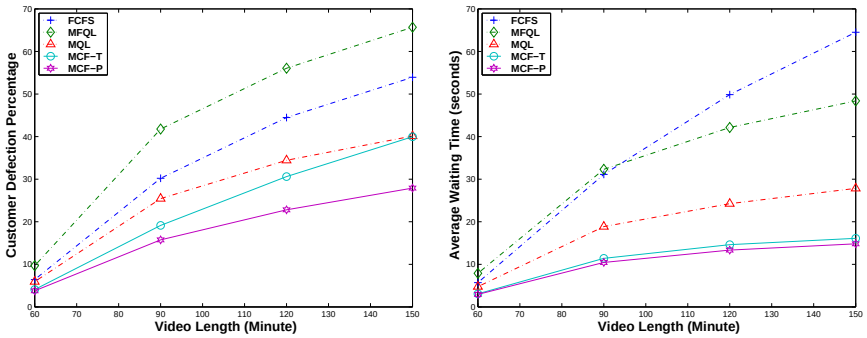
(a) Defection

(b) Waiting



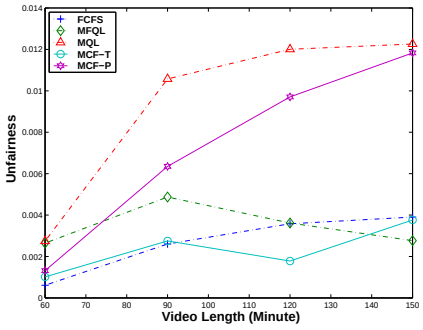
(c) Unfairness

Figure 12: Impact of Number of Videos [*Transition Patching*, 500 Server Channels,$\mu_{tol} = 1.5$ minute]



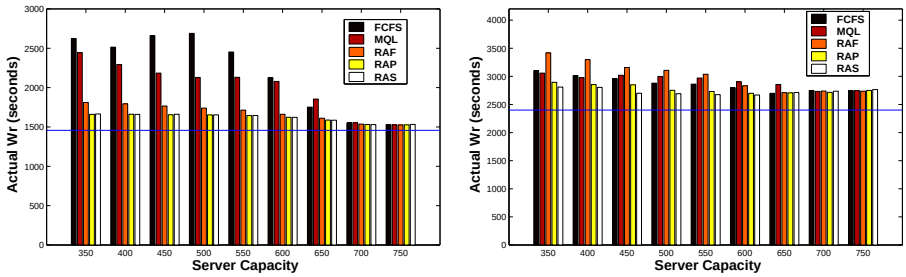
(a) Defection

(b) Waiting



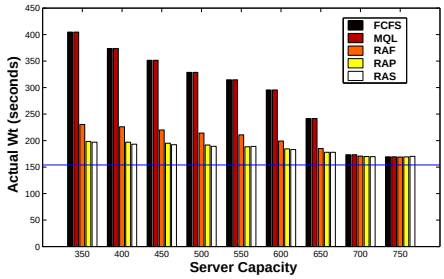
(c) Unfairness

Figure 13: Impact of Video Length [*Transition Patching*, 500 Server Channels, $\mu_{tol} = 1.5$ minute]

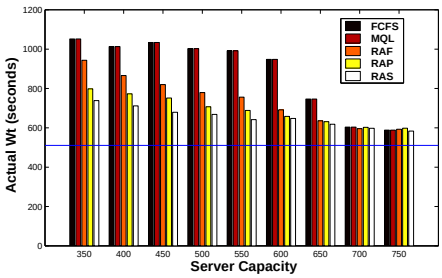


(a) Wr for Video 1

(b) Wr for Video 11



(c) Wt for Video 1

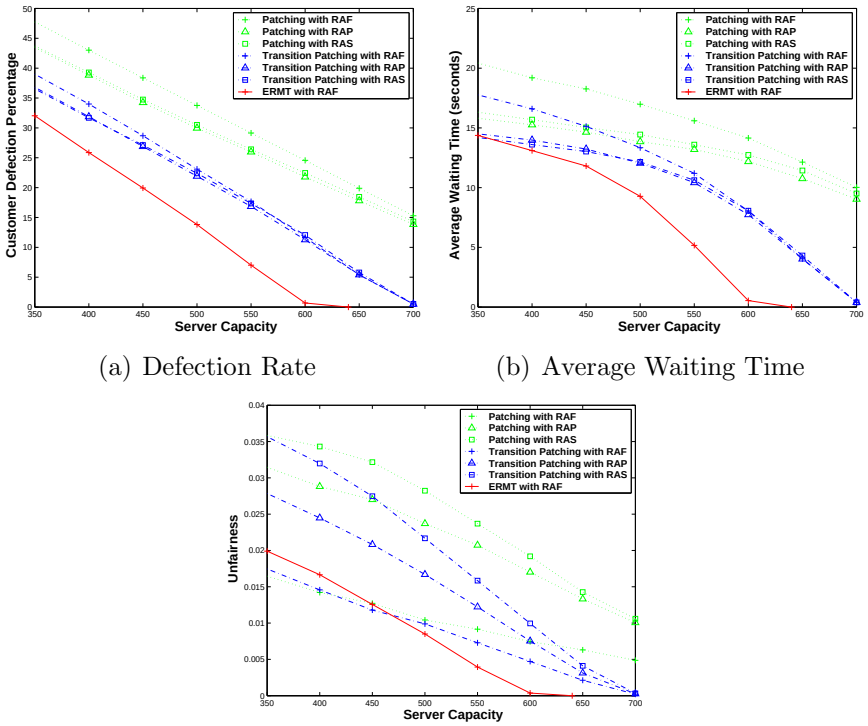


(d) Wt for Video 11

Figure 14: Actual Wr and Wt [*Transition Patching*, $\mu_{tol} = 1.5$ minutes]

Analysis of Stream Merging and MCF-P Variants

Let us now compare the performance of the three stream merging techniques when the newly proposed MCF-P policy is employed and analyze the three variants of MCF-P. (RAP and RAS are applicable for only Patching and Transition Patching.) Figure 15 plots the comparisons in the three metrics. The results are relatively similar to those with previous scheduling policies, such as MFQL and MQL. MCF-P, however, widens the performance gaps among stream merging techniques and makes the more complex ones even more efficient. The results indicate that the gaps between the RAF, RAP, and RAS are wider with Patching and that RAP is generally the best performer, although it is less fair than RAF. Moreover, RAP has smaller implementation complexity than RAS. The higher unfairness with RAP and RAS compared with RAF is due to treating regular streams and transition patches effectively as (or close to) patches, thereby increasing the chances of service to the requests for hot videos, which tend to have shorter patches.



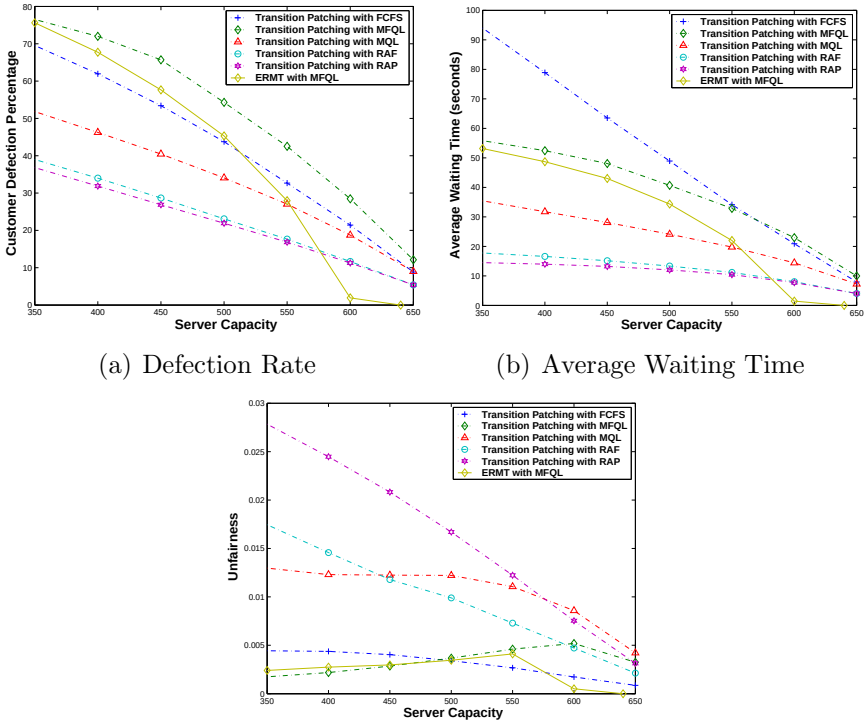
(a) Defection Rate (b) Average Waiting Time

(c) Unfairness

Figure 15: Comparisons among Stream Merging Techniques and MCF-P Variants[$\mu_{tol} = 1.5$ minutes]

Comparing the Performance of Scheduling and Stream Merging

Finally, let us compare the impact on performance of stream merging and scheduling. In Figure 16, ERMT uses one, poor scheduling policy (MFQL), whereas Transition Patching uses different scheduling policies with varying performance. These results demonstrate that the decision of what scheduling policy to select can be more important and have stronger impact on performance than the choice of a stream merging technique. For example, ERMT is the most scalable stream merging technique, but when applied with a poor scheduling policy, it performs worse than Transition Patching, when the server is loaded.



(a) Defection Rate (b) Average Waiting Time

(c) Unfairness

Figure 16: Comparing Stream Merging and Scheduling Performance [μ_{tol} = 1.5 minutes]

0.5 Conclusions

We have investigated the issue of request scheduling in streaming media servers applying stream merging. We have proposed a simple, yet highly effective scheduling policy, called *Minimum Cost First* (MCF), and have presented two alternative implementations: *MCF-T* and *MCF-P*. MCF captures the great variation in stream lengths caused by stream merging. Thus, it favors the requests requiring the least cost, which is proportional to the required stream length. MCF-T considers the cost for all requests in a video queue, whereas MCF-P computes the cost per request in the queue. We have also presented three MCF-P variants, which differ in how regular streams and transition patches are treated: *RAF*, *RAP*, and *RAS*.

We have compared various policies through extensive simulation in terms of three performance metrics: the customer defection probability, waiting times, and unfairness. We have considered the impacts of customer waiting tolerance, server capacity, request arrival rate, number of videos, and video lengths. The main results can be summarized as follows. (1) The results with stream merging differ significantly from prior results in the case of Batching. For example, MFQL performs poorly compared with MQL and FCFS, when stream merging is used. (2) Interestingly, MCF performs significantly better than existing policies in terms of *both* the defection probability and average request waiting time (the two most important performance metrics). Moreover, it performs better than both MQL and MFQL in terms of unfairness in the most important regions of operation. (3) MCF works well with all stream merging techniques. The benefits of MCF, however, in both the defection rate and waiting times become larger when a more efficient stream merging is used. Thus, the improvements are better with ERMT than Transition Patching and with Transition Patching than Patching. (4) The performance improvements achieved by MCF-P, compared with the best performer of existing policies (MQL), in the defection rate and waiting times, generally increase with lower customer waiting tolerance (corresponding to higher customer expectations and service closer to true Video-on-Demand). (5) The implementation MCF-P (RAP in particular) offers the best overall performance although it performs somewhat worse than MCF-T in unfairness, which is far less important than the other two performance metrics.

Bibliography

[1] E. L. Abram-Profeta and K. G. Shin. Providing unrestricted VCR functions in multicast Video-on-Demand servers. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 66 –75, June 1998.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. The maximum factor queue length batching scheme for Video-on-Demand systems. *IEEE Trans. on Computers*, 50(2):97–110, Feb. 2001.

[3] A. Bar-Noy, G. Goshi, R. Ladner, and K. Tam. Comparison of stream merging algorithms for Media-on-Demand. *Multimedia Systems Journal*, 9:211–223, 2004.

[4] A. Bar-Noy, R. Ladner, and T. Tamir. Scheduling techniques for Media-on-Demand. In *Annual ACM-SIAM Symposium on Discrete Algorthims*, pages 791–800, Jan. 2003.

[5] Y. Cai and K. A. Hua. An efficient bandwidth-sharing technique for true video on demand systems. In *Proc. of ACM Multimedia*, pages 211–214, Oct. 1999.

[6] Y. Cai and K. A. Hua. Sharing multicast videos using patching streams. *Multimedia Tools and Applications journal*, 21(2):125–146, Nov. 2003.

[7] Y. Cai, W. Tavanapong, and K. A. Hua. Enhancing patching performance through double patching. In *Proc. of 9th Int Conf. on Distributed Multimedia Systems*, pages 72–77, Sept. 2003.

[8] S. W. Carter and D. D. E. Long. Video-on-Demand server efficiency through stream tapping. In *Proceedings of 6th International Conference*

on Computer Communications and Networks (ICCCN), pages 200–207, Sep. 1997.

[9] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel allocation under batching and vcr control in movie-on-demand servers. *Journal of Parallel and Distributed Computing*, 30(2):168–179, Nov. 1995.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia*, pages 391–398, Oct. 1994.

[11] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for Video-on-Demand servers. In *Proc. of ACM Multimedia*, pages 199–202, Oct. 1999.

[12] D. L. Eager, M. K. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective Video-on-Demand. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, pages 206–215, Jan. 2000.

[13] D. L. Eager, M. K. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. on Knowledge and Data Engineering*, 13(5):742–757, Sept. 2001.

[14] C. Griwodz, M. Liepert, M. Zink, and R. Steinmetz. Tune to Lambda Patching. *ACM Performance Evaluation Review*, 27(4):20–26, March 2000.

[15] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true Video-on-Demand services. In *Proc. of ACM Multimedia*, pages 191–200, 1998.

[16] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan Video-on-Demand system. In *Proc. of ACM SIGCOMM*, pages 89–100, Sept. 1997.

[17] C. Huang, R. Janakiraman, and L. Xu. Loss-resilient on-demand media streaming using priority encoding. In *Proc. of ACM Multimedia*, pages 152–159, Oct. 2004.

[18] L. Juhn and L. Tseng. Harmonic broadcasting for Video-on-Demand service. *IEEE Trans. on Broadcasting*, 43(3):268–271, Sept. 1997.

[19] H. Ma, G. K. Shin, and W. Wu. Best-effort patching for multicast true vod service. *Multimedia Tools Appl.*, 26(1):101–122, 2005.

[20] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient broadcasting protocols for video on demand. In *Proc. of the Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 127–132, July 1998.

[21] B. Qudah and N. J. Sarhan. Analysis of resource sharing and cache management techniques in scalable video-on-demand. In *To Appear in Proceedings of IEEE/ACM International Symposium on Measurement and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2006.

[22] B. Qudah and N. J. Sarhan. Towards scalable delivery of video streams to heterogeneous receivers. In *To Appear in Proc. of ACM Multimedia*, Oct. 2006.

[23] M. Rocha, M. Maia, I. Cunha, J. Almeida, and S. Campos. Scalable media streaming to interactive users. In *Proc. of ACM Multimedia*, pages 966–975, Nov. 2005.

[24] N. J. Sarhan and C. R. Das. Caching and scheduling in NAD-based multimedia servers. *IEEE Trans. on Parallel and Distributed Systems*, 15(10):921–933, Oct. 2004.

[25] N. J. Sarhan and C. R. Das. A new class of scheduling policies for providing time of service guarantees in Video-On-Demand servers. In *Proc. of the 7th IFIP/IEEE Int'l Conf. on Management of Multimedia Networks and Services*, pages 127–139, Oct. 2004.

[26] A. K. Tsiolis and M. K. Vernon. Group-guaranteed channel capacity in multimedia storage servers. In *Proc. of ACM SIGMETRICS*, pages 285–297, June 1997.