Analysis of Waiting-Time Predictability in Scalable Media Streaming*

Mohammad A. Alsmirat
msmirat@wayne.edu

Musab Al-Hadrusi
hadrusi@wayne.edu

Nabil J. Sarhan
nabil@ece.eng.wayne.edu

Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI 48202

ABSTRACT

Providing video streaming users with expected waiting times enhances their perceived quality-of-service (QoS) and encourages them to wait. In the absence of any waiting-time feedback, users are more likely to defect because of the uncertainty as to when they will start to receive services. In this paper, we analyze waiting-time predictability in scalable video streaming. We present three prediction schemes and study their effectiveness when applied with various stream merging techniques and scheduling policies. The results demonstrate that the waiting time can be predicted accurately, especially when enhanced cost-based scheduling is applied. The combination of waiting-time prediction and cost-based scheduling leads to outstanding performance benefits.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.4 [**Computer Systems Organization**]: Performance of Systems; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Design, Performance

Keywords

Scheduling, stream merging, time-of-service guarantees, video streaming, waiting-time prediction.

1. INTRODUCTION

The interest in media streaming has grown dramatically over the Internet, cellular networks, and Cable TV. 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 stored content.

*This work was supported in part by NSF grant CNS-0626861.

MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany.

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. Client-pull strategies include *Batching* [7, 24, 1] and *Stream Merging* techniques [11, 3, 8, 19, 15, 18]. Batching increases resource sharing by accumulating the requests for the same videos and servicing them together using multicast streams. Stream merging techniques reduce the cost further by aggregating clients into larger groups that share the same multicast streams. These techniques include *Stream-Tapping/Patching* [6, 11, 4], *Transition Patching* [3, 5], and *Earliest Reachable Merge Target* (ERMT) [8, 9]. Server-push techniques (also called *Periodic Broadcasting* techniques) [12, 14, 16, 13, 23] 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 [7, 24, 1, 21, 20]. The main policies include *First Come First Serve* (FCFS) [7], *Maximum Queue Length* (MQL) [7], and *Maximum Factored Queue Length* (MFQL) [1]. For stream merging techniques, a cost-based scheduling policy, called *Minimum Cost First* [22], has been recently proposed. MCF-P captures the significant variation in stream lengths caused by stream merging techniques through selecting the requests requiring the least cost.

Most prior studies focused on only three main performance metrics: server throughput, average waiting time, and unfairness against unpopular videos. Motivated by the rapidly growing interest in human-centered multimedia, we consider other user-oriented metrics, such as the ability to inform users about how long they need to wait for service. Today, even for short videos with medium quality, users of online video websites may experience significant delays. The transition, in the near future, to streaming long videos (such as full-length movies) at high quality may lead to even larger delays.

This paper analyzes two approaches for giving waiting-time feedback to users of scalable video streaming. The first approach provides users with hard time-of-service guarantees. By contrast, the second approach, which is proposed here, provides expected times of service, or alternatively expected waiting times. Thus, this approach is referred to as the *predictive approach*. Providing users with waiting-time feedback enhances their perceived quality-of-service (QoS) and encourages them to wait, thereby increasing server throughput. In the absence of any waiting-time feedback, users are more likely to defect because of the uncertainty as to when they will start to receive services.

The issue of providing time of service guarantees has not been analyzed in the context of scalable video delivery techniques. A

policy, called *Next Schedule Time First* (NSTF), was proposed in [21] for Batching. This policy provides customers with schedule times and guarantees that they will be serviced no later than and accurately at their schedule times. We show that NSTF can be extended to stream merging but has two inherent shortcomings. First, NSTF performs significantly worse in server throughput and average waiting time than the recently proposed MCF policy, which utilizes aggressive cost-based scheduling but cannot provide time-of-service guarantees. Second, NSTF cannot work with hierarchical stream merging techniques (such as ERMT), which achieve the most scalable performance, because they may extend streams to satisfy the needs of new requests.

The proposed predictive approach eliminates the shortcomings of NSTF by providing expected waiting times of service (or approximate times of service) rather than hard time-of-service guarantees. This approach can be applied with the MCF to ensure the highest performance. We present three schemes for predicting the waiting times. The first two schemes simply use the average waiting time for all videos or for the requested video, respectively, as the expected waiting time. Thus, they dynamically compute the overall average waiting time or the average waiting time for each video. In contrast, the third scheme is highly intelligent and adaptive to server workload. It utilizes detailed information about the current state of the server to predict the waiting time and considers the applied scheduling policy. This information includes the current queue lengths, the completion times of running streams, and statistics such as the average request arrival rate for each video (which is to be updated periodically). This scheme predicts the future scheduling decisions over a certain period, called *prediction window*. It uses the completion times of running streams to know when channels will become available, and thus when waiting requests can be serviced. This scheme can be configured to allow any tradeoff between the prediction accuracy and the percentage of users receiving expected times.

The main contributions of this paper can be summarized as follows.

- We study how to provide time-of-service guarantees in scalable video delivery techniques by extending NSTF.
- We propose the predictive approach in which the server provides expected waiting times rather than hard time-of-service guarantees.
- We present three predictions schemes of waiting times that can be applied with any stream merging technique and any scheduling policy.
- We study the effectiveness of the two waiting-time feedback approaches (NSTF and the predictive approach) and the three prediction schemes under various stream merging techniques and scheduling policies through extensive simulation. The analyzed performance metrics include the overall customer defection (turn-away) probability, the average request waiting time, unfairness against unpopular videos, and prediction accuracy. Moreover, we evaluate the impacts of prediction window, server capacity, customer waiting tolerance, arrival rate, and number of videos.

The results show that the predictive approach is highly accurate and leads to outstanding performance benefits.

The rest of the paper is organized as follows. Section 2 provides background information. Section 3 discusses how to provide time-of-service guarantees in scalable video streaming, and Section 4 presents the proposed prediction schemes. Subsequently, Section 5 discusses the performance evaluation methodology and Section 6 presents and analyzes the main results. Finally, conclusions are drawn.

2. BACKGROUND INFORMATION

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* (Wr), a new regular stream is initiated instead.

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). A transition patch is shared by all subsequent patches until the next transition patch.

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 [8]. 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 [8, 10, 2].

These three stream merging techniques differ in complexity and performance. 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.

2.2 Request Scheduling

To facilitate scheduling, the server maintains a waiting queue for every video 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: (i) minimize the overall customer defection (turn-away) probability, (ii) minimize the average request waiting time, and (iii) 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:

$unfairness = \sqrt{\sum_{i=1}^{N_v}(d_i - \bar{d})^2/(N_v - 1)}$, where d_i is the defection probability for the waiting queue i, $\bar{d}$ is the mean defection 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.

- *First Come First Serve* (FCFS) [7] - This policy acts fairly by selecting the queue with the oldest request.
- *Maximum Queue Length* (MQL) [7] - 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) [1] - 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).
- *Minimum Cost First* (MCF) [22] – This policy has been recently proposed to exploit the variations in stream lengths caused by stream merging techniques. It gives 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. *MCF-P* (P for "Per") is the preferred implementation of MCF. It selects the queue with the least cost per request.

3. PROVIDING TIME-OF-SERVICE GUARANTEES

For Batching, a scheduling policy, called *Next Schedule Time First* (NSTF), was proposed in [21] to provide time-of-service guarantees. In this paper, we extend NSTF to work with stream merging techniques and analyze its effectiveness in this environment.

Let us start by discussing how (NSTF) works. NSTF assigns schedule times to incoming requests and guarantees that they will be serviced no later than scheduled. In addition, it ensures that these schedule times are very close to the actual times of service. When a new request calls for the playback of a video with no waiting requests, NSTF assigns the request a new schedule time that is equal to the closest unassigned completion time of a running stream. If the new request, however, is for a video that has already at least one waiting request, then NSTF assigns it the same schedule time assigned to the other waiting request(s) because all these requests can be serviced together using only one stream. NSTF eliminates some potential problems when the basic FCFS is used to provide time-of-service guarantees as done in [24].

When all waiting requests for a video are canceled, their schedule time becomes available and can be used by other requests. This leads to two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns the freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests who will wait beyond a certain threshold, and thus are likely to defect without being assigned better schedule times. Hence, requests that are assigned schedule times that require them to wait beyond a certain threshold should be notified that they may be serviced earlier.

NSTFo assigns each freed schedule time to an appropriate waiting queue that meets the following three conditions: (i) it is nonempty, (ii) its assigned schedule time is worse than the freed schedule time, and (iii) the expected waiting time for each request in it is beyond a certain threshold. If no candidate is found, NSTFo grants the freed schedule time to a new request. In contrast, if more than one queue meet these conditions, it selects the most appropriate one. The *NSTFo-MQL* implementation (which was shown to provide the best overall performance) selects the longest queue among the candidates. NSTFo-MQL combines the benefits of FCFS and MQL by assigning schedule times on a FCFS basis and reassigning freed schedule times on a MQL basis.

We present next an efficient generalized implementation of NSTF that can be applied for Batching as well as stream merging techniques. The server maintains a running queue (RQ), which keeps track of all currently running streams. These streams are stored in an increasing order of their completions times. To provide time-of-service guarantees, the server needs to maintain an index, $RQIndex$, which points to the next stream in RQ whose completion time has not been assigned yet. $RQ[0]$ is the first element of RQ, and it corresponds to the stream with the furthest completion time. $RQIndex$ is incremented every time the only waiting request in a queue gets canceled or when a queue is selected for service and is decremented every time a schedule time is assigned from the RQ. In addition, the server needs to maintain a free pool of freed schedule times. This pool can be implemented using a priority queue, where schedule times are placed in an ascending order. When a request for a video with no waiting requests arrives, the server first tries to assign it a schedule time from the top of the free pool. If the pool does not contain any live schedule times (i.e., schedule times that are further than the current time), then the server assigns it a new schedule time that corresponds to the next unassigned completion time.

To handle NSTF efficiently when stream merging techniques are applied, we propose and utilize two enhancements. First, NSTF triggers a schedule-time reassignment not only when a schedule time is freed but also when a new running stream has a closer completion time than an earlier stream whose completion time has already been assigned. The latter situation does not happen when Batching is applied because all streams are of the same length. In stream merging techniques, however, streams vary significantly in length, and thus the completion time of a new stream may be closer than that of an old stream. Assigning the new completion time to existing requests with significant expected waiting times enhances performance and increases fairness. Second, we improve the performance of NSTFo by utilizing the old schedule times that have been reassigned with better schedule times. When the requests for a certain video receive a new schedule time, their old schedule time can be assigned to the requests for the video with a worse schedule time. This enhancement, called *schedule-times cascading*, is valid because the reassignment of schedule times in NSTFo is highly constrained and the freed schedule times may not be assigned to the requests with the worst schedule time. This enhancement can also be used with Batching but is likely to be more effective with stream merging techniques.

4. WAITING-TIME PREDICTION

Unfortunately, the NSTF approach has two main inherent shortcomings. First, it may not perform well in terms of server throughput (or defection probability) and waiting times compared with other aggressive scheduling approaches, such as MCF-P. The cost-based approach is indeed hard to beat in terms of defection probability, especially by a policy whose main objective is to provide hard time-of-service guarantees and in which the initial assignment of schedule times is done on a FCFS basis. Second, NSTF cannot work with hierarchical stream merging techniques (such as ERMT) which achieve the highest performance. NSTF is incompatible with

these techniques because in hierarchical stream merging, streams may be extended to satisfy the needs of new users, thereby violating some time-of-service guarantees.

To overcome both these shortcomings, we propose a predictive approach of waiting times. In this approach, the server provides expected waiting times for service (or alternatively, approximate times of service) rather than hard time-of-service guarantees. The main advantage of this approach is that the server can use highly scalable scheduling policies, such as MCF-P, while improving customer-perceived QoS by informing customers with their expected waiting times. The success of the predictive approach depends on whether the waiting times can be accurately predicted when such scheduling policies are used. If the predictions are accurate, customers will appreciate the service, trust the service provider, and feel motivated to wait. By encouraging customers to wait, the server throughput will be further improved.

We present three schemes for predicting the waiting times:

- *Assign Overall Average Waiting Time* (AOW)
- *Assign Per-Video Average Waiting Time* (APW)
- *Assign Expected Stream Completion Time* (AEC)

The first scheme(AOW) simply computes the average waiting time for all requests dynamically and gives that value as the predicted waiting time for new requests. APW, however, keeps track of the average waiting time of the requests for each video and assigns it to the incoming requests for that video. Dealing with the requests for different videos in such a differentiated manner is more accurate because the behavior of most scheduling policies depends on the requested video. MQL, for example, tends to favor popular videos because they are more likely to have longer waiting queues. The third scheme (AEC) exhibits the highest intelligence. This scheme predicts the future scheduling decisions over a certain period of time and uses the completion times of running streams to know when channels will become available, and thus when waiting requests can be serviced.

4.1 Proposed AEC Scheme

Let us now discuss the proposed AEC scheme in more detail. Basically, this scheme predicts the waiting times (or times of service) by "simulating" the future behavior of the server. It utilizes detailed information about the current state of the server to predict the waiting time and considers the applied scheduling policy. This information includes the current queue lengths, the completion times of running streams, and statistics, such as the average request arrival rate for each video (which is to be updated periodically).

As discussed earlier, a stream's completion time indicates when a server channel will be free and can be used by new requests. Thus, the server knows when each channel will be available. The server can use these times as expected times of service for new requests. The assignment of a completion time to a new request is done by predicting the future scheduling decisions.

The basic idea of AEC can be explained as follows. When a new request arrives, the server determines the closest stream completion time that can be assigned to that request as the expected time of service. The server examines the completion times in the order of their closeness from the current time and finds the expected video to be serviced at that completion time. The process continues until the expected video to be serviced is the same as the currently requested video. To predict the scheduling outcome at a certain completion time, the server needs to estimate the video queue lengths at that completion time if the scheduling policy requires so (such as MQL, MFQL, and MCF) based on the video arrival rates, which are to be computed periodically, but not frequently. The expected queue length for video v at completion time T is given by

$$expected\_qlen[v] = qlen[v] + \lambda[v] \times (T - T_{Now}), \quad (1)$$

where $qlen[v]$ is the queue length of video v at the current time (T_{Now}), and $\lambda[v]$ is the arrival rate for video v. Note that the same video may be serviced again at later completion times. Equation (1) assumes that video v has not been identified before (while running the AEC algorithm to find the expected time of service for the new request) as the expected video to be serviced at an earlier stream completion time. Otherwise, the expected arrivals will have to be found during the time interval between the latest completion time (T_l) at which video v is expected to be serviced and T. In that case, the expected queue length for video v at completion time T is given by

$$expected\_qlen[v] = \lambda[v] \times (T - T_l). \quad (2)$$

Note that $qlen$ is not part of the equation since all existing requests (as of time T_{Now}) for video v are expected to be serviced at time T_l. To predict the scheduling decisions of MCF-P, AEC also considers the expected stream lengths required by various videos (in addition to the expected queue lengths).

To reduce the implementation complexity, AEC predicts the future scheduling decisions during only certain duration of time, called *prediction window* (W_p). Thus, the server needs to examine only the next stream completion times within W_p seconds from the arrival of the request for which we need to find the expected time of service. Therefore, AEC may not give an expected time of service for each request. Our analysis shows that giving an expected time for each request is not effective because it not only increases the implementation complexity but also decreases the prediction accuracy. The prediction window introduces a tradeoff between the percentage of requests receiving expected times of service and prediction accuracy.

Figure 1 shows a simplified algorithm of AEC. This algorithm is performed upon the arrival of request R_i to video v_j if the server is fully loaded. If the server is not fully loaded, the request can be serviced immediately. The assigned time for video v ($assigned\_time[v]$) corresponds to the latest completion time (T_l) at which video v is expected to be serviced in Equation (2).

Figure 2 demonstrates the basic idea of AEC. A new request for video 2 ($v2$) arrives at time T_{Now}. The server finds that video 1 ($v1$) is the expected video to be serviced at stream completion time $T1$. Then, the server examines the next completion time $T2$ (which is still within the prediction window) and determines that $v2$ is the most likely to be serviced at that time. Since $v2$ is the requested video for which we need to find the expected time of service, the prediction algorithm terminates by assigning $T2$ as the expected time of service to the new request.

The following points serve as further clarifications of AEC.

- The assignment of times of service is done based on predicting scheduling decisions, but the actual scheduling is kept totally isolated from prediction. Thus, the server performs scheduling solely based on the scheduling criterion and does not consider the assigned expected times of service in any way.
- The server may assign the same expected time to requests for different videos because the assignments are based on the current server state and workload, which vary with time. Similarly, the requests for the same video, which are currently together in the waiting queue, may receive different

```
for (v = 0; v < N_v; v++) // Initialize the assigned time for each video
    assigned_time[v] = −1;
T = closest completion time; // Start with closest completion time
while (T < T_Now + W_p) { // Loop till prediction window is exceeded
    // Find expected video queue lengths
    for (v = 0; v < N_v; v++){
        if (video v has not been assigned an expected time)
            expected_qlen[v] = qlen[v] + λ[v] × (T − T_Now);
        else
            expected_qlen[v] = λ[v] × (T − assigned_time[v]);
        Compute scheduling objective function for video v;
    } //for
    // Find the expected video to be served at time T
    expected_video = find video with minimum objective function;
    if (expected_video == v_j){
        Assign T to request R_i as the expected service time
        break; // Done
    }
    else
        assigned_time[expected_video] = T;
    T = next completion time; // Try again for the completion time
} //while
```

Figure 1: Simplified Algorithm for AEC [performed upon the arrival of request R_i to Video v_j]

expected waiting times although they will be serviced together. Later requests in the queue are more likely to receive more accurate predictions.

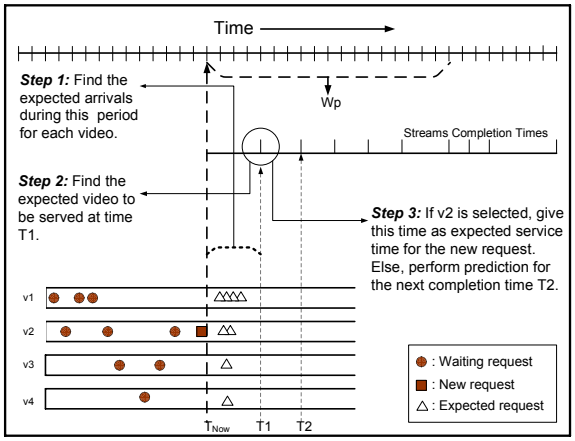
- The server may not give an expected time for each customer. Attempting to give each customer an expected time may lead to low prediction accuracy, and thus to users' mistrust in the expected times issued by the server.

4.2 Enhancements of AEC

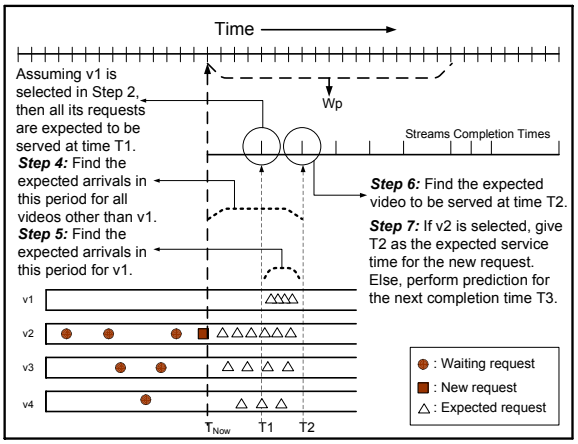
We present now the following enhancements of AEC.

Enhancement 1: Predict Stream Completion Times – This enhancement uses aggressive prediction. The server here not only predicts how the completion times will be assigned to incoming requests but also predicts new stream completion times and assigns them if possible to new requests. With this enhancement, when AEC assigns a stream completion time to a request as the expected time of service, it adds the expected completion time of the new stream to the set of the to-be examined completion times if its completion time falls within the prediction window. This enhancement is challenging to implement efficiently, especially with ERMT because the new "predicted" streams should impact the stream merging decisions as well to get accurate prediction. To isolate the actual request scheduling from prediction, the implementation creates a virtual running queue by duplicating the portion of the running stream queue containing all streams within the current prediction window. When a new stream is predicted to be scheduled at a certain completion time, its own completion time is inserted in the proper position in the virtual queue. Proper stream merging and potential stream extensions are performed in the virtual queue.

Enhancement 2: Account for User Defections – This enhancement accounts for the impact of user defections on the expected video queue lengths. The video waiting queues are likely to experience some defections, and these defections will become more significant during longer periods. Accounting for these defections is effective, especially for large prediction windows. AEC here needs to compute dynamically the defection rate for each video and to adjust the expected video queue length accordingly.



(a) Assignment of Completion Time T1



(b) Assignment of Completion Time T2

Figure 2: Clarification of AEC Prediction

Enhancement 3: Give Preference to Real Requests – With AEC, the expected queue lengths are computed and used to predict future scheduling decisions. An expected queue length includes a number of real requests and a number of expected requests. This enhancement values real requests more than expected requests. One interesting way to implement this enhancement (to a certain degree) is to truncate the expected queue lengths in Equations (1) and (2). Thus, if a video has an expected queue length less than one, it will not be selected as an expected video to be serviced at any stream completion time.

5. EVALUATION METHODOLOGY

We have extended the media streaming simulator in [17] by implementing NSTF and various prediction schemes and enhancements. We discuss next the workload characteristics and analyzed performance metrics.

5.1 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 and that the

access to videos is highly localized and follows a Zipf-like distribution with skewness parameter $\theta = 0.271$. We characterize the waiting tolerance of customers by three models. In *Model A*, the waiting tolerance follows an exponential distribution with mean μ_{tol}. In *Model B*, users with expected waiting times less than μ_{tol} will wait and the others will have the same waiting tolerance as Model A. We introduce *Model C* to capture situations in which users either wait or defect immediately depending on the expected waiting times. The user waits if the expected waiting time is less than μ_{tol} and defects immediately if the waiting time is greater than $2\mu_{tol}$. Otherwise, the defection probability decreases linearly from 1 to 0 for the expected waiting times between μ_{tol} and $2\mu_{tol}$.

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 150 to 850. We also study the impacts of arrival rate, number of videos, and video length.

Table 1: Summary of Workload Characteristics

Parameter	**Model/Value(s)**
Request Arrival	Poisson Process
Request Arrival Rate	Variable, Default = 40 Req./min
Server Capacity	150 to 850 channels
Video Access	Zipf-Like Skewness Parameter $\theta = 0.271$
Number of Videos	Variable, Default = 120
Video Length	Variable, Default 120 min
Waiting Tolerance Model	Models A, B, and C
Mean Waiting Tolerance (μ_{tol})	Variable, Default = 30 sec

5.2 Performance Metrics

We use two performance metrics to compare the effectiveness of various prediction schemes: *prediction accuracy* and *percentage of clients receiving expected times of service* (PCRE). The *average deviation* between the expected and actual times of service is used as a measure of accuracy. The accuracy decreases with the deviation. We compare the effectiveness of the predictive approach and NSTF approach in terms of customer defection probability, the average waiting, and unfairness against unpopular videos.

6. RESULT PRESENTATION AND ANALYSIS

6.1 Waiting Time Distribution

Let us start by comparing the waiting time distributions of requests resulting from various scheduling policies: FCFS, MQL, MCF-P (RAF), and MCF-P (RAP). The two implementations of MCF-P vary in how regular streams and transition streams are treated. *Regular as Full* (RAF) uses the full lengths of regular and transition streams in computing the cost, whereas *Regular as Patch* (RAP) treats them as if they were patches. (Only RAF is applicable in the case of ERMT.) Figure 3 depicts the overall waiting time distributions (considering all videos), and the waiting distributions of two individual videos with significantly varying popularities. Only the results for Patching are shown. The results for Transition Patching and ERMT are similar, and thus not shown to save space. The waiting times are distributed between 0 and 30 seconds because the waiting tolerance is set to 30 seconds. The waiting times with MCF-P (RAP) and MCF-P (RAF) are concentrated around $0 - 5$ seconds and decay quickly as we approach large values, in a manner similar to the exponential distribution. Moreover, the decay is faster for more popular videos. The waiting times with MQL follow the same pattern, but the decay happens less quickly. By contrast, FCFS produces a bell-shaped distribution. These results suggest that using the average value to predict the waiting time leads to the lowest accuracy in FCFS and the highest accuracy in MCF-P (RAF) and MCF-P (RAP).

6.2 Effectiveness of Prediction Schemes

Let us now compare the effectiveness of the three prediction schemes (AOW, APW, and AEC) in terms of accuracy, which is the most important metric in this case. Figures 4 and 5 depict the average deviation results for Patching and Transition Patching under three different scheduling policies: MQL, MCF-P (RAF), and MCF-P (RAP). (FCFS, in the form of NSTF, is more suited for providing hard time-of-service guarantees than prediction. Subsection 6.4 analyzes the performance of NSTF.) To save space, Figure 6 shows the results for ERMT under only MCF-P (RAF). AEC here is integrated with Enhancements 1 and 2. These figures demonstrate that AEC performs significantly better than the other two schemes and the results are better with more scalable stream merging. The deviation with AEC is within only two seconds. AOW and APW have the advantage of giving an expected time of service to each user, but the accuracy of these expectations is a more important factor.

The two variants of MCF-P (RAF and RAP) perform very close to each other in accuracy. MCF-P, however, is more predictable than MQL because the scheduling decisions of MCF-P are based on both the queue lengths and the required stream costs, whereas MQL uses only the queue lengths. The required stream cost for a video can be determined precisely, but the queue lengths in the future require prediction. Fortunately, MCF-P is not only more predictable than MQL but also achieves better performance (as shown in [22]) in terms of defection probability, average waiting time, and unfairness. From this point on, we consider only AEC and MCF-P.

6.3 Analysis of AEC

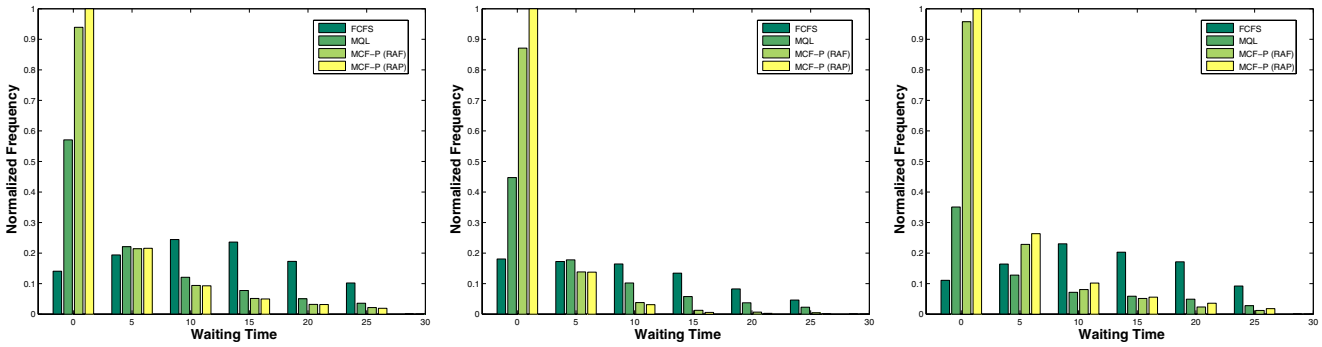
Let us now analyze the performance of AEC in more detail by examining the impact of the prediction window, the impact of various workload parameters, and the effectiveness of Enhancement 3.

6.3.1 Impact of Prediction Window

Figure 7 plots the impact of prediction window (W_p) on the accuracy and the percentage of clients receiving expected times of service (PCRE) for the three stream merging techniques. As expected, both the deviation and PCRE increase with W_p. PCRE, however, increases sublinearly with W_p. The accuracy and PCRE are generally better with more scalable stream merging, except for large values of W_p.

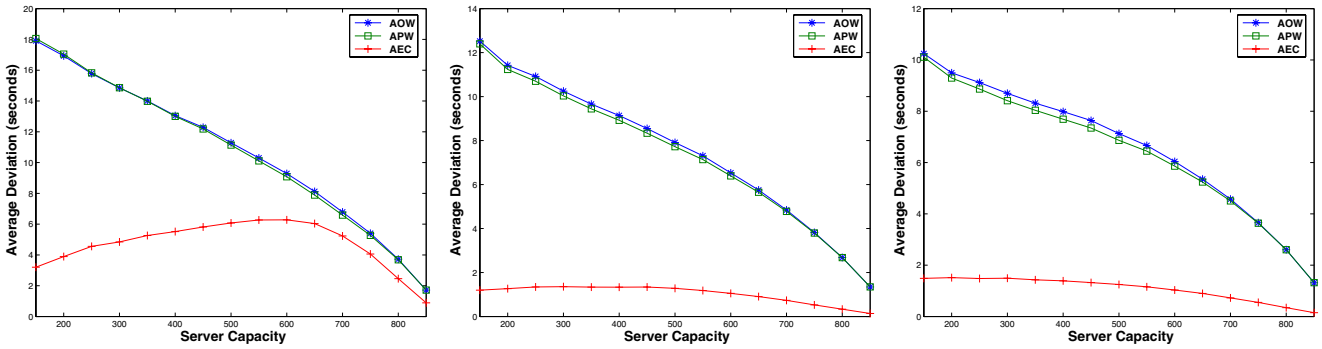
6.3.2 Impact of Workload Parameters

Figures 8 and 9 show the impacts of customer waiting tolerance, request arrival rate, and number of videos on the effectiveness of AEC in terms of accuracy and PCRE, respectively. The deviation increases at a relatively high rate with the waiting tolerance because requests stay longer in the waiting queue and queue lengths become harder to predict. We believe that smaller values of the mean waiting tolerance are more realistic because the expectations of customer these days are getting much higher and their waiting tolerance is getting lower. The deviation also increases with the arrival rate and the number of videos but remains with 2 seconds



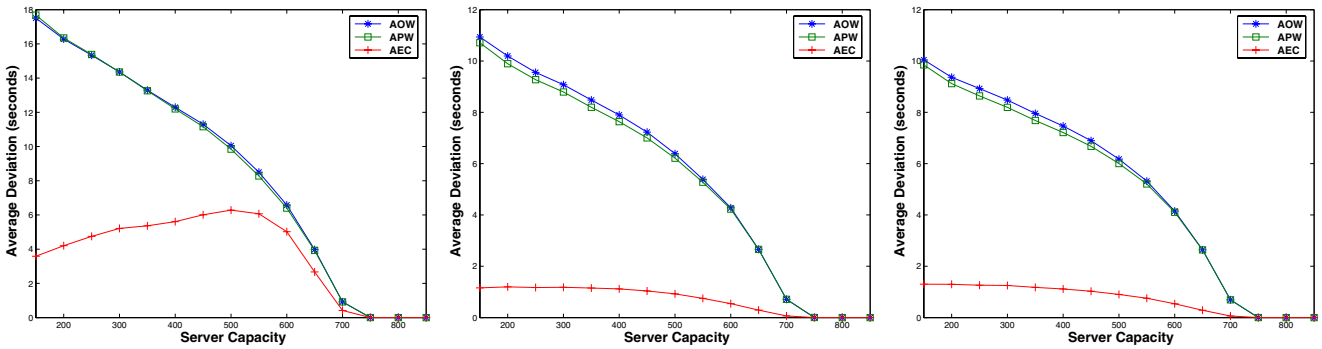
(a) All Videos (b) Video 1 (c) Video 10

Figure 3: Waiting Time Distribution [*Patching*, 600 Channels, Model A]



(a) MQL (b) MCF-P (RAF) (c) MCF-P (RAP)

Figure 4: Comparing Prediction Schemes [*Patching*, $W_p = 0.5\mu_{tol}$, Model A]



(a) MQL (b) MCF-P (RAF) (c) MCF-P (RAP)

Figure 5: Comparing Prediction Schemes [*Transition Patching*, $W_p = 0.5\mu_{tol}$, Model A]

even for up to 70 requests/minute and 240 supported videos. PCRE decreases with the arrival rate and number of videos but does not change much with the waiting tolerance.

6.3.3 *Effectiveness of Giving Preference to Real Requests*

Let us now discuss the effectiveness of Enhancement 3 (Giving Preference to Real Requests). Enhancements 1 and 2 are integrated with AEC because they always lead to better performance. Enhancement 3, however, presents a tradeoff between accuracy and PCRE, and thus we need to evaluate it separately. Figure 10 shows that Enhancement 3 significantly improves PCRE, but at the expense of accuracy. Enhancement 3 may be a good choice in certain situations, especially since the deviation is within 3 seconds.

6.4 Effectiveness of the Predictive Approach Compared with NSTF

Let us now compare the effectiveness of the proposed predictive approach and the approach of providing time of service guarantees. Figures 11 and 12 compare the performance of the "Predictive MCF-P" and NSTF in customer defection probability, average waiting time, and unfairness for Model B and C of the waiting tolerance, respectively. The two variants of the Predictive MCF-P

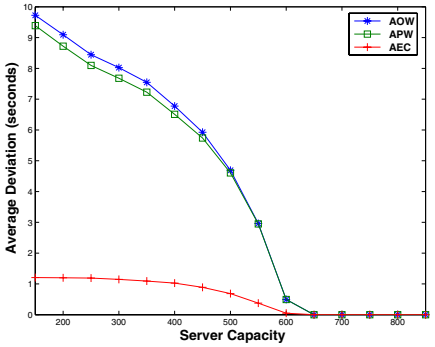


Figure 6: Comparing Prediction Schemes [*ERMT*, MCF-P, $W_p = 0.5\mu_{tol}$, Model A]

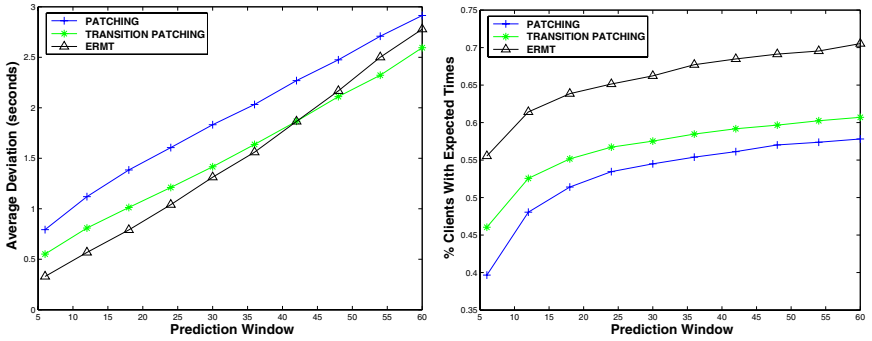
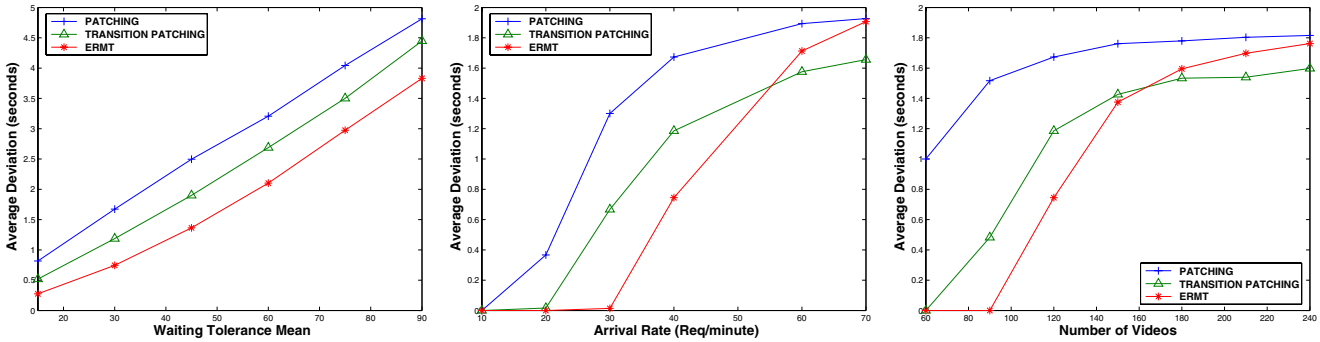
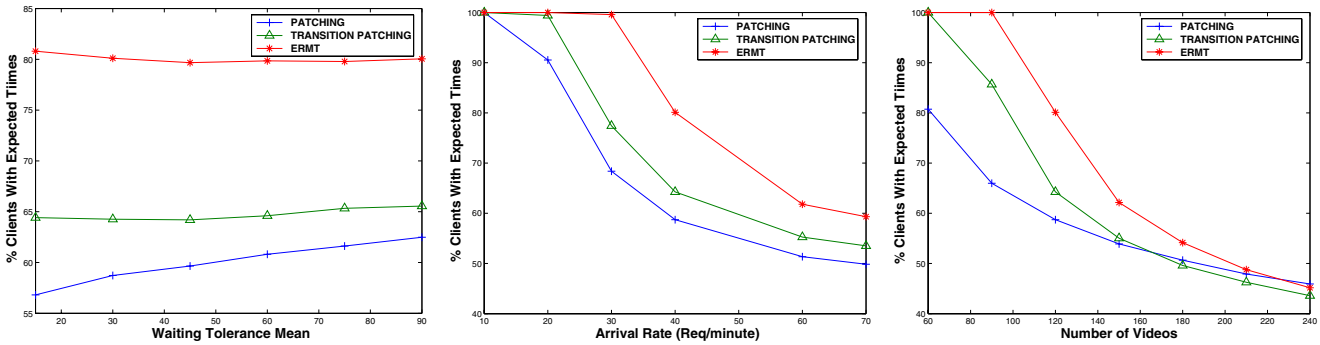


Figure 7: Impact of Prediction Window [500 Channels, Model A]



(a) Impact of Waiting Tolerance (b) Impact of Arrival Rate (c) Impact of Number of Videos

Figure 8: Impact of Workload on Average Deviation [500 Channels, Model A]



(a) Impact of Waiting Tolerance (b) Impact of Arrival Rate (c) Impact of Number of Videos

Figure 9: Impact of Workload on PCRE [500 Channels, Model A]

apply the proposed AEC prediction scheme. The best implementation of NSTF (NSTFo-MQL) is used to ensure a fair comparison. These results demonstrate that predictive MCF-P performs significantly better than NSTF under both tolerance models in terms of the two most important performance metrics. MCF-P (RAP) performs slightly better than MCF-P (RAF) in defection probability and average waiting times but is less fair than MCF-P (RAF).

Finally, Figure 13 captures the fact that NSTF cannot be applied with ERMT, whereas the predictive approach can. The figure compares NSTF and predictive MCF-P when each is applied with the most scalable stream merging technique that is applicable to it. Here, only the results under Model C (which is more realistic) are shown for space limitation. Model B exhibits a similar behavior. The results demonstrate the outstanding performance gains achieved by applying the predictive approach in terms of the two most important performance metrics.

7. CONCLUSIONS

We have proposed a predictive approach of the user waiting time in scalable video streaming. We have presented three prediction schemes: *Assign Overall Average Waiting Time* (AOW), *Assign Per-Video Average Waiting Time* (APW), and *Assign Expected Stream Completion Time* (AEC). AOW and APW simply use the average waiting time for all videos or for the requested video, respectively,

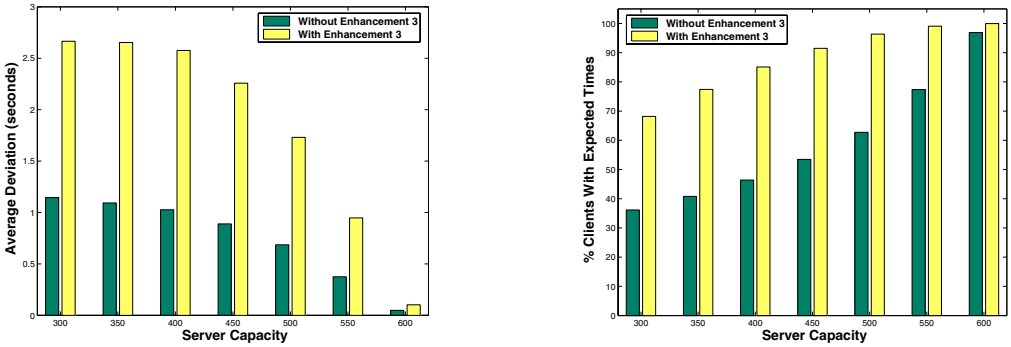
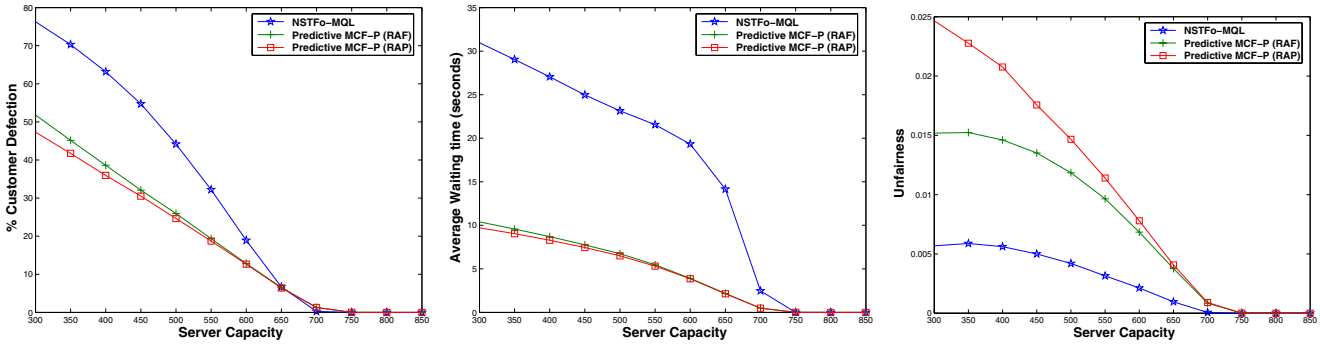
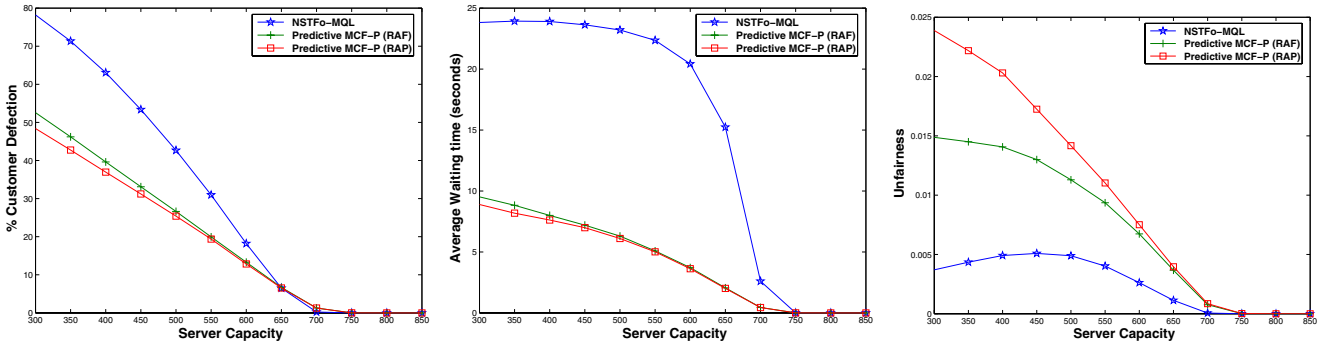


Figure 10: Effectiveness of Enhancement 3 [*ERMT*, MCF-P, Model A]



(a) Customer Defection (b) Waiting Time (c) Unfairness

Figure 11: Comparing NSTF with Predictive MCF-P (RAP) and MCF-P (RAF) [*Transition Patching*, $W_p = 0.5\mu_{tol}$, Model B]



(a) Customer Defection (b) Waiting Time (c) Unfairness

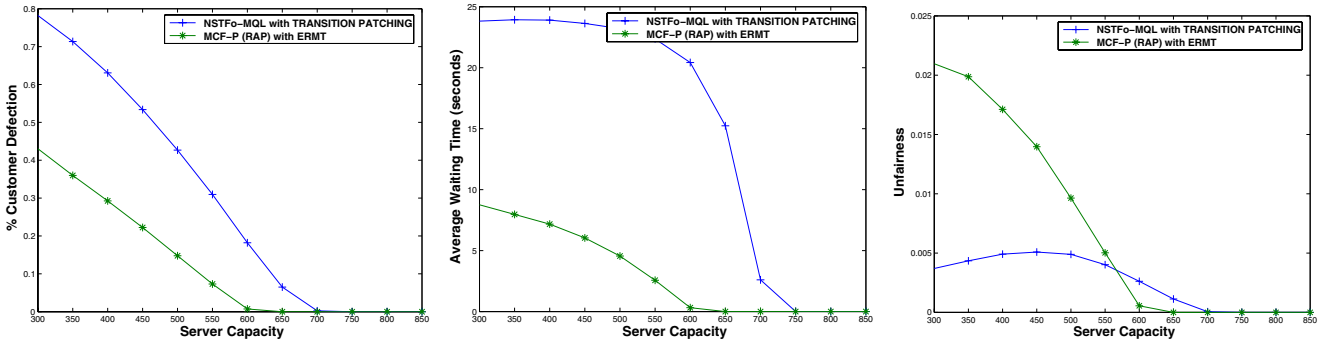
Figure 12: Comparing NSTF with Predictive MCF-P (RAP) and MCF-P (RAF) [*Transition Patching*, $W_p = 0.5\mu_{tol}$, Model C]

as the expected waiting time. In contrast, AEC utilizes detailed information about the server state and considers the applied scheduling policy to predict the future scheduling decisions over a certain period, called *prediction window*. This window introduces a trade-off between the prediction accuracy and the number of users receiving expected waiting times.

We have analyzed the effectiveness of the three prediction schemes when applied with various stream merging techniques and scheduling policies. We have also compared the effectiveness of the predictive approach with the *Next Schedule Time First* (NSTF) policy, which provides hard time-of-service guarantees. In addition, we have studied the impacts of prediction window, server capacity, customer waiting tolerance, arrival rate, and number of videos.

The main results can be summarized as follows.

- The waiting time can be predicted accurately, especially with AEC and when MCF-P is used. MCF-P is not only highly predictable (in terms of user waiting time) but also achieves the best performance in server throughput and average waiting time.
- Combining AEC with MCF-P leads to outstanding performance benefits, compared with NSTF.
- This combination, called *Predictive MCF-P*, can be applied with hierarchical stream merging techniques (such as ERMT) to improve performance further, whereas NSTF cannot.



(a) Customer Defection (b) Waiting Time (c) Unfairness

Figure 13: Comparing NSTF and Predictive MCF-P, Each with Its Most Scalable Stream Merging Technique [Model C]

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

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

[6] S. W. Carter and D. D. E. Long. Improving Video-on-Demand server efficiency through stream tapping. In *the Int'l Conf. on Computer Communication and Networks (ICCCN)*, pages 200–207, Sept. 1997.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] B. Qudah and N. J. Sarhan. Analysis of resource sharing and cache management techniques in scalable video-on-demand. In *Proc. of the 14th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, page 327–334, Sept. 2006.

[18] B. Qudah and N. J. Sarhan. Towards scalable delivery of video streams to heterogeneous receivers. In *Proc. of ACM Multimedia*, pages 347–356, Oct. 2006.

[19] 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.

[20] 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.

[21] 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.

[22] N. J. Sarhan and B. Qudah. Efficient cost-based scheduling for scalable media streaming. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, Jan. 2007.

[23] L. Shi, P. Sessini, A. Mahanti, Z. Li, and D. L. Eager. Scalable streaming for heterogeneous clients. In *Proc. of ACM Multimedia*, pages 337–346, October 2006.

[24] 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.