Waiting-Time Prediction in Scalable On-Demand Video Streaming

NABIL J. SARHAN, MOHAMMAD A. ALSMIRAT, and MUSAB AL-HADRUSI
Wayne State University

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 their services will start. We analyze waiting-time predictability in scalable video streaming. We propose two 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 ProtocolsC.4 [**Performance of Systems**]: I.6.5 [**Simulation and Modeling**]: Model Development

General Terms: Design, Performance

Additional Key Words and Phrases: Scheduling, stream merging, time-of-service guarantees, video streaming, waiting-time prediction

ACM Reference Format:

Sarhan, N. J., Alsmirat, M. A., and Al-Hadrusi, M. 2010. Waiting-time prediction in scalable on-demand video streaming. ACM Trans. Multimedia Comput. Commun. Appl. 6, 2, Article 11 (March 2010), 25 pages.
DOI = 10.1145/1671962.1671967 http://doi.acm.org/10.1145/1671962.1671967

1. INTRODUCTION

Due to the phenomenal growth of social media and online video Web sites, video streaming has recently grown dramatically in popularity over the Internet. There has also been growing interest in video streaming over wireless 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 prerecorded content.

A preliminary version of this article [Alsmirat et al. 2007] was presented at ACM Multimedia 2007.

This work was supported in part by NSF grants CNS-0626861 and CNS-0834537.

Authors' address: N. J. Sarhan, M. Alsmirat, and M. Al-Hadrusi, Department of Electrical and Computer Engineering, Media Research Lab, Wayne State University, Detroit, MI 48202; email: {nabil, msmirat, hadrusi}@wayne.edu.

DOI 10.1145/1671962.1671967 http://doi.acm.org/10.1145/1671962.1671967

Scalable delivery of video streams can be achieved using *stream merging* [Hua et al. 1998; Cai and Hua 1999; Eager et al. 1999; Rocha et al. 2005; Ma et al. 2005; Qudah and Sarhan 2006; Al-Hadrusi and Sarhan 2008] and *periodic broadcasting* [Hua and Sheu 1997; Juhn and Tseng 1997; Pâris et al. 1998; Huang et al. 2004; Shi et al. 2006]. These techniques offer scalable performance when compared with unicast and batching [Dan et al. 1994; Tsiolis and Vernon 1997]. Batching merely accumulates the requests for the same videos, and services them together using multicast streams. Stream merging techniques further reduce the cost by aggregating users into larger groups that share the same multicast streams. Request scheduling is an important aspect for these techniques. A cost-based scheduling policy, called *Minimum Cost First* [Sarhan and Qudah 2007], has recently been proposed to capture the significant variation in stream lengths caused by stream merging. While stream merging delivers data in a client-pull fashion (on-demand), periodic broadcasting techniques deliver data in a server-push fashion by dividing each video into multiple segments and broadcasting each segment periodically on dedicated server channels. Thus, they can be used only for the most popular videos and require the users to wait until the next broadcast time of the first segment. This article considers the stream merging approach.

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 Web sites may experience significant delays. The transition, in the near future, to streaming long videos (such as full-length movies) at high quality (such as HDTV) may lead to even larger delays.

This article analyzes waiting-time predictability in scalable video streaming services. In particular, it seeks to assess through an extensive analysis, whether the waiting times can be accurately predicted when stream merging techniques are employed. Moreover, it investigates the impacts of stream merging techniques, scheduling policies, and numerous workload and design parameters. Providing users with waiting-time feedback enhances their perceived quality-of-service (QoS) and encourages them to wait (given that the waiting times are not too long), 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 proposed waiting-time prediction approach provides users with expected times of service, or alternatively, expected waiting times.

To assess the effectiveness of the waiting-time prediction approach, we present and analyze two alternative prediction schemes. The first, called *Assign Expected Stream Completion Time* (AEC), is highly intelligent and adaptive to server workload by utilizing detailed information about the current state of the server and considering the specific dynamic nature of the applied scheduling policy. This information includes the current queue lengths, the completion times of running streams, and regularly updated statistics, such as the average request arrival rate for each video. AEC uses the completion times of running streams to know when server channels will become available, and thus when waiting requests can be serviced. The main idea of AEC is to predict the future scheduling decisions over a certain period, called *prediction window*. As the prediction window increases, the percentage of users receiving expected times increases, but at the expense of increasing the implementation complexity and more importantly reducing the prediction accuracy. The prediction accuracy is an essential QoS metric that also contributes to establishing the users' trust and confidence in the provided expected times, and thus should not be significantly reduced to provide an expected time to each user. We utilize feedback control theory to tune the value of the prediction window to allow a prespecified tolerance in the accuracy. The inability of AEC to provide an expected time to each user is addressed by the second scheme, called *Hybrid Prediction*. This scheme employs two predictors.

This article compares the effectiveness of the waiting-time prediction approach with another approach that provides users with time-of-service guarantees. 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) [Sarhan and Das 2004], was proposed for batching. This policy provides users with schedule times and guarantees that they will be serviced no later than scheduled. We show that NSTF can be extended to stream merging but has two inherent shortcomings. First, it performs significantly worse in 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 *Earliest Reachable Merge Target* (ERMT) [Eager et al. 1999; 2000]), which achieve the most scalable performance, because they may extend streams to satisfy the needs of new requests.

The proposed waiting-time prediction 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 MCF and hierarchical stream merging to ensure the highest performance.

The main contributions of this article can be summarized as follows. (1) We study how to provide time-of-service guarantees in scalable video delivery techniques by presenting an extended NSTF policy, called *Generalized NSTF* (GNSTF). (2) We propose the waiting-time prediction approach, which provides users with expected waiting times rather than hard time-of-service guarantees. (3) We present two prediction schemes, which can be applied with any stream merging technique and scheduling policy. (4) We utilize feedback control theory to tune the value of the prediction window to allow a prespecified tolerance in the prediction accuracy. (5) We study the effectiveness of the two waiting-time feedback approaches (GNSTF and the waiting-time prediction approach) and the two prediction schemes, under various stream merging techniques and scheduling policies, through extensive simulation. The analyzed performance metrics include the overall user defection (turn-away) probability, the average request waiting time, unfairness against unpopular videos, prediction accuracy, and the percentage of users receiving expected times. Furthermore, we evaluate the impacts of prediction window, server capacity, user's waiting tolerance, skew in video access, arrival rate, video length, and number of videos.

The results show that the waiting-time prediction approach is highly accurate and leads to outstanding performance benefits.

The rest of the article 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.

2. BACKGROUND INFORMATION

2.1 Main Performance Metrics

The main performance metrics of scalable media streaming servers are *user defection probability*, *average waiting time*, and *unfairness*. The defection probability is the probability that a new user leaves the server without being serviced because of a waiting time exceeding the user's tolerance. It is the most important metric, followed by the average waiting time, because it translates directly to the number of users that can be serviced concurrently and to server throughput. Unfairness measures the bias of a scheduling policy against unpopular videos and can be computed as the standard deviation of the defection probability among the videos.

2.2 Stream Merging

Stream merging techniques aggregate users into larger groups that share the same multicast streams. In this subsection, we discuss three main stream merging techniques: patching [Carter and Long 1997; Hua et al. 1998; Cai and Hua 2003], transition patching [Cai and Hua 1999; Cai et al. 2003], and ERMT [Eager et al. 1999; 2000].

In patching, a new request immediately joins the latest full-length multicast stream for the video and receives the missing portion as a *patch* using a unicast stream. A full-length multicast stream is called a *regular stream*. Both the regular and patch streams are delivered at the video playback rate. The length of a patch stream, and thus its delivery cost, are proportional to the temporal skew from the latest regular stream. The playback starts using the data received from the patch stream, whereas the data received from the regular stream is locally buffered for use upon the completion of the patch stream. Because patch streams are not sharable with later requests and their cost increases with the temporal skew from the latest regular stream, it is cost-effective to start a new regular stream when the patch stream length exceeds a certain value, called *regular window* (*Wr*).

Transition patching allows some patch streams to be sharable by extending their lengths. It introduces another multicast stream, called *transition stream*. The threshold to start a regular stream is *Wr*, as in Patching, and the threshold to start a transition stream is called *transition window* (*Wt*).

ERMT is a near optimal hierarchical stream merging technique. Whereas a stream can merge at most once in patching and at most twice in transition patching, ERMT allows streams to merge multiple times, thereby leading to a dynamic merge tree. In particular, a new user or a newly merged group of users snoops on the closest stream that it can merge with if no later arrivals preemptively catch them [Eager et al. 1999]. The target stream may be extended to satisfy the needs of the new user, and thus its own merging target may change. 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 [Eager et al. 1999; 2001; Bar-Noy et al. 2004].

These three stream merging techniques differ in complexity and performance. Selecting the most appropriate stream merging technique depends on a trade-off between the required implementation complexity and the achieved performance. Both the implementation complexity and performance increase from patching to transition patching to ERMT.

2.3 Request Scheduling

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

The main scheduling policies include *First Come First Serve* (FCFS) [Dan et al. 1994], *Maximum Queue Length* (MQL) [Dan et al. 1994], *Maximum Factored Queue Length* (MFQL) [Aggarwal et al. 2001], and *Minimum Cost First* (MCF) [Sarhan and Qudah 2007]. These policies are described in Table I. MCF is the best overall performer when stream merging is used. It selects for service, the video whose requests require the least delivery cost. The cost is defined as the required stream length in seconds. (With stream merging, the required stream length is usually much smaller than the video length and depends on the current positions of other existing streams for the same video.) The length of the stream is directly proportional to the cost of servicing that stream because the server allocates a channel for the entire time the stream is active. *MCF-P* (P for "Per"), the preferred implementation of MCF, selects the video with the least cost per request. The costs of regular and transition streams in

Table I. Main Scheduling Policies

Scheduling Policy	Selects the Video with the
First Come First Serve (FCFS)	oldest waiting request
Maximum Queue Length (MQL)	longest waiting queue
Maximum Factored Queue Length (MFQL)	longest factored queue length
Minimum Cost First - Total (MCF-T)	least total required cost
Minimum Cost First - Per (MCF-P)	least required cost per waiting request

Table II. Cost Computations by MCF-P Implementations in Patching and Transition Patching [D_i is the Length (Duration) of Video i, s_j is the Skew From the Latest Regular Stream or Transition Stream, N_i is the Number of Waiting Requests for Video i, Wr_i is the Regular Window for Video i, Wt_i is the Transition Window for Video i]

Implementation	Regular Stream j for Video i	Transition Stream j for Video i
RAF	$F_j(i) = \frac{D_i}{N_i}$	$F_j(i) = \frac{2 \times Wt_i + s_j}{N_i}$
RAP	$F_j(i) = \frac{s_j}{N_i}$	$F_j(i) = \frac{s_j}{N_i}$

MCF-P can be computed in two ways: *Regular as Full* (RAF) and *Regular as Patch* (RAP). RAF simply uses the full lengths of regular and transition streams in determining the costs. By contrast, RAP treats these streams in a preferential manner because they are shared by later streams. It computes their costs as if they were patch streams and thus uses the skew from the last regular or transition stream, in determining the cost. Table II shows how the objective function $F_j(i)$ for regular/transition stream j for video i is computed by these two alternatives. The video with the minimum objective is to be selected. In ERMT, only RAF is applicable. A detailed treatment of MCF can be found in Sarhan and Qudah [2007].

3. PROVIDING TIME-OF-SERVICE GUARANTEES

For batching, a scheduling policy, called *Next Schedule Time First* (NSTF), was proposed in Sarhan and Das [2004] to provide time-of-service guarantees. In this article, 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. Note that the completion times of running streams (streams currently being serviced) represent when channels will become available and thus when new requests can be serviced. 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 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 Tsiolis and Vernon [1997].

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 that 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: (1) it is nonempty, (2) its assigned schedule time is worse than the freed schedule time,

and (3) 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 meets 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, called *Generalized NSTF* (GNSTF), which can be applied for batching as well as some stream merging techniques, including patching and transition patching. The server maintains a running queue (RQ), which keeps track of all currently running streams. These streams are stored in decreasing order of their completion 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 yet been assigned. $RQ[0]$ is the first element of RQ, and it corresponds to the stream with the furthest completion time. is incremented when a video is selected for service and the location of its stream completion time in RQ precedes $RQIndex$. In contrast, $RQIndex$ is decremented every time a schedule time is assigned from RQ. In addition, the server needs to maintain a pool of freed schedule times (free pool). Any assigned schedule time that is freed (due to request defection for instance) and thus can be used by later requests will be inserted into this pool. 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 (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.

For an efficient operation of GNSTF when used with stream merging, we propose and utilize two enhancements. First, GNSTF 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 significantly long 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.

Figure 1 clarifies the general operation of GNSTF. The figure shows three request waiting queues (WQs) (one for each video) and the stream running queue (RQ). The stream running queue holds information for each stream that is currently being delivered. This information contains the video number, the stream completion time, and the waiting request to which this completion time is assigned as the schedule time. Note that $RQ[0]$ corresponds to the bottom of RQ. At time T_6, request R_6 for video V_2 is made. Since the free pool is empty, this request will be assigned the completion time of the stream pointed to by $RQIndex$, and subsequently $RQIndex$ will be decremented by 1. At time T_7, request R_5 is canceled (request defection) and because it is the only request in the waiting queue, its schedule time (T_9) will become available and can be used by other requests. Request R_6 has a further schedule time and thus will be assigned this schedule time, releasing its own schedule time (T_{15}) to the free pool.

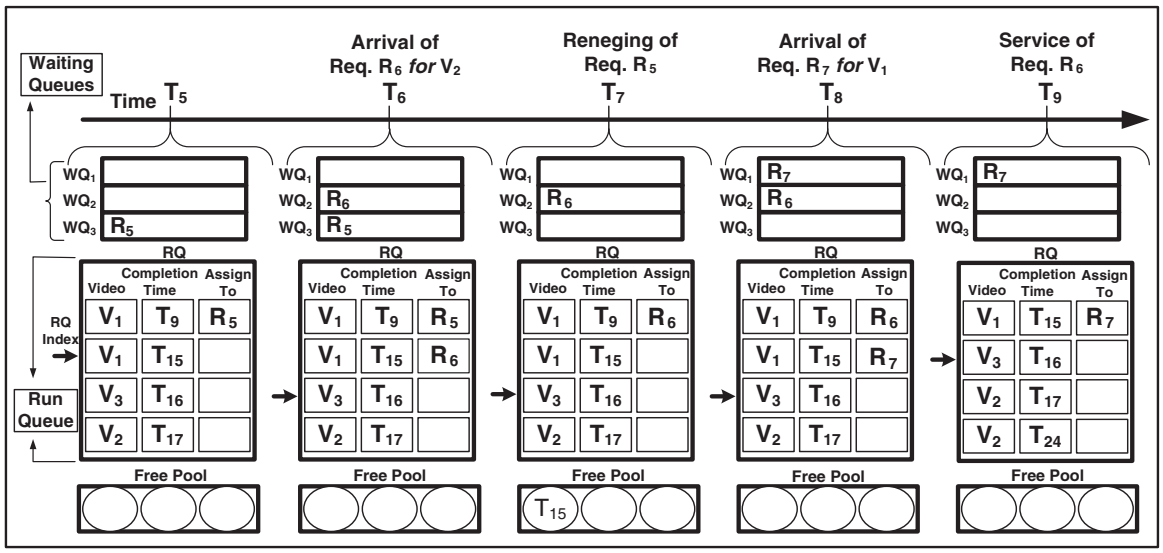


Fig. 1. Clarification of GNSTF.

Request R_7 for video V_1 is made at time T_8. Since a schedule time (T_{15}) is available in the free pool, this request will receive T_{15} as the assigned schedule time. Finally, at time T_9, R_6 is serviced, and thus a new completion time (T_{24}) becomes available and *RQIndex* will be incremented by 1.

4. PROPOSED WAITING-TIME PREDICTION APPROACH

Unfortunately, the NSTF/GNSTF 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. It is indeed hard to outperform the cost-based approach 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 an FCFS basis. Second, NSTF/GNSTF cannot work with hierarchical stream merging techniques (such as ERMT), which achieve the highest performance. NSTF/GNSTF 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 the *waiting-time prediction* approach. This approach provides users with 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 hierarchical stream merging techniques and aggressive scheduling policies (such ERMT and MCF-P, respectively) while improving user-perceived QoS by informing users of their expected waiting times. The success of the waiting-time prediction approach depends on whether the waiting times can be accurately predicted when such scheduling policies are used. If the predictions are accurate, users will appreciate the service, trust the service provider, and feel motivated to wait. Encouraging users to wait further improves the server throughput.

We present two schemes for predicting the waiting times: *Assign Expected Stream Completion Time* (AEC) and *Hybrid Prediction*. These two schemes are compared with a straightforward approach that dynamically computes the average waiting time for each video and provides the average value as the predicted waiting time for the new requests for the corresponding video. This scheme is referred to

as *Assign Per-Video Average Waiting Time* (APW). In contrast with APW, the proposed AEC scheme exhibits high intelligence. It 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. The hybrid scheme combines AEC with APW to provide an expected time to each user.

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, which is to be periodically updated, includes the current queue lengths, the completion times of running streams, and statistics, such as the average request arrival rate for each video.

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 to 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 it (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:

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

where $qlen[v]$ is the queue length of video v at the current time (T_{Now}), $\lambda[v]$ is the arrival rate for video v, and $def\_rate[v]$ is the defection rate for video v. 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. Thus, the expected queue length is adjusted by the current video defection rate. Note that the waiting tolerance distribution is generally a memoryless process. Therefore, it is not advantageous to use the current waiting times in determining when users will actually defect.

Note that the same video may be serviced again at later completion times. Equation (1) assumes that video v has not been previously identified (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:

$$expected\_qlen[v] = \lambda[v] \times (T - T_l) \times def\_rate[v]. \tag{2}$$

Note that *qlen* is not part of the equation because 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 considers the expected stream lengths required by various videos in addition to the expected queue lengths.

```
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 ( assigned_time[v] == −1) // video v has not been assigned an expected time
            expected_qlen[v] = (qlen[v] + λ[v] × (T − T_Now)) × def_rate[v];
        else // video v has been assigned an expected time
            expected_qlen[v] = λ[v] × (T − assigned_time[v]) × def_rate[v];
        Compute scheduling objective function for video v;
    } //for
    // Find the expected video to be served at time T
    expected_video = find video with the 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 this new completion time
} //while
```

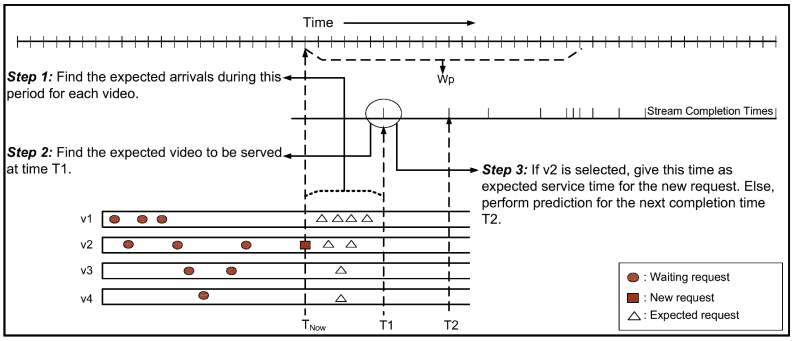
Fig. 2. Simplified algorithm for the AEC scheme performed upon the arrival of request R_i to Video v_j.

To reduce the implementation complexity in terms of algorithm computation time, AEC predicts the future scheduling decisions only during a certain duration of time, called *prediction window* (W_p). Thus, it needs to examine only the next stream completion times within W_p seconds from the arrival of the new request. Therefore, AEC may not give an expected time of service for each request. The prediction window introduces a tradeoff between the percentage of requests receiving expected times of service and prediction accuracy. Subsection 4.3 provides additional details on the implications of the value of the prediction window.

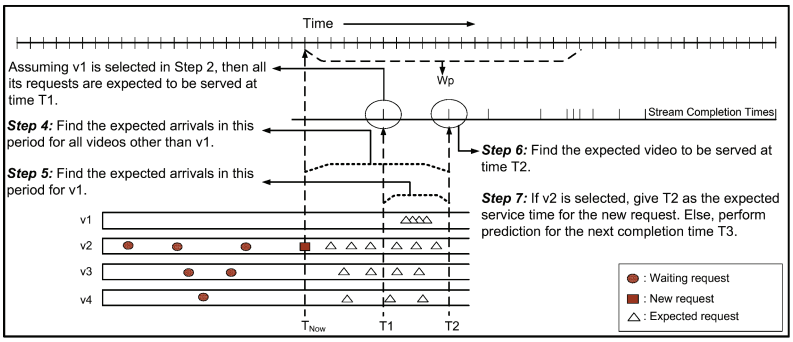
Figure 2 shows a simplified algorithm of AEC. This algorithm is performed upon the arrival of request R_i to video v_j when the server is fully loaded. If the server is not fully loaded, the request can be serviced immediately. The assigned time for video v ($assigned\_time[v]$) corresponds to the latest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

Figure 3 demonstrates the general 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. Because $v2$ 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 proposed AEC algorithm involves another important aspect: prediction of stream completion times. The server here uses aggressive prediction. It 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. 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 aspect is challenging to implement efficiently, especially with ERMT, because the impacts of these new predicted streams on stream merging decisions should be considered in order to achieve accurate predictions. To isolate



(a) Assignment of Completion Time T1



(b) Assignment of Completion Time T2

Fig. 3. Clarification of the AEC scheme.

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 on the virtual queue.

The following points serve as further clarifications of AEC. (1) The assignment of times of service is done based on predicting scheduling decisions, but the actual scheduling is kept totally isolated from prediction. Thus, scheduling is performed based on only the scheduling criterion and does not consider the assigned expected times of service in any way. (2) AEC 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 that are currently together in the waiting queue may have received different expected waiting times although they will be serviced together. Later requests in the queue are more likely to receive more accurate predictions. (3) Because of the prediction window constraint, AEC may not give an expected time for each user. As will be discussed in Subsection 4.4, the proposed hybrid scheme addresses this limitation.

4.2 Proposed Enhancements of AEC

We propose the following two enhancements of AEC: Preferential Treatment of Real Requests and Refine Assigned Expected Times.

4.2.1 *Preferential Treatment of 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 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.

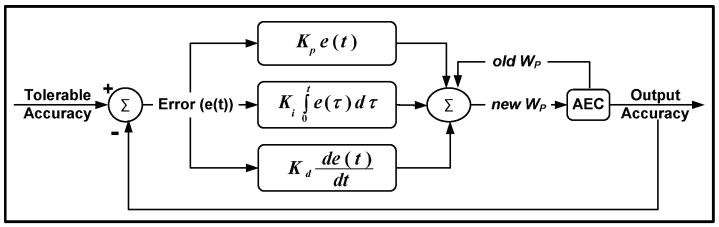
4.2.2 *Periodic Refinement of Assigned Expected Times.* With this enhancement, the server periodically attempts to provide users with updated expected times of service. For a waiting request already assigned an expected time of service, a new time of service becomes available whenever a new request for the same video arrives and receives an expected time because all requests for the same video will be serviced concurrently, using only one stream. The new expected time will most likely be more accurate than the old one because it is estimated based on the latest system state. To avoid unnecessary updates, this enhancement provides updated expected times only when a considerable difference exists between the new and old expected times. With the periodic refinement, some requests that never received expected times before may be able to get expected times later on as updates. Therefore, this enhancement is expected to improve both the prediction accuracy and the percentage of users receiving expected times. Unless otherwise indicated, the reported accuracy is determined based on the deviation of the initial expected service time and the actual time of service.

4.3 Feedback Control of the Prediction Window

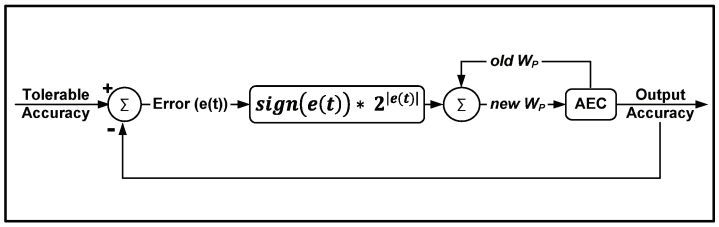
In addition to limiting the implementation complexity, the prediction window introduces a trade-off between the percentage of requests receiving expected times of service and the prediction accuracy. As the prediction window increases, the number of requests receiving expected times increases at the expense of reducing the prediction accuracy, as well as increasing the implementation complexity. The AEC algorithm computation time is proportional to the prediction window and can be shown to be $O(N_v \times W_p)$, where N_v is the number of videos. Subsection 6.4 analyzes the impact of the prediction window based on actual runs of the algorithm. Large prediction windows have negative impacts on the algorithm computation time and more importantly, the prediction accuracy. The reduced prediction accuracy may have a serious impact on the user-perceived quality of service and the confidence of users in the provided expected waiting times. The prediction window in AEC is constrained because the proposed AEC algorithm does not accurately predict the waiting times beyond a certain value of the prediction window, and it is better to provide no prediction than to provide misleading or inaccurate waiting times.

We utilize feedback control theory to tune W_p to allow a prespecified tolerance in the accuracy. Here, the administrator sets the minimum value of accuracy that can be tolerated. This value, called *setpoint*, can be specified in the form of the tolerable average deviation between the actual and expected times of service. Ideally, W_p should be set to the largest possible value that satisfies the setpoint in order to maximize the number of users receiving expected times. The loop control is driven by the *error* ($e(t)$), which is the difference between the setpoint and the actual average deviation (called the *process variable*).

For stable and accurate control, we use a *Proportional Integral Differential* (PID), as depicted in Figure 4(a), to adjust W_p based on the history and rate of change of the error. It has three components: proportional, integral, and differential. Each component is weighted by a constant. The proportional component changes W_p based on the immediate value of the error. The integral component considers the past values of the error, whereas the differential component anticipates the future, and thus they help reduce the steady state error and the overshoot, respectively.



(a) PID controller



(b) Proposed exponential controller

Fig. 4. Controllers of prediction accuracy.

To eliminate the problem of optimizing three constants in the PID Controller, we propose another controller, called *Exponential Controller*, which uses the power of the error, as shown in Figure 4(b). The power of the error is used because when the error is small, W_p should be changed slightly to minimize overshooting and undershooting. When the error however, is large, W_p should be changed by large values to expedite the convergence to the desired setpoint. By extensive analysis, we found that the best function is $sign(e(t)) \times 2^{|e(t)|}$.

4.4 Proposed Hybrid Prediction Scheme

The main problem with the AEC scheme is that not all users may receive an expected time of service. To address this problem, we propose and analyze an alternative scheme, called *Hybrid Prediction*, which can give an expected time to each user. The hybrid scheme first uses AEC to predict the waiting time, and if no prediction is made, it provides the user with the average per-video waiting time. Thus, it can be thought of as a hybrid of AEC and APW. The use of APW enables the hybrid scheme to provide a predicted waiting time for each request at the expense of lower prediction accuracy. The prediction window has an important impact on the accuracy of this hybrid predictor. As the prediction window increases, a larger fraction of users will receive expected times using AEC, which is more accurate than APW for small values of the prediction window.

5. EVALUATION METHODOLOGY

We analyze the effectiveness of the proposed schemes through extensive simulation.

5.1 Workload Characteristics

Table III summarizes the workload characteristics used. Like most prior studies, we generally assume that the arrival of the requests to the server follows a Poisson process with average arrival rate λ. We also experiment with the Weibull distribution with two parameters: shape and scale [Costa et al. 2004]. We analyze the impact of the shape (k), while adjusting the scale so that the desired average request

Table III. Summary of Workload Characteristics

Parameter	Model/Value(s)
Request Arrival	Poisson Process (Default) Weibull Distribution with shape $k = 0.6$ to 0.9
Request Arrival Rate (λ)	10 to 70 Requests/min, Default = 40 Requests/min
Server Capacity	300 to 750 channels
Video Access	Zipf-Like
Video Skew (θ)	0.1 to 0.6, Default = 0.271
Number of Videos	60 to 240, Default = 120
Video Length	Fixed-Length Video Workload (Default) with length of 30 to 120 min (same for all videos), Default = 120 min Variable-Length Video Workload 1: with lengths randomly in the range: 30 to 120 min Variable-Length Video Workload 2: with lengths randomly in the range: 100 to 200 min
Waiting Tolerance Model	A, B, and C
Waiting Tolerance Mean (μ_{tol})	15 to 90 sec, Default = 30 sec

arrival rate is reached. Additionally, we assume 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 parameter θ and normalized constant C. The parameter θ controls the skew of video access. Note that the skew reaches its peak when $\theta = 0$, and that the access becomes uniformly distributed when $\theta = 1$. We analyze the impact of this parameter, but we generally assume a value of 0.271 [Tsiolis and Vernon 1997; Sarhan and Das 2004].

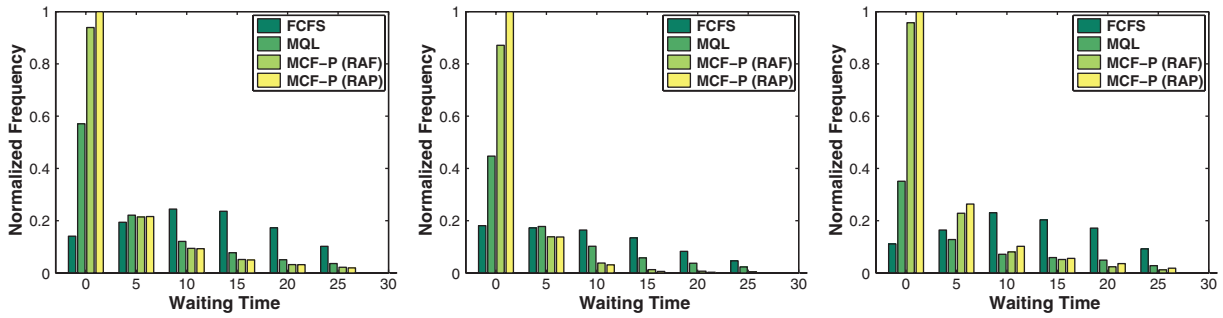
We characterize the waiting tolerance of users by three models. In *Model A*, the waiting tolerance follows an exponential distribution with mean μ_{tol} [Tsiolis and Vernon 1997; Sarhan and Das 2004]. In *Model B*, users with expected waiting times less than μ_{tol} will wait, and the others exhibit the same waiting tolerance as Model A [Tsiolis and Vernon 1997; Sarhan and Das 2004]. 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 increases linearly from 0 to 1 for the expected waiting times between μ_{tol} and $2\mu_{tol}$. In all these models, defections happen only while users are waiting for service.

We generally study a server with 120 videos, each of which is 120 minutes 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 300 to 750. In addition to the fixed-length video workload (in which all videos have the same length), we experiment with two variable-length video workloads. Moreover, we study the impacts of arrival rate, user's waiting tolerance, number of videos, and video length (in the fixed-length workload).

5.2 Performance Metrics

We use two performance metrics to compare the effectiveness of various prediction schemes: *waiting-time 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 and GNSTF approaches in terms of the user defection probability, average waiting time, and unfairness against unpopular videos.



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

Fig. 5. Waiting time distribution [*Patching, 600 Channels, Model A*].

6. RESULT PRESENTATION AND ANALYSIS

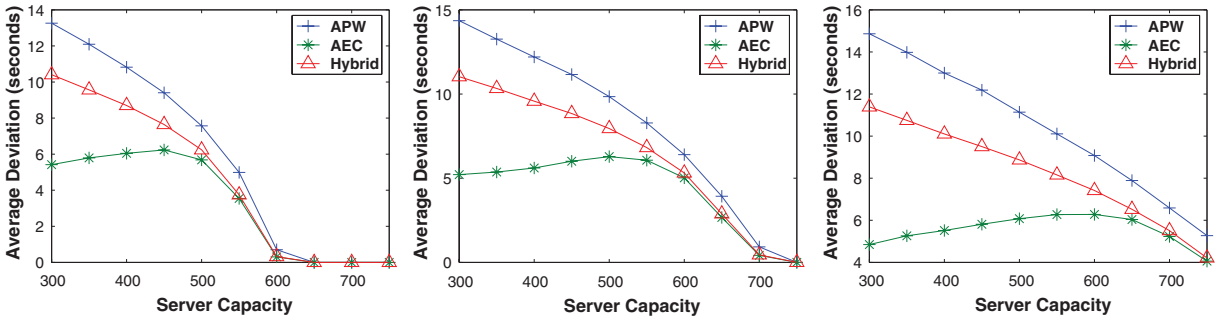
6.1 Waiting-Time Distribution under Various Scheduling Policies

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). Figure 5 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 with 600 server channels are shown. The results for transition patching and ERMT are similar, and thus not shown. 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 to 5 seconds in this example and decay quickly as we approach large values, in a manner similar to an exponential distribution. Note that the numbers vary with the stream merging technique and server capacity but with similar behavior. 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 Waiting-Time Predictability and Effectiveness of Various Prediction Schemes

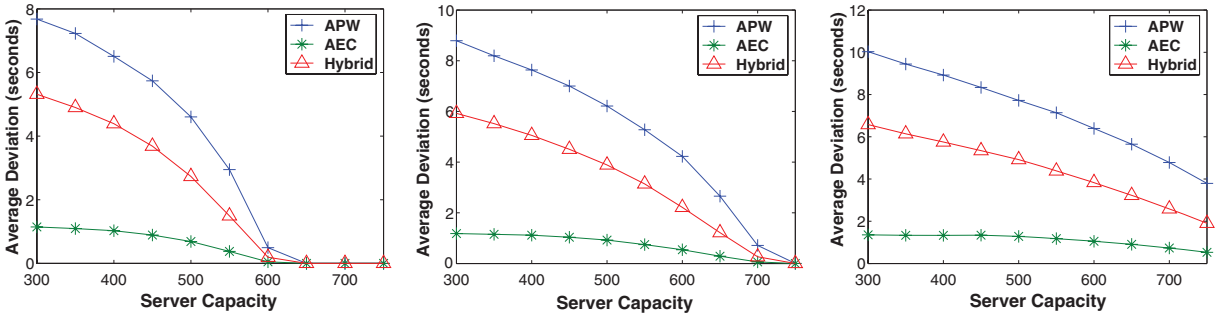
Let us now compare the effectiveness of the proposed prediction schemes in terms of accuracy, which is the most important metric in this case. Figures 6, 7, and 8 depict the average deviation results for ERMT, transition patching, and patching under three different scheduling policies: MQL, MCF-P (RAF), and MCF-P (RAP). FCFS, in the form of NSTF/GNSTF, is more suited for providing hard time-of-service guarantees than prediction. Subsection 6.8 analyzes the performance of GNSTF. These figures demonstrate that AEC performs significantly better than APW and the results are better with more scalable stream merging. The deviation with AEC is within only two seconds. APW has the advantage of giving an expected time of service to each user, but the accuracy of these expectations is a more important factor. The hybrid scheme serves as a compromise between AEC and APW.

The performance of the two variants of MCF-P (RAF and RAP) is very close 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 queue lengths in the future require prediction, as done in Equations (1) and (2). Fortunately, MCF-P is not only more predictable than MQL



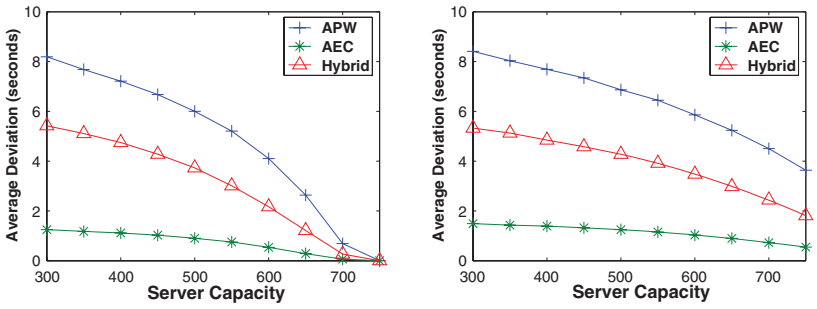
(a) ERMT (b) Transition patching (c) Patching

Fig. 6. Comparing the accuracy of prediction schemes [*MQL*, $W_p = 0.5\mu_{tol}$, *Model A*].



(a) ERMT, MCF-P (b) Transition patching, MCF-P (RAF) (c) Patching, MCF-P (RAF)

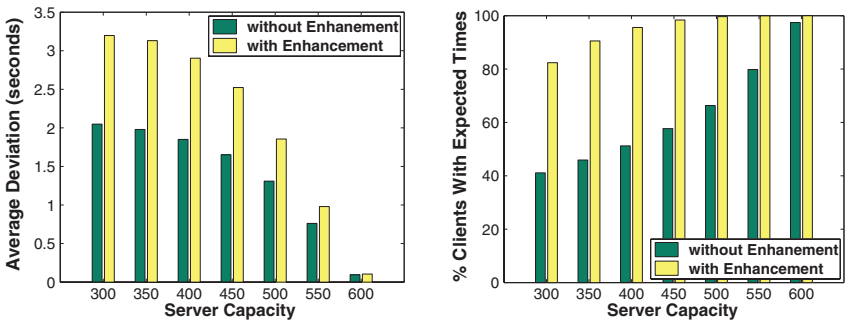
Fig. 7. Comparing the accuracy of prediction schemes [$W_p = 0.5\mu_{tol}$, *Model A*].



(a) Transition patching (b) Patching

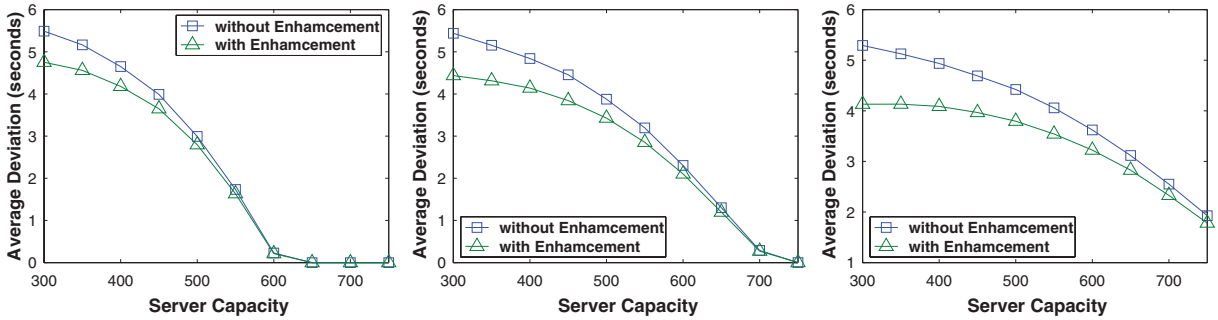
Fig. 8. Comparing the accuracy of prediction schemes [*MCF-P (RAP)*, $W_p = 0.5\mu_{tol}$, *Model A*].

but also achieves better performance (as shown in Sarhan and Qudah [2007]) in terms of defection probability, average waiting time, and unfairness. From this point on, we consider only MCF-P (RAP) and refer to it simply as MCF-P.



(a) Deviation (b) PCRE

Fig. 9. Effectiveness of the preferential treatment of real requests enhancement [*ERMT, MCF-P, AEC,* $W_p = \mu_{tol}$*, Model A*].



(a) ERMT, MCF-P (b) Transition patching, MCF-P (RAP) (c) Patching, MCF-P (RAP)

Fig. 10. Impact of the periodic refinement of assigned expected times enhancement on the hybrid prediction scheme [$W_p = \mu_{tol}$, *Model A*].

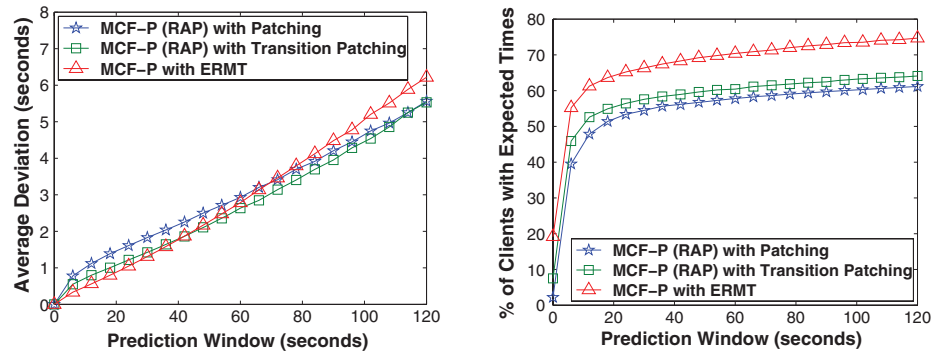
6.3 Effectiveness of Further Enhancements

The effectiveness of the Preferential Treatment of Real Requests Enhancement is illustrated in Figure 9, which shows that this enhancement significantly improves PCRE, but at the expense of accuracy. It may be a good choice in certain situations, primarily because the deviation is within 3 seconds.

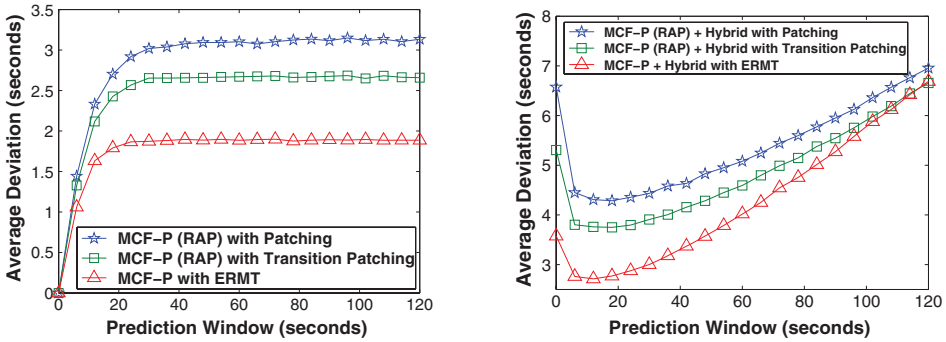
Figure 10 illustrates the effectiveness of the periodic refinement of assigned expected times enhancement in terms of the average deviation when the hybrid prediction scheme is used under the three stream merging techniques. The results are similar when AEC is used and thus not shown. The figure shows that the enhancement reduces the average deviation in patching, transition patching, and ERMT by up to 24%, 18%, and 11%, respectively.

6.4 Impact of Prediction Window

Figure 11(a) plots the impact of prediction window (W_p) in AEC on the prediction accuracy and PCRE for the three stream merging techniques. As expected, both the deviation and PCRE increase with W_p. PCRE significantly increases with W_p up to a certain point, after which it starts to increase slightly. Both these metrics generally improve with more scalable stream merging, except for large values of W_p. When the Preferential Treatment of Real Requests Enhancement is used, a significantly different behavior is observed, especially in the deviation, as shown in Figure 11(b). The deviation increases with W_p up to a certain point, after which it reaches a steady value. The impact of W_p in the case



(a) AEC without enhancement



(b) AEC with enhancement

(c) Hybrid without enhancement

Fig. 11. Impact of prediction window without and with the preferential treatment of real requests enhancement [*MCF-P, 500 Channels, Model A*].

of the hybrid prediction scheme is shown in Figure 11(c). The accuracy increases with W_p up to a certain value and then starts to decrease. (Recall that the accuracy decreases with the deviation.) The increase is due to increasing the fraction of clients receiving expected times by AEC (which is more accurate) rather than APW. After a certain value, the reduced accuracy of AEC with W_p becomes more dominant.

Figure 12 demonstrates the impact of the prediction window on the average computation time of the AEC algorithm. The results are obtained by averaging the computation time values during the entire lifetime of the simulation. The figure illustrates the increased implementation complexity with the prediction window and indicates a linear relationship.

6.5 Analysis of Deviation Distributions under Various Prediction Schemes

So far, we compared various prediction schemes in only the average deviation. In this subsection, we discuss the distributions of the deviation results, so that we can compare various schemes in range, standard deviation (σ), and confidence interval (CI). Figure 13 shows the distributions of the deviation for the three prediction schemes. The results for AEC and Hybrid are shown for two and three values of the prediction window, respectively. Table IV shows the means, standard deviations, and the 90% confidence intervals for various schemes. As expected, AEC provides the smallest standard deviation, and the shortest confidence interval, and these values increase with the prediction window. Although

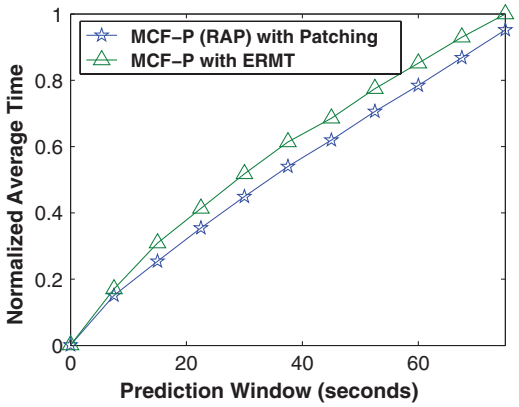
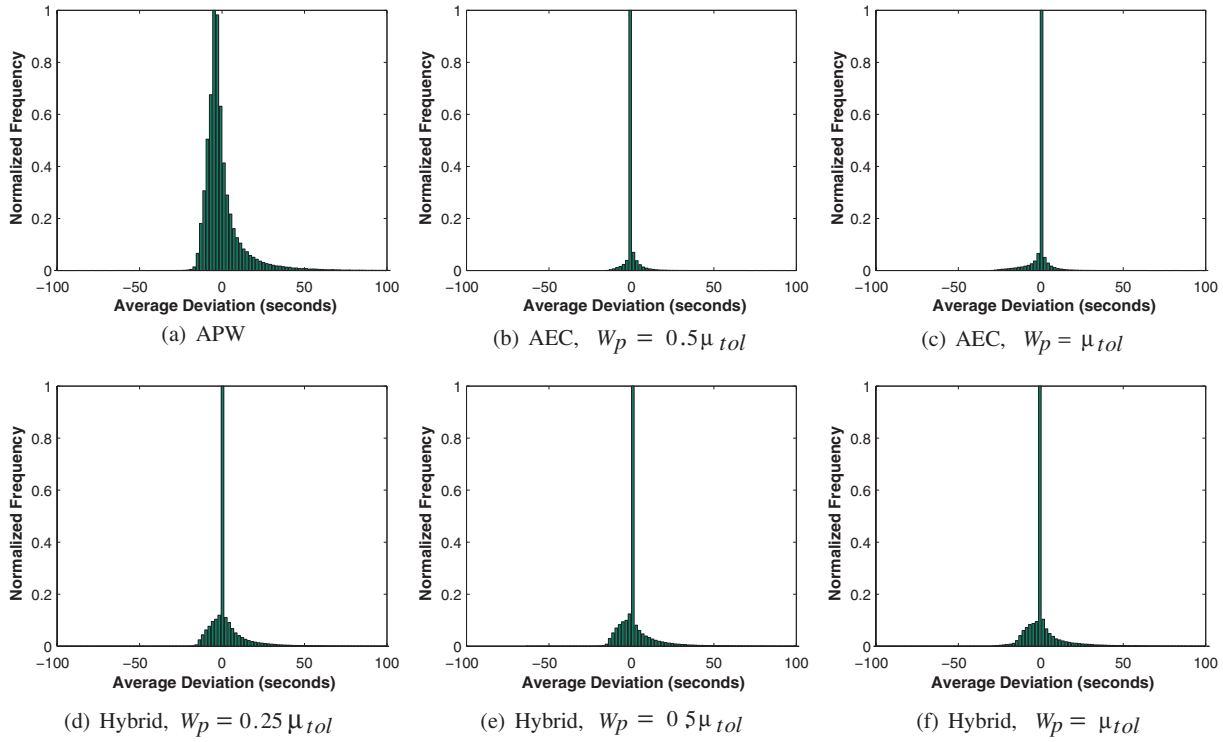


Fig. 12. Impact of prediction window on algorithm computation time [*AEC, 500 Channels, Model A*].



(a) APW (b) AEC, $W_p = 0.5\mu_{tol}$ (c) AEC, $W_p = \mu_{tol}$

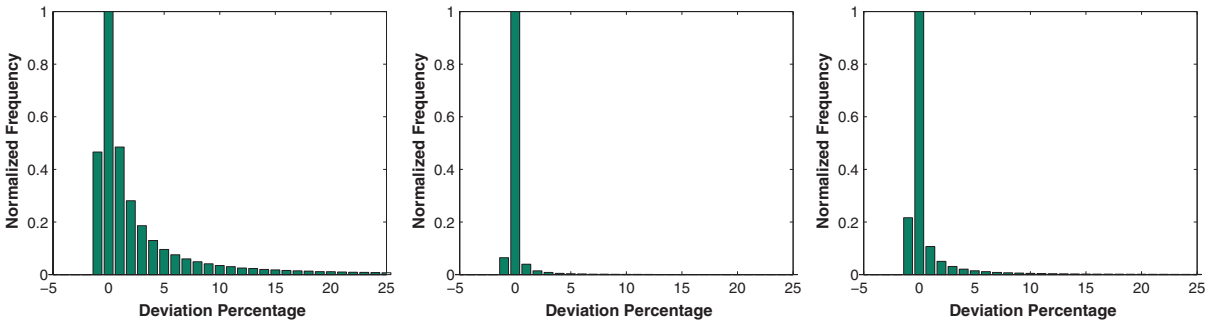
(d) Hybrid, $W_p = 0.25\,\mu_{tol}$ (e) Hybrid, $W_p = 0.5\mu_{tol}$ (f) Hybrid, $W_p = \mu_{tol}$

Fig. 13. Comparing the deviation distributions using various prediction algorithms [*ERMT, MCF-P, 300 Channels, Model A*].

the hybrid scheme performs better than APW in average accuracy (as shown in Subsection 6.2), it provides comparable results to APW in terms of standard deviation and confidence interval. Note that the means of the distribution can be positive or negative. A negative value indicates a stronger negative deviation component, whereas a positive value indicates a stronger positive deviation component. A negative deviation means that a user waits less than expected, while a positive deviation means waiting

Table IV. Summary of Deviation Distributions [*ERMT, MCF-P, 300 Channels, Model A*]

Scheme	Mean (sec)	Standard Deviation (sec)	90% Confidence Interval (sec)
APW	−0.027	12.5604	[−14.0396,14.0104]
AEC, $W_p = 0.25\mu_{tol}$	0.4537	2.6762	[−1.5707,2.4793]
AEC, $W_p = 0.5\mu_{tol}$	0.2367	3.3914	[−3.8063,4.2437]
AEC, $W_p = \mu_{tol}$	−0.6131	5.2066	[−7.8732,6.6768]
Hybrid, $W_p = 0.25\mu_{tol}$	2.4055	11.7448	[−11.7366,16.5134]
Hybrid, $W_p = 0.5\mu_{tol}$	2.0781	11.6507	[−11.8441,16.0059]
Hybrid, $W_p = \mu_{tol}$	1.3027	11.8306	[−13.5439,16.1061]



(a) APW (b) AEC (c) Hybrid

Fig. 14. Comparing the distributions of the deviation percentage [*ERMT, MCF-P,* $W_p = 0.5\mu_{tol}$*, 300 Channels, Model A*].

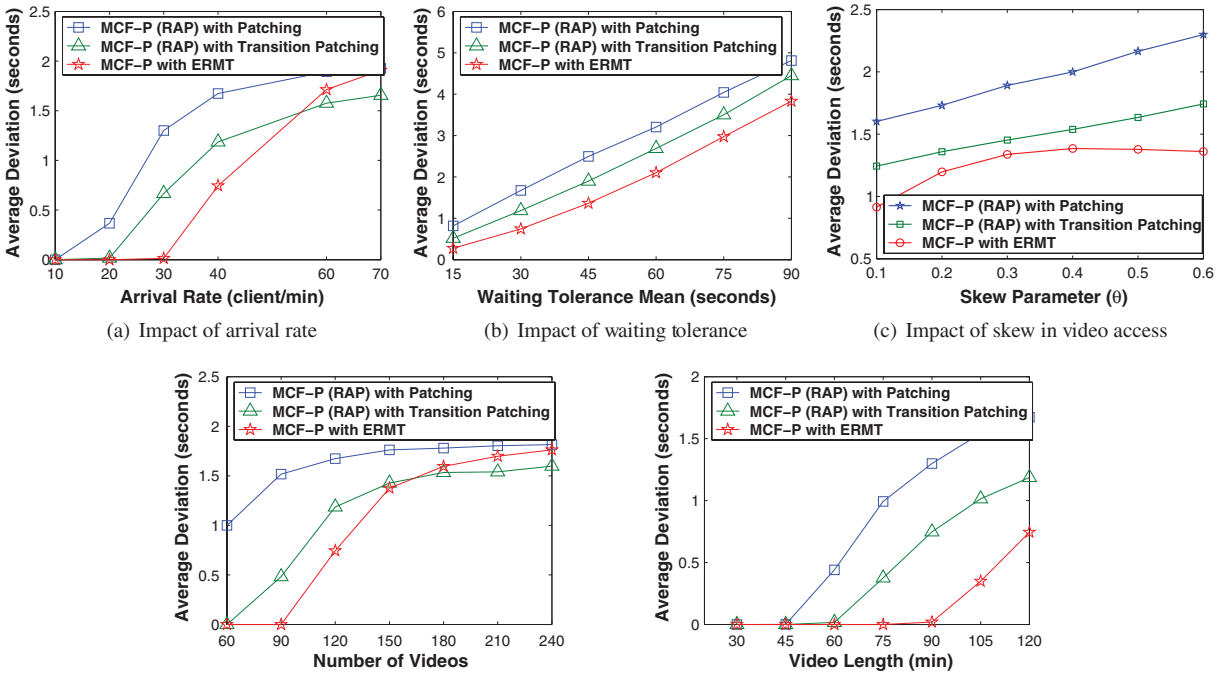
longer than expected. It is possible to assign different weights for negative and positive deviations, but in this article we treat them equally. Accurate waiting times help users wait accordingly, so it may not be beneficial if the user waits less than expected because the user may be doing something else meanwhile. Finally, it is useful to study the relative deviation compared to the actual waiting time. Figure 14 compares the distributions of the deviation percentage of the three prediction algorithms. These results illustrate that the benefits of AEC, especially compared to APW, are more outstanding in terms of the percentage deviation.

6.6 Impact of Workload Parameters on AEC Performance

Figures 15 and 16 show the impacts of request arrival rate, user's waiting tolerance, skew in video access (θ), number of videos, and video length on the effectiveness of AEC in terms of accuracy and PCRE, respectively. The deviation increases with the arrival rate but remains with 2 seconds even for up to 70 requests/minute. The deviation also 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 users these days are getting much higher and their waiting tolerance is getting lower. PCRE decreases with the arrival rate but does not change much with the waiting tolerance.

The skew in video access has significant impacts on the average deviation and PCRE. Recall that as θ increases, the skew in video access decreases. Both the prediction accuracy and PCRE are worsened by the reduction in the skew. This is due to the reduced predictability of which video can be serviced at any particular time as the video access approaches the uniform distribution.

Finally, both the accuracy and PCRE also decrease with the number of videos and video length, primarily due to the increased load on the server, as can be noted by the increase in the user defection rate (not shown for space limitation). This behavior is consistent with the results in Subsection 6.2.



(a) Impact of arrival rate (b) Impact of waiting tolerance (c) Impact of skew in video access

(d) Impact of number of videos (e) Impact of video length

Fig. 15. Impact of workload on average deviation [*AEC, 550 Channels,* $W_p = \mu_{tol}$*, Model A*].

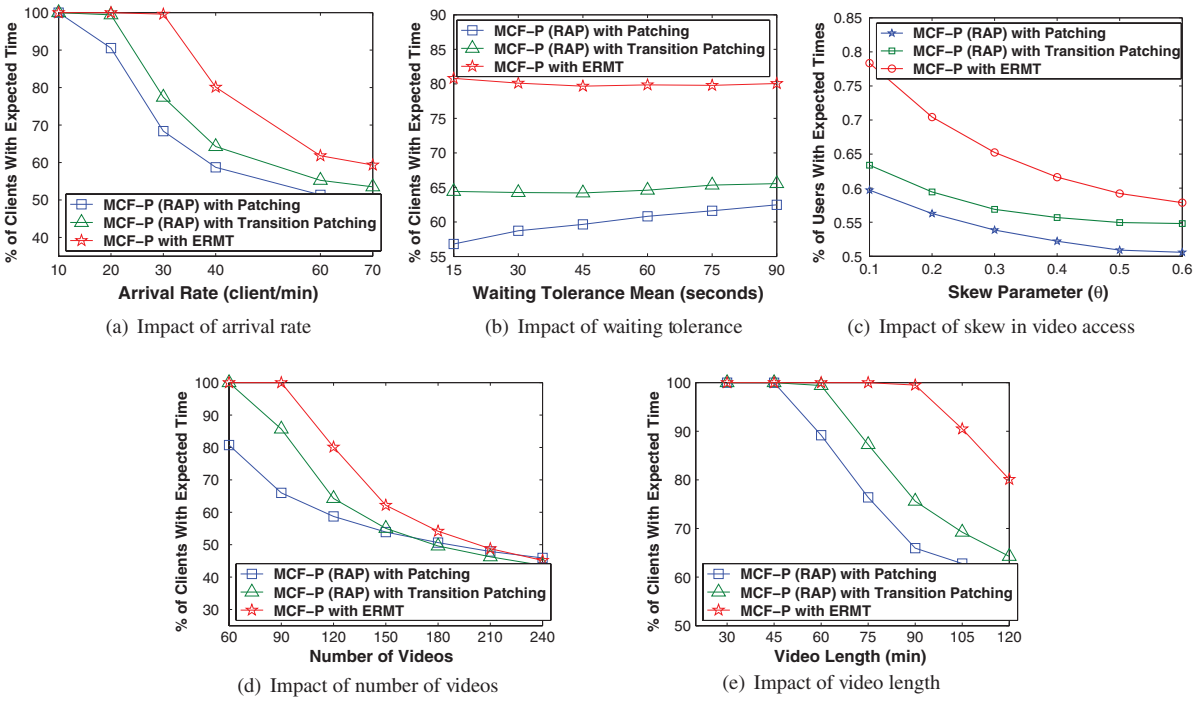
The increase in the server load as the number of videos increases happens as a result of the reduction in data sharing. Although the deviation increases with the number of videos, it remains with 2 seconds even for up to 240 supported videos.

The results so far are for a video workload of a fixed video length. Figure 17 shows the average deviation and PCRE results for two different variable-length workloads. The first is comprised of videos with lengths in the range of 30 to 120 minutes, whereas the lengths in the second range from 100 to 200 minutes. The length of each video is generated randomly within the specified range. The results for each workload are obtained by averaging the values of three runs. The AEC algorithm also works well in these workloads, with an average deviation within only 2.5 seconds. For workloads with longer videos, the server load becomes larger (as indicated in Figure 17(c)), and thus both the average deviation and PCRE become worse.

The results so far assume a Poisson request arrival process. Let us now examine the behavior with a Weibull distribution with different shape (k) values. Figure 18 demonstrates that the waiting times can still be predicted accurately with the AEC algorithm. The shape has a little impact, especially when the prediction window is smaller than 35 seconds.

6.7 Feedback Control of the Prediction Window in AEC

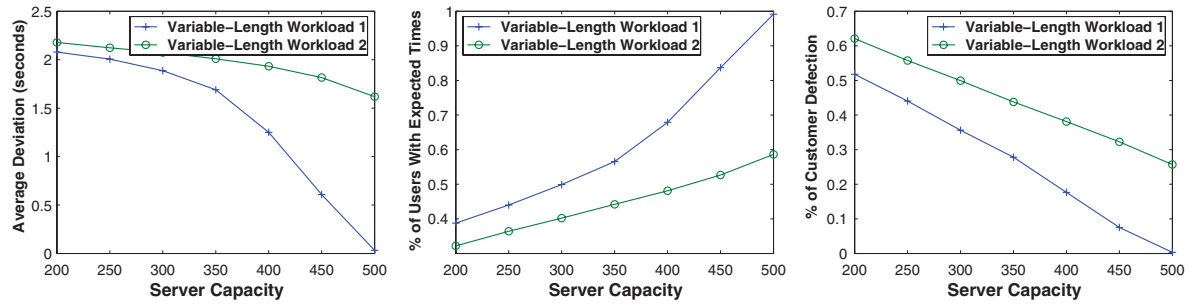
Let us now discuss the effectiveness of using the proposed controller of the prediction window, when the AEC scheme is used. Figure 19 demonstrates how the PID Controller can effectively achieve five desired accuracy values (setpoints): 0.5, 1.5, 2.5, 3.5, and 4.5 seconds with different stream merging techniques. By dynamically tuning W_p to the largest possible value that satisfies the setpoint, the PID



(a) Impact of arrival rate (b) Impact of waiting tolerance (c) Impact of skew in video access

(d) Impact of number of videos (e) Impact of video length

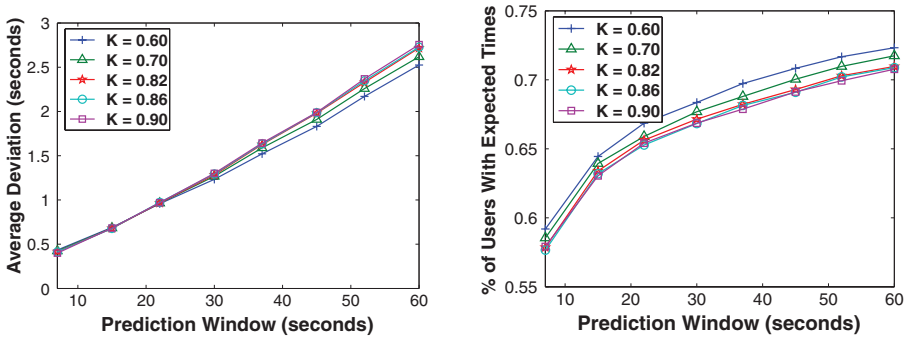
Fig. 16. Impact of workload on PCRE [*AEC, 550 Channels,* $W_p = \mu_{tol}$*, Model A*].



(a) Average deviation (b) PCRE (c) User defection rate

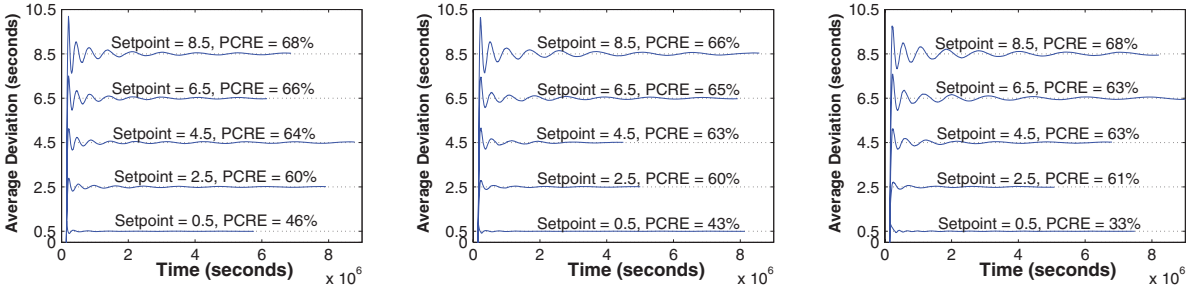
Fig. 17. Impact of variable-length video workloads [*AEC, ERMT, MCF-P,* $W_p = \mu_{tol}$*, Model A*].

Controller ensures the largest possible value of PCRE. As the tolerable accuracy increases from 0.5 to 4.5, PCRE increases by more than 39%, 46%, and 93% with ERMT, Transition Patching, and Patching respectively. The system administrator should consider this significant implication of the setpoint on PCRE. Figure 20 shows how that the Exponential Controller behaves close to the PID Controller. The PID Controller has a little advantage over the Exponential for small setpoints, where it achieves larger PCRE because it has a slightly larger overshoot and is a little faster in reaching the desired setpoint.



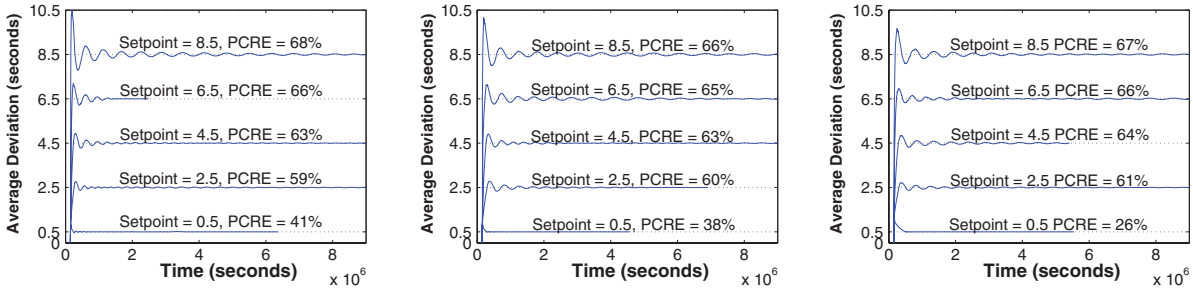
(a) Average deviation (b) PCRE

Fig. 18. Impact of the shape parameter for a Weibull Arrival Distribution [*AEC, ERMT, MCF-P,* $W_p = \mu_{tol}$*, 500 Channels, Model A*].



(a) *ERMT*, MCF-P, 450 channels (b) *Transition Patching*, MCF-P (RAP), 500 channels (c) *Patching*, MCF-P (RAP), 550 channels

Fig. 19. Effectiveness of PID controller using different average deviation setpoints [*AEC, Model A,* $K_p = 7, K_i = 1, K_d = 0.1$].

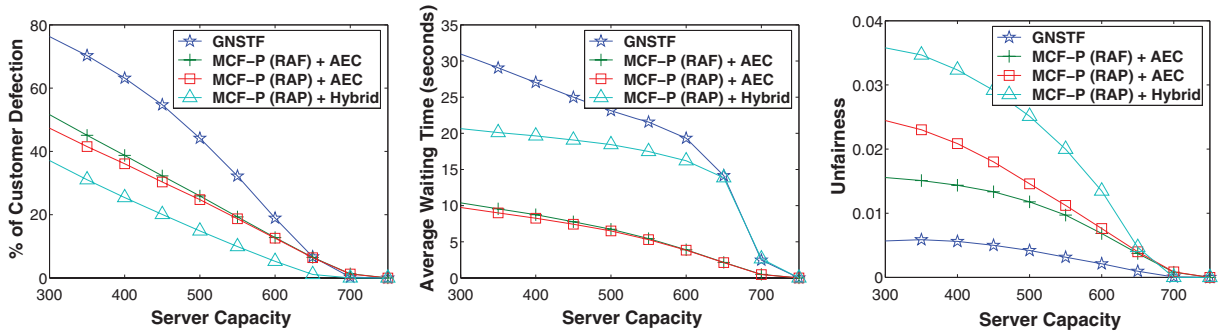


(a) *ERMT*, MCF-P, 450 channels (b) *Transition Patching*, MCF-P (RAP), 500 channels (c) *Patching*, MCF-P (RAP), 550 channels

Fig. 20. Effectiveness of exponential controller using different average deviation setpoints [*AEC, Model A*].

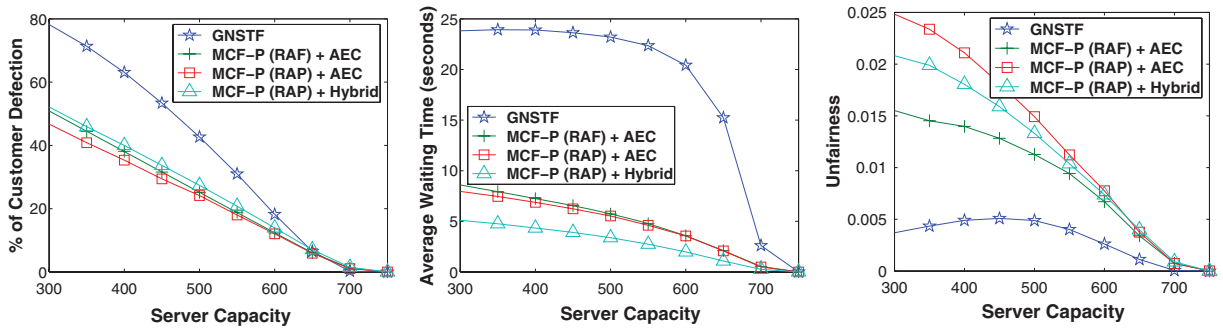
6.8 Effectiveness of the Waiting-Time Prediction Approach Compared with GNSTF

As discussed earlier, GNSTF is a scheduling policy that performs request scheduling based on the schedule times, which are initially assigned on an FCFS basis. The proposed waiting-time prediction approach allows the application of aggressive cost-based scheduling policies and hierarchical stream merging techniques (such as MCF-P and ERMT, respectively). In this subsection, we demonstrate the



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

Fig. 21. Comparing GNSTF with predictive MCF-P (Three Variants) [*Transition Patching*, $W_p = 0.5\mu_{tol}$, *Model B*].

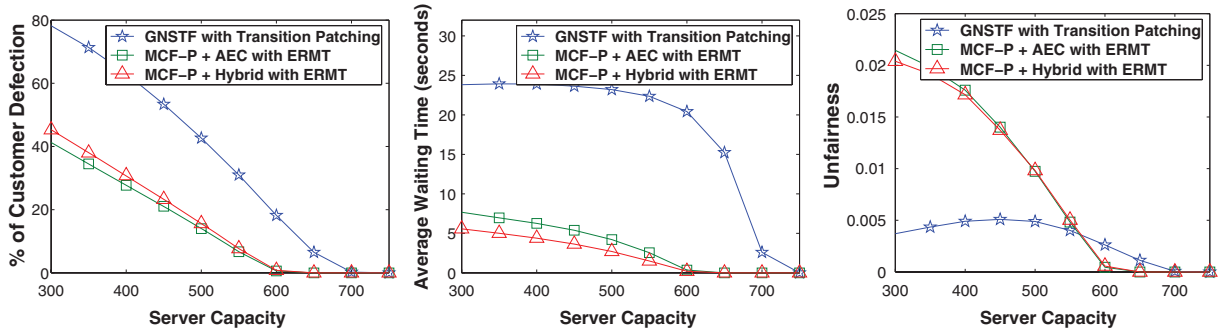


(a) Defection rate (b) Waiting time (c) Unfairness

Fig. 22. Comparing GNSTF with predictive MCF-P (Three Variants) [*Transition Patching*, $W_p = 0.5\mu_{tol}$, *Model C*].

implications of the predictive approach on improving system performance in terms of the overall user defection rate and average waiting time. The prediction approach here is applied with MCF-P and this combination is referred to as "Predictive MCF-P." Figures 21 and 22 compare the two approaches in user defection probability, average waiting time, and unfairness for Models B and C of the waiting tolerance, respectively. Three variants of predictive MCF-P are analyzed. The first two apply the AEC prediction scheme, whereas the third applies the hybrid. The best implementation of GNSTF (GNSTFo-MQL) is used to ensure a fair comparison. These results demonstrate that predictive MCF-P performs significantly better than GNSTF under both tolerance models in terms of the two most important performance metrics. The relative performance among the different predictive MCF-P variants depends on the tolerance model. With model B, the hybrid scheme leads to the minimum defection rate and the longest waiting time among various variants. With model C, however, it leads to the shortest waiting time and the highest defection rate. With both models, the RAP and RAF variants performance is very close in the two most important metrics.

Finally, Figure 23 captures the fact that GNSTF cannot be applied with ERMT, whereas the waiting-time prediction approach can. The figure compares GNSTF and two variants of predictive MCF-P (AEC and Hybrid), when each is applied with the most scalable stream merging technique that is applicable to it. Here, only the results with model C (which is more realistic) are shown. Model B exhibits a similar behavior. The results demonstrate the outstanding performance gains achieved by applying the



(a) Defection rate (b) Waiting time (c) Unfairness

Fig. 23. Comparing GNSTF and predictive MCF-P (Two Variants), each with its most scalable stream merging technique [$W_p = \mu_{tol}$, *Model C*].

prediction approach in terms of the two most important performance metrics. The performance of the two variants is generally close.

7. CONCLUSIONS

We have analyzed the waiting-time predictability in scalable video streaming and have presented two prediction schemes: *Assign Expected Stream Completion Time* (AEC) and *Hybrid Prediction*. 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 tradeoff between the prediction accuracy and the number of users receiving expected waiting times. The hybrid scheme uses AEC and then assigns the average video waiting time for those requests that did not obtain a predicted time by AEC.

We have analyzed the effectiveness of the two prediction schemes when applied with various stream merging techniques and scheduling policies. We have also compared the effectiveness of the waiting-time prediction approach with the approach that provides time-of-service guarantees. The latter is represented by an extended policy, called *Generalized Next Schedule Time First* (GNSTF). In addition, we have studied the impacts of prediction window, server capacity, user's waiting tolerance, arrival rate, skew in video access, video length, and number of videos.

The main results can be summarized as follows. (1) 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. (2) The prediction accuracy in AEC can be controlled efficiently using a PID, or the proposed exponential controller. The administrator sets a value for the maximum tolerable accuracy (setpoint), and then the controller changes the prediction window dynamically to reach the largest value that satisfies the setpoint, to maximize the number of users receiving expected waiting times. The exponential controller removes the complexity in efficiently tuning three parameters in the PID Controller. (3) In contrast with AEC, the hybrid prediction scheme provides expected times to each user but achieves lower accuracy and a longer confidence interval. (4) Combining AEC or the hybrid scheme with MCF-P leads to outstanding performance benefits, compared with GNSTF. (5) *Predictive MCF-P*, can be applied with hierarchical stream merging techniques (such as ERMT) to improve performance further, whereas GNSTF cannot.

REFERENCES

AGGARWAL, C. C., WOLF, J. L., AND YU, P. S. 2001. The maximum factor queue length batching scheme for Video-on-Demand systems. *IEEE Trans. Comput. 50*, 2, 97–110.

AL-HADRUSI, M. AND SARHAN, N. J. 2008. A scalable delivery framework and a pricing model for streaming media with advertisements. In *Proceedings of the SPIE/ACM Multimedia Computing and Networking Conference (MMCN)*.

ALSMIRAT, M., AL-HADRUSI, M., AND SARHAN, N. J. 2007. Analysis of waiting-time predictability in scalable media streaming. In *Proceedings of the ACM Multimedia*. 791 – 794.

BAR-NOY, A., GOSHI, G., LADNER, R., AND TAM, K. 2004. Comparison of stream merging algorithms for Media-on-Demand. *Multimedia Syst. J. 9*, 211–223.

CAI, Y. AND HUA, K. A. 1999. An efficient bandwidth-sharing technique for true video on demand systems. In *Proceedings of the ACM Multimedia*. 211–214.

CAI, Y. AND HUA, K. A. 2003. Sharing multicast videos using patching streams. *Multimedia Tools Appl. J. 21*, 2 (Nov.), 125–146.

CAI, Y., TAVANAPONG, W., AND HUA, K. A. 2003. Enhancing patching performance through double patching. In *Proceedings of the 9th International Conference on Distributed Multimedia Systems*. 72–77.

CARTER, S. W. AND LONG, D. D. E. 1997. Improving Video-on-Demand server efficiency through stream tapping. In *Proceedings of the International Conference on Computer Communication and Networks (ICCCN)*. 200–207.

COSTA, C., CUNHA, I., BORGES, A., RAMOS, C., ROCHA, M., ALMEIDA, J., AND RIBEIRO-NETO, B. 2004. Analyzing client interactivity in streaming media. In *Proceedings of the The World Wide Web Conference*. 534–543.

DAN, A., SITARAM, D., AND SHAHABUDDIN, P. 1994. Scheduling policies for an on-demand video server with batching. In *Proceedings of the ACM Multimedia*. 391–398.

EAGER, D. L., VERNON, M. K., AND ZAHORJAN, J. 1999. Optimal and efficient merging schedules for Video-on-Demand servers. In *Proceedings of the ACM Multimedia*. 199–202.

EAGER, D. L., VERNON, M. K., AND ZAHORJAN, J. 2000. Bandwidth skimming: A technique for cost-effective Video-on-Demand. In *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*. 206–215.

EAGER, D. L., VERNON, M. K., AND ZAHORJAN, J. 2001. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. Knowl. Data Eng. 13*, 5 (Sept.), 742–757.

HUA, K. A., CAI, Y., AND SHEU, S. 1998. Patching: A multicast technique for true Video-on-Demand services. In *Proceedings of the ACM Multimedia*. 191–200.

HUA, K. A. AND SHEU, S. 1997. Skyscraper broadcasting: A new broadcasting scheme for metropolitan Video-on-Demand system. In *Proceedings of the ACM SIGCOMM*. 89–100.

HUANG, C., JANAKIRAMAN, R., AND XU, L. 2004. Loss-resilient on-demand media streaming using priority encoding. In *Proceedings of the ACM Multimedia*. 152–159.

JUHN, L. AND TSENG, L. 1997. Harmonic broadcasting for Video-on-Demand service. *IEEE Trans. Broadcast. 43*, 3, 268–271.

MA, H., SHIN, G. K., AND WU, W. 2005. Best-effort patching for multicast true VoD service. *Multimedia Tools Appl. 26*, 1, 101–122.

PÂRIS, J.-F., CARTER, S. W., AND LONG, D. D. E. 1998. Efficient broadcasting protocols for video on demand. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 127–132.

QUDAH, B. AND SARHAN, N. J. 2006. Towards scalable delivery of video streams to heterogeneous receivers. In *Proceedings of the ACM Multimedia*. 347–356.

ROCHA, M., MAIA, M., CUNHA, I., ALMEIDA, J., AND CAMPOS, S. 2005. Scalable media streaming to interactive users. In *Proceedings of the ACM Multimedia*. 966–975.

SARHAN, N. J. AND DAS, C. R. 2004. A new class of scheduling policies for providing time of service guarantees in Video-On-Demand servers. In *Proceedings of the the 7th IFIP/IEEE International Conference on Management of Multimedia Networks and Services*. 127–139.

SARHAN, N. J. AND QUDAH, B. 2007. Efficient cost-based scheduling for scalable media streaming. In *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*.

SHI, L., SESSINI, P., MAHANTI, A., LI, Z., AND EAGER, D. L. 2006. Scalable streaming for heterogeneous clients. In *Proceedings of ACM Multimedia*. 337–346.

TSIOLIS, A. K. AND VERNON., M. K. 1997. Group-guaranteed channel capacity in multimedia storage servers. In *Proceedings of the ACM SIGMETRICS*. 285–297.

Received September 2008; revised February 2009; accepted April 2009