A Scalable Solution for Interactive Near Video-on-Demand Systems

Kamal K. Nayfeh and Nabil J. Sarhan, *Member, IEEE*

***Abstract*— The required real-time and high-rate transfer of multimedia data limits the numbers of requests that can be concurrently serviced by video-on-demand (VOD) systems. Resource-sharing techniques can be used to address this scalability challenge, but they greatly complicate the efficient support for interactive operations. We develop an overall solution for interactive near VOD systems that employ resource sharing. The proposed solution supports user interactions with short response times and low rejection probabilities. The solution includes a novel stream provisioning policy, which dynamically determines the best number of I-Streams (unicast streams for supporting interactive requests) and the maximum I-Stream length that can be allocated by the server. Furthermore, we use a sophisticated client-side cache management policy to maximize the percentage of interactive requests serviced from the client's own cache. We study the system using realistic workload through extensive simulation.**

***Index Terms*— Interactive requests, multicast, near video on demand (NVOD), realistic workload, stream merging, video streaming, VOD.**

I. Introduction

DUE to the remarkable growth of online video and social media, video streaming has grown drastically. Real-time entertainment, including video on demand (VOD), is the largest traffic category on virtually every network and is expected to continue growing [17]. In true VOD (TVOD), all requests are serviced immediately, but this stringent requirement is hard to meet. Near VOD (NVOD) is the general case because it converges to TVOD when resources become sufficiently high. This paper considers the design of NVOD systems.

The distribution of streaming media faces a significant scalability challenge due to the high server and network requirements. Hence, many resource-sharing techniques [5], [10], [12], [15], [16] have been proposed to utilize the multicast facility. In particular, stream merging techniques, such as patching [12] (and the references within) and earliest reachable merge target (ERMT) [10], aggregate users into larger groups that share the same multicast streams. Unfortunately, the overwhelming majority of prior studies used simple workload, in which media objects are homogeneous in type, length, and bitrate, and are sequentially accessed from the beginning to the end without interactions (such as pause, resume, jump backward, and jump forward). More recent characterization studies [3], [6], [7], [9], however, reveal that the actual workload is more complex and dynamic. In particular, the users issue many interactive operations and may access objects from points other than the beginning and may stop before the end. In addition, the user behavior varies with media type, content type, and object length. Only few studies have considered the design of interactive VOD systems, but they assumed that all requests are immediately serviced [16] or did not use stream merging [11], [14].

We study the design of interactive NVOD systems that utilize stream merging for scalable delivery. The proposed solution enhances customer-perceived quality of service (QoS) by servicing requests quickly and ensuring pleasant and smooth playbacks. It also supports user interactions and realistic access patterns with short response times and low rejection probabilities. We generalize the split and merge (SAM) protocol [11] to work with stream merging techniques, such as patching and ERMT. The solution employs a variety of stream types, including B-Streams (full-length multicast streams), P-Streams (multicast streams for supporting patching), and I-Streams (unicast streams for supporting interactive requests). The solution includes a novel I-Stream provisioning policy. This policy determines the optimal I-Stream threshold, which is the maximum I-Stream length (in seconds) allowed to be allocated by the server. It then adjusts the number of I-Streams and B-Streams, according to the current system requirement. By restricting the I-Stream threshold, this policy prevents any request from using an I-Stream for an extended period, thereby allowing other requests to be serviced as soon as possible. Moreover, we propose a modified request scheduling policy to address the unfairness issue in servicing unpopular videos. Furthermore, we use a client-side cache management policy to maximize the percentage of interactive requests serviced from the client's own cache, thereby reducing the required server bandwidth.

We evaluate through extensive simulation the effectiveness of the proposed solution under realistic workload and using different resource-sharing techniques and scheduling policies. We study the effect of a wide range of system parameters on the clients' waiting and blocking metrics. Furthermore, we develop a new metric, called aggregate delay, to quantify the average delay experienced by a client due to both waiting and blocking during a streaming session. This metric captures

Manuscript received January 4, 2015; revised April 9, 2015 and June 23, 2015; accepted August 29, 2015. Date of publication September 14, 2015; date of current version September 30, 2016. This paper was recommended by Associate Editor L. Zhou.

The authors are with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202 USA (e-mail: knayfeh@wayne.edu; nabil@wayne.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2015.2478708

the increased customer frustration with subsequent blocking, especially if it occurs within a short time from the last blocking.

The rest of this paper is organized as follows. Section II discusses the background information and related work. Section III presents the proposed solution. Section IV discusses the performance evaluation methodology. Finally, Section V presents and analyzes the main results.

II. Background Information and Related Work

This paper considers the multicast-based approach. The other primary approaches used to address the scalability challenge in delivering multimedia streams are content distribution networks (CDNs) and peer-to-peer (P2P) [19] (and the references within). Whereas the CDN approach requires maintaining a huge number of geographically distributed servers [13], the P2P approach still heavily relies on central servers [2], [4]. Both of these approaches mitigate the scalability problem but do not eliminate it because the fundamental problem is due to unicast delivery [2]. As multicast is highly effective in delivering high-usage content and in handling flash crowds, there has been a growing interest in enabling native multicast to support high-quality on-demand video distribution, Internet Protocol Television, and other major applications [2].

The main performance metrics of VOD systems include the overall client defection probability, average waiting time, and unfairness. The defection probability is the likelihood that a user leaves the server without being serviced because the waiting time exceeded the user's tolerance. It is the most important metric because it translates to the number of concurrently serviced customers and server throughput. The second and third metrics are indicators of the perceived QoS. As it is desirable that the server treats the requests for different videos equally, unfairness measures the bias of a policy against unpopular videos and is equal to $(\sum_{i=1}^{M}(d_i - d)^2/(M-1))^{1/2}$, where M is the number of videos, d_i is the defection rate of video i, and d is the mean video defection rate.

Next, we discuss the three main design aspects of VOD systems.

A. Resource Sharing

The main resource-sharing techniques include batching [1], patching [12], and ERMT [10]. Batching simply services all waiting requests for a video using one full-length multicast stream. In contrast, patching dynamically expands the multicast tree to include new requests. A new request joins the latest full-length stream for the same video and receives the missing portion as a patch. When the playback of the patch is completed, the client continues the playback of the remaining portion using the data received from the full-length stream and buffered locally. When the length of the patch stream exceeds a certain dynamic threshold, delivering a full-length multicast stream becomes more cost effective than delivering the patch. Whereas patching allows a stream to merge only once, ERMT allows streams to merge multiple times, resulting in a hierarchical stream merging tree. A new client or a newly merged group of clients listens to the closest stream it can merge with if no later arrivals can preemptively catch them. The target stream can later be extended to satisfy the needs of the new client (only after the merging stream finishes and merges with the target), and this extension can affect its own merge target.

B. Request Scheduling

The server maintains a queue for every video and applies a scheduling policy to select an appropriate queue for service when server resources become available. All requests for a certain video can be serviced using one channel, which can be defined as the set of required server resources (network bandwidth, disk I/O bandwidth, etc.) for delivering a multimedia stream. The most popular policies are first come first serve (FCFS) [8], maximum queue length (MQL) [8], and maximum cost first (MCF) [18]. FCFS selects the queue with the oldest request, and thus it is the fairest policy. MQL, however, tries to maximize the number of requests serviced with one channel by selecting the queue with the largest number of requests. In contrast, MCF selects the queue with the minimum cost in terms of the stream length. MCF-P (P for Per), the preferred implementation of MCF, selects the video with the least cost per request. Reference [18] shows that MCF-P outperforms other scheduling policies when stream merging is used.

C. Support for Interactive Operations

Interactive requests can be supported using dedicated unicast streams, called I-Streams [11], [14]. The server utilizes an I-Stream when it cannot service the interactive request by any other way. I-Streams can start from any playback point in the video and can last as long as the customer needs it. Since I-Streams are unicast, they cannot be shared with other requests. The SAM protocol [11] was proposed to service interactive requests when using batching. As soon as a client issues an interactive request, the client is split from the multicast stream, and temporarily assigned an I-Stream to perform the interaction. Once the interaction is complete, the client is merged into an ongoing stream. For a pause interaction, no I-Stream is required, and the user is merged into an ongoing multicast stream as soon as the user resumes.

III. Proposed Solution

A. Overall System Design

As shown in Fig. 1, the proposed system consists of a VOD server, clients streaming videos from the server, and a network connecting them. The major components of the VOD server are waiting queues (with one queue being for each video), a blocking queue, a queuing manager, an I-Stream provisioning module, an enhanced SAM protocol, a resource-sharing protocol, a waiting request scheduling policy, a blocking request scheduling policy, and streams. The streams are divided into full-length multicast streams (B-Streams), multicast patch streams (P-Streams), and unicast interactive streams (I-Streams). B-Streams and P-Streams are used to service waiting customers and can be shared with

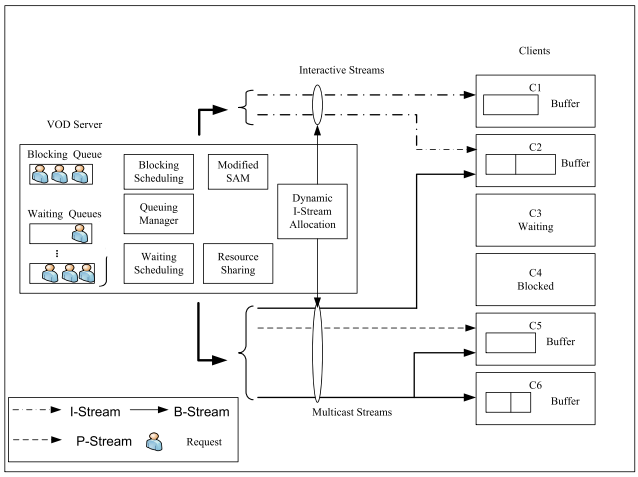


Fig. 1. Proposed system design.

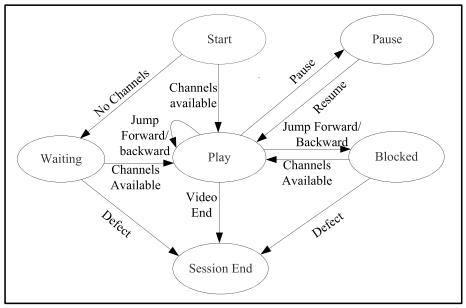


Fig. 2. Simplified state transition diagram.

other requests for only the same video. An I-Stream is a dedicated unicast stream for servicing interactive requests and thus cannot be shared with other requests. The server utilizes the SAM technique to manage I-Streams. The I-Stream is freed once it finishes delivering the requested data.

The customer starts a streaming session by issuing a request to playback a certain video. The server makes the decision whether it can service the request or not based on the number of available server channels. If there are adequate server channels, the server starts streaming the video to the customer using a resource-sharing technique (batching, patching, or ERMT). Otherwise, the customer's request is placed in the waiting queue for that video. The server applies a waiting scheduling policy (FCFS, MQL, or MCF) to determine which waiting queue to service when streams become available. Waiting customers defect if they stay in the waiting queue for too long. During a streaming session, the customer issues interactive requests to pause the video, resume the video, jump forward, and/or jump backward. The last two are by a certain distance relative to the current playback position. The customer continues to cache from all listening streams during a pause until the cache is full. Since pause and resume requests do not require any additional resources, they are serviced immediately. Jump requests either get serviced immediately (if adequate resources are available) or block. Once a customer's request blocks, the customer stops listening to all streams. The server applies a blocking scheduling policy to service blocked customers when server channels become available. The waiting and blocking scheduling policies are not necessarily the same. Since blocked requests exhibit a different aggregation behavior and the server's objective is to minimize the blocking time, the server applies the FCFS scheduling policy to blocked requests. Like waiting customers, blocking customers defect if they stay in the blocking queue for too long.

B. Proposed Support for Interactive Requests

The server keeps track of the customer's state to help decide when and how to service each group of customers with the same state. As shown in the customer's state transition diagram in Fig. 2, the customer enters the system when he/she posts the first request to playback a certain video. If there are enough server resources, the customer is immediately serviced and assigned the play state. Otherwise, the customer is assigned a waiting state. When the server handles a waiting request, the customer transitions to the play state. As the customer issues interactive requests, the customer's state might change. A customer in the play state stays in the same state until the video ends, he/she issues an interactive request, or runs out of buffer and subsequently blocks. A pause request always changes the customer's state to pause. Similarly, a resume request sets the customer's state to play. A jump forward/backward is either handled immediately, and the customer stays in the play state, or causes the request to block setting the customer's state to blocked. When a blocked request is handled, the customer is returned to the play state.

Fig. 3 shows the methods by which the customer's jump requests are serviced.

1) The customer utilizes its own cache. Each customer allocates a certain amount of memory for stream merging and for keeping old data for future use. If the target playback point falls inside the customer's cache, the interactive request is serviced from the cache without demanding additional server resources. As will be detailed in Section III-E, the cache is organized into segments, with each segment holding a contiguous set of video data.
2) The server merges the request with an already existing B-stream for the same video without using any I-Stream. For a successful merge, the target stream playback point should be within a certain tolerance, called playback point deviation tolerance (DeviationTolerance) from the target playback point. This tolerance is a measure of how close the requested target playback point to the actual (i.e., achieved) playback point. For example, if the playback point deviation tolerance is 10 s, the current playback point is 50 s, and the requested jump distance is +100 s, the actual playback point can be in the range $\{50 + 100 - 10, 50 + 100 + 10\} = \{140, 160\}$ s. The system performance improves with this tolerance, but customers appreciate smaller values.
3) The server utilizes an available I-Stream for merging the request with an existing B-Stream for the same video. The server tries to minimize the skew between

```
ServiceJumpRequest() {
  //1. Try to service request from client cache
  //Loop through all client cache segments
  for (s = 0; s < NumSegments; s++){
    //System keeps track of segment start and end times in sec.
    //Target is Target Playback point in sec.
    if (Target≤Seg[s].end and Target≥Seg[s].start) {
      Service request from segment(s) ;
      break ; } //if
  } //for
  //2. Try to merge with another B-Stream
  MinSkew = MAXFLOAT ; //Initialize minimum skew
  ClosestStream = −1 ; //closest to Target Playback point
  //Loop through all B-Streams for same video
  for (i = 0; i < NumStreams; i++){
    Skew = |Target − Stream[i].Playback|; //calc. skew
    //Choose the stream with minimum skew
    if (Skew < DeviationTolerance)and(Skew < MinSkew) {
      ClosestStream = i; //this is the closest stream
      MinSkew = Skew; } //this is the minimum skew
  } //for
  if (ClosestStream ≥ 0) { //Found a stream to merge with
    Client stops listening to current stream;
    Merge client request with ClosestStream;
    if (stream[current].NumListeners == 0)
      //If Current stream has no listeners, then free it
      free current stream;
    break ; } //if
  //3. Try to service request with an I-Stream
  //Check if there are any available I-Streams
  if (UsedIStreams < AvailableIStreams){
    MinSkew = MAXFLOAT ; ClosestStream = −1 ;
    //Loop through all Streams for same video
    for (v = 0; v < NumStreams; v++){
      Skew = |Target − streams[v].Playback| ; //calc. skew
      if (Skew < MinSkew) {
        ClosestStream = v; //this is the closest stream
        MinSkew = Skew; } } //for
  } //if
  if (ClosestStream ≥ 0) { //Found a stream to merge with
    Client stops listening to current stream;
    Client starts listening to ClosestStream stream;
    Client starts listening to I-Stream;
    if (streams[current].NumListeners == 0)
      free current stream ;
    break ;} //if
  //4. Block
  else{
    Client stops listening to all streams;
    Place request in blocking queue; }
} //ServiceJumpRequest
```

Fig. 3. Simplified algorithm for servicing jump forward and jump backward requests.

the target playback point and the to-be-merged-with-B-Stream playback point in order to reduce the I-Stream length. While an I-Stream is used by a certain customer, the server keeps searching for a suitable B-Stream to merge the customer with it. Upon a successful merge, the server frees the I-Stream.

The order of the methods mentioned above depends on the interactive request type and the streams currently being used by the customer. When a customer issues a pause request, the customer continues to listen to the stream and buffer data if it is a B-Stream until the buffer is full. However, if the customer is listening to an I-Stream, the customer stops listening to the I-Stream immediately. If the interactive request cannot be serviced, the customer's request is placed in the blocking queue. The server services blocked requests by merging them with other B-Streams or using I-Streams. Like waiting customers, blocked customers defect when the blocking time exceeds their tolerance.

C. Proposed Metric: Aggregate Delay

We introduce a new metric called aggregate delay, which quantifies the average delay experienced by clients due to both waiting and blocking. To reflect the increased customer frustration with subsequent blocking, especially if it occurs within a short time of the last blocking, we assign a weight for the blocking, which is the multiplication of the occurrence weight and temporal weight. The occurrence weight, varying from *MinOccurWt* to *MaxOccurWt*, reflects the blocking number within the same session. The first time the customer blocks during the session, the occurrence weight is set to *MinOccurWt*. The occurrence weight is set to *MaxOccurWt* when the number of occurrences is equal to or greater than *MaxOccur*, and is linearly interpolated in between. The temporal weight is to reflect the time difference between two consecutive blocking instances. It varies from *MinTempWt* to *MaxTempWt*. It is set to *MaxTempWt* when the time difference is 0 and to *MinTempWt* when the time difference exceeds a certain time duration, called *TemporalThreshold*. The *TemporalWeight* is linearly interpolated in between. Fig. 4 (particularly CalcAggrDelay function) shows the calculation of the aggregate delay.

D. Proposed I-Stream Provisioning Policy

Since the workload varies with time and hence the system's requirement of I-Streams changes accordingly, we propose an I-Stream provisioning policy. This policy is guided by one of two objectives.

1) Minimize the aggregate delay experienced by the customer, which accounts for both the average waiting time and the average blocking time.
2) Minimize the overall customer defection probability, which encompasses waiting defection and blocking defection probabilities.

We primarily consider the former objective.

The proposed policy first determines the optimal I-Stream threshold measured in seconds for the system. Subsequently, it responds to changing system requirements by adjusting the number of I-Streams and B-Streams. We generalize the SAM protocol to work with stream merging techniques (patching and ERMT) and take advantage of the buffer used by these techniques.

The I-Stream threshold is the maximum I-Stream length allowed to be allocated by the server. Since I-Streams are unicast and hence very costly, any interactive request that could be served using an I-Stream must request it for a period less than the threshold. Otherwise, the request will be denied access to I-Streams, even when they are available. The I-Streams are intended to be used for a short period to service interactive requests that could not be serviced by any other way until they could be merged with multicast streams. By restricting the I-Stream length, this policy prevents any request from using I-Streams for an extended period and hence prevents the system performance from degrading.

```
IStreamProv() { // Run the I-Stream Provisioning Policy
if (TransientPeriod){
  //Direction determines to increase or decrease threshold
  Direction_Prev = 1 ; //Initialize Direction
  CalcAggrDelay() ; //Calculate current Aggregate Delay
  Diff = AggrDelay_Curr - AggrDelay_Prev;
  if (Diff < 0)  //Aggregate Delay improved
    Direction_Curr = Direction_Prev; //Keep same Direction
  else //Aggregate delay worsened
    Direction_Curr = -1 * Direction_Prev; //Reverse Direction
  //Calculate I-Stream threshold adjustment amount
  AdjustAmnt = ceil(Diff) * ADJUSTSTEP;
  if (AdjustAmnt > MAXADJUST)
    AdjustAmnt = MAXADJUST; //Cap adjustment amount
  //Apply change to I-Stream Threshold
  Threshold = Threshold + Direction_Curr * AdjustAmnt;
  AggrDelay_Prev = AggrDelay_Curr;
  Direction_Curr = Direction_Prev ;
} //transient period
else { //Adjust number of IStreams after transient period
  Direction_Prev = 1 ; //Initialize Direction
  CalcAggrDelay() ; //Calculate Aggregate Delay
  Diff = AggrDelay_Curr - AggrDelay_Prev;
  if (Diff < 0)  //Aggregate Delay improved
    Direction_Curr = Direction_Prev; //Keep same Direction
  else  //Aggregate Delay worsened
    Direction_Curr = -1 * Direction_Prev; //Reverse Direction
  AdjustAmnt = (Diff/AggrDelay_Curr) * AdjustFactor;
  if (AdjustAmnt > MAXADJUST)
    AdjustAmnt = MAXADJUST; //Cap adjustment amount
  //Apply change to the number of B-Streams and I-Streams
  BStreams = BStreams + Direction_Curr * AdjustAmnt;
  IStreams = IStreams - Direction_Curr * AdjustAmnt;
  AggrDelay_Prev = AggrDelay_Curr;
  Direction_Curr = Direction_Prev ; }  //else
} //IStreamProv
CalcAggrDelay() { // Calculate the Aggregate Delay
  //Accumulate Total Waiting Time (TotWaitingTime) of waiting customers
  for (c = 0; c < NumWaiting; c++)
    TotWaitingTime+ = WaitingTime[c]; }
  //Loop through all blocking customers
  for (c = 0; c < NumBlkng; c++){
    //Calculate time difference between current occurrence blocking
    //start time and previous occurrence blocking end time
    if (Occur[c] > 1) //Occurrence number during session
      TimeDiff = BlockStartTime_Curr - BlockEndTime_Prev;
    else
      //If this is the first blocking occurrence, use Playback point
      TimeDiff = PlayBack[c];
    //Calculate Temporal Weight (TempWt)
    TempWt = MaxTempWt*TimeDiff/TemporalThreshold;
    if (TempWt > MaxTempWt) TempWt = MaxTempWt;
    if (TempWt < MinTempWt) TempWt = MinTempWt ;
    TempWt = MinTempWt + MaxTempWt - TempWt;
    //Calculate Occurrence Weight (OccurWt)
    OccurWt = (MinOccurWt*(Occurr - 1)/(MaxOccurr - 1);
    OccurWt = MinOccurWt + OccurWt;
    if (OccurWt > MaxOccurWt) OccurWt = MaxOccurWt ;
    //Accumulate Total Blocking Time (TotBlkngTime)
    TotBlkngTime+ = OccurWt * TempWt * BlkngTime[c];
  } //for
  //Calc. Average Waiting Time. NumWaiting is # of Waiting Customers
  AvgWaitingTime = TotWaitingTime/NumWaiting;
  //Calc. Average Blocking Time. NumBlkng is # of Blocking Customers
  AvgBlkngTime = TotBlkngTime/NumBlkng;
  //Calculate current Aggregate Delay
  AggrDelay_Curr = AvgWaitingTime + AvgBlkngTime;
} //CalcAggrDelay
```

Fig. 4. Simplified algorithm for I-Stream provisioning periodically executed after AdjustPeriod.

Subsequently, the system dynamically adjusts the number of I-Streams in response to workload variations. Since the overall server capacity is fixed, this policy works by periodically adjusting the relative ratio of I-Streams and B-Streams in response to workload variations. Any addition to the I-Streams must be accompanied by a subtraction of the same amount to the B-Streams.

Fig. 4 (particularly IStreamProv function) shows the proposed I-Streams provisioning policy. It is executed every specified period, called Adjustment Period or simply AdjustPeriod, which is the time interval between two consecutive adjustments. AdjustPeriod could be kept small to respond quickly to workload variations in rapidly changing workloads, or it could be set to a moderate value in more stable workloads to prevent the server from overreacting to temporary workload changes. Let us first explain the procedure for adjusting the I-Stream threshold. The average waiting time, blocking time, and aggregate delay are calculated for the current AdjustPeriod. The system adjusts the threshold in step increments (AdjustStep) every AdjustPeriod and monitors the aggregate delay. If the aggregate delay becomes better (i.e., smaller) in the current AdjustPeriod compared with the test, the server continues adjusting in the same direction (increase/decrease threshold) as it did in the previous AdjustPeriod. Otherwise, the server reverses the adjustment direction. The AdjustStep has to be big enough to make a difference on the aggregate delay every AdjustPeriod and small enough such that it does not cause the threshold to oscillate. We experimentally found that 10 s is an appropriate step value. MAXADJUST is the maximum value that could be added or subtracted in one AdjustPeriod. The server limits the adjustment amount each AdjustPeriod to prevent the I-Streams from rapidly oscillating back and forth, which degrades the system performance. The procedure for adjusting the number of I-Streams is similar but employs a parameter, called AdjustFactor, which determines the amount of change in the aggregate delay corresponding to the amount of change in the I-Streams. It could be set to a more aggressive value for rapidly changing workloads and it could be kept at a moderate value for steady workloads.

E. Cache Management

We implement a cache management policy to maximize the client cache utilization. The client's cache consists of one or more data segments, with each segment holding a contiguous set of video data. Both future and past data are utilized to service interactive requests. As the cache becomes full, the oldest data in the cache are purged. When two segments overlap, they are merged to form one contiguous segment.

All stream types are used to fill the cache. With patching, for example, a client simultaneously receives data from both a B-Stream and a P-Stream and caches them in two different segments. Any interactive request for data inside either segment is serviced from the cache without requesting any additional server resources. The two segments keep growing until the P-Stream is finished, at which time they overlap and get merged into one contiguous segment.

IV. Performance Evaluation

A. Simulation Platform

We have developed a simulator for both the client-side caches/buffers and the VOD server, supporting various resource sharing and scheduling techniques. To generate each point, we run the simulation a minimum of 450 000 customers

and corresponding interactive requests, accounting for about one million seconds of the simulated time. To make sure that results are stable, we ignore the data during the transient period, which is set to 10% of the total simulation time. The simulator checks the stability of the results and stops after a steady-state analysis with 95% confidence interval is reached.

B. Client, Server, and Workload Characteristics

Table I summarizes the default parameters used. We characterize the waiting and blocking tolerance of customers with a Poisson process with a mean of 30 s. We examine the server at different loads by varying server capacities from 294 to 540 Gbits/s. Only 10% of the server capacity is reserved for I-Streams. We use Minimum Cost First - Per Normalized (MCF-PN) scheduling policy for waiting customers and FCFS for blocking customers except when evaluating the effect of scheduling policies. MCF-PN is a new scheduling policy that we introduce and will be defined in Section V-C. By default, the playback deviation point tolerance is kept at 10 s. The default client-side cache size dedicated to the streaming application is 200 MB.

The default average arrival rate (λ) is 30 requests per minutes. We utilize the results of [7], which characterizes the workload of a media streaming server using actual access logs. This workload determines the distributions of all requests, including interactive requests, for various videos. It shows a strong relationship between the video file duration and interactive operations and also shows a correlation between consecutive interactive operations. The results of that study are consistent with [6], but the latter is less detailed and did not allow for the generation of a full synthetic workload. We study 100 videos of varying lengths and request rates. The workload indicates that 91% of the requested files have average bitrates of 300–350 kbits/s. The average bitrates of the remaining files are in the range 200–300 kbits/s. To be more reflective of recent video and future bitrates, we use 4.9 Mbits/s for the first group and 4.2 Mbits/s for the second.

C. Performance Metrics

We consider the following performance metrics: 1) waiting defection probability; 2) average waiting time; 3) blocking defection probability; 4) average blocking time; 5) blocking probability; and 6) unfairness. The term blocking refers to cases when interactive requests cannot be serviced immediately. The first two blocking-related metrics are analogous to the two waiting-related metrics, which were defined in Section II, but pertain to servicing the interactive requests instead of the initial playback requests. Blocking probability is the likelihood that an interactive request blocks.

For the aggregate delay, we use the following default values: $MinOccurWt = 2$, $MaxOccurWt = 4$, $MaxOccurr = 5$, $MinTempWt = 1$, $MaxTempWt = 3$, *and* $TemporalThreshold = 5$ *min*.

V. Result Presentation and Analysis

We analyze the system performance through extensive simulation. We primarily consider the NVOD server model.

TABLE I
Default Parameters Values

Parameter	Default Value(s)
Arrival Rate	30 Requests / minute
Number of Videos	100
Waiting Tolerance Model	Poisson with mean = 30 sec.
Blocking Tolerance Model	Poisson with mean = 30 sec.
Server Capacity	294 - 540 Gbps
Video Bit-rate	$4.2 - 4.9$ Mbps
I-Streams	10% of server capacity
Buffer Size	200 MByte
Playback Point Deviation Tolerance	10 seconds
Waiting Scheduling Policy	MCF-PN
Blocking Scheduling Policy	FCFS

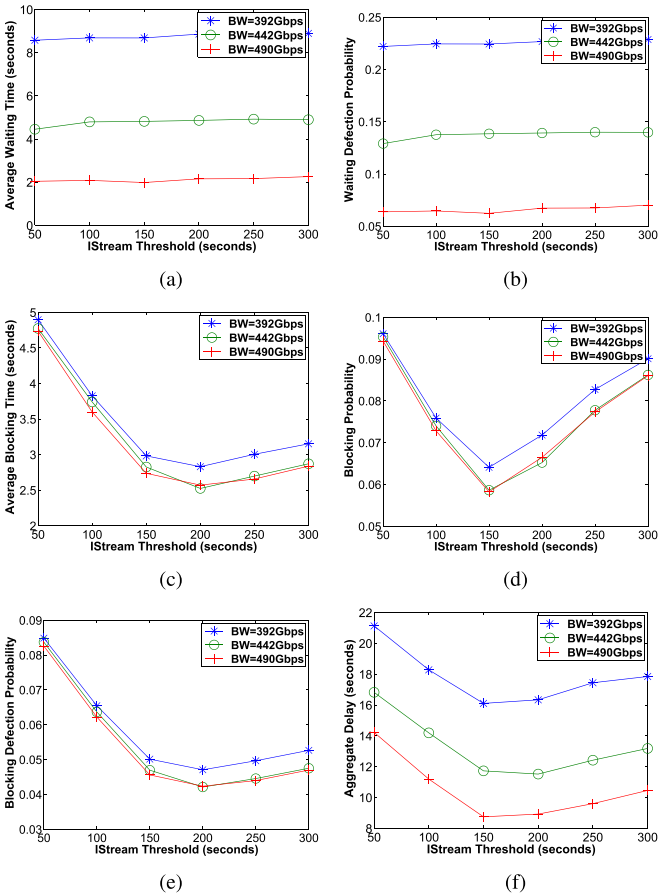


Fig. 5. Impact of I-Stream threshold (BW is the server capacity). (a) Average waiting time. (b) Waiting defection probability. (c) Average blocking time. (d) Blocking probability. (e) Blocking defection probability. (f) Aggregate delay.

The NVOD model converges to TVOD when the server capacity becomes sufficiently high. We focus on the issues introduced by realistic interactive workloads (blocking metrics) since this is an area that has not been investigated by previous studies. Only the most important results are shown for space limitations.

A. Impact of I-Stream Threshold

Fig. 5 shows the effectiveness of the I-Stream threshold on the system performance. When the threshold is set to a small value, most requests are denied access to the I-Streams,

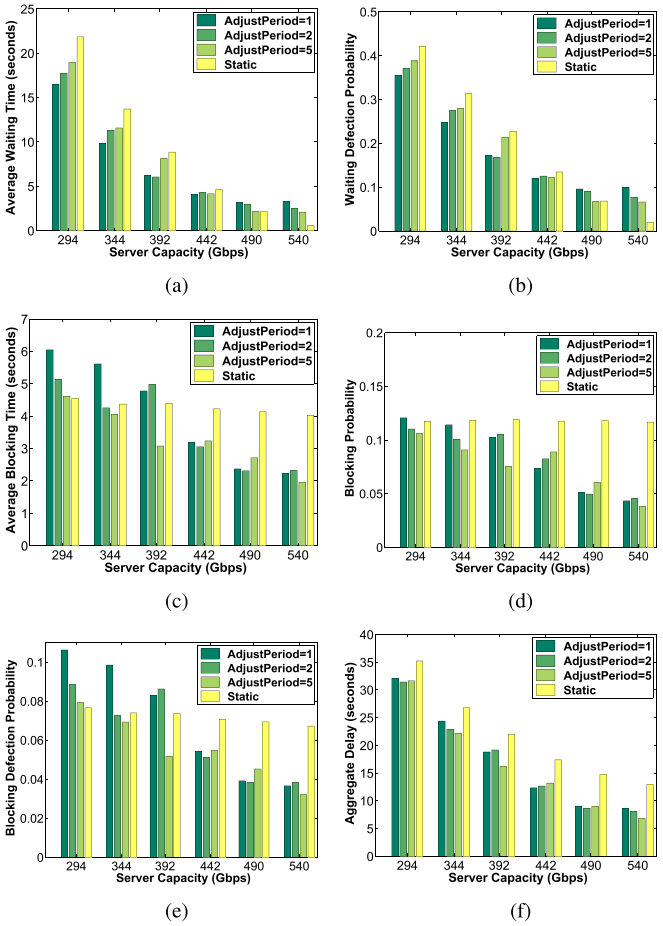


Fig. 6. Effectiveness of I-Stream provisioning policy (AdjustPeriod is in multiples of the longest video length). (a) Average waiting time. (b) Waiting defection probability. (c) Average blocking time. (d) Blocking probability. (e) Blocking defection probability. (f) Aggregate delay.

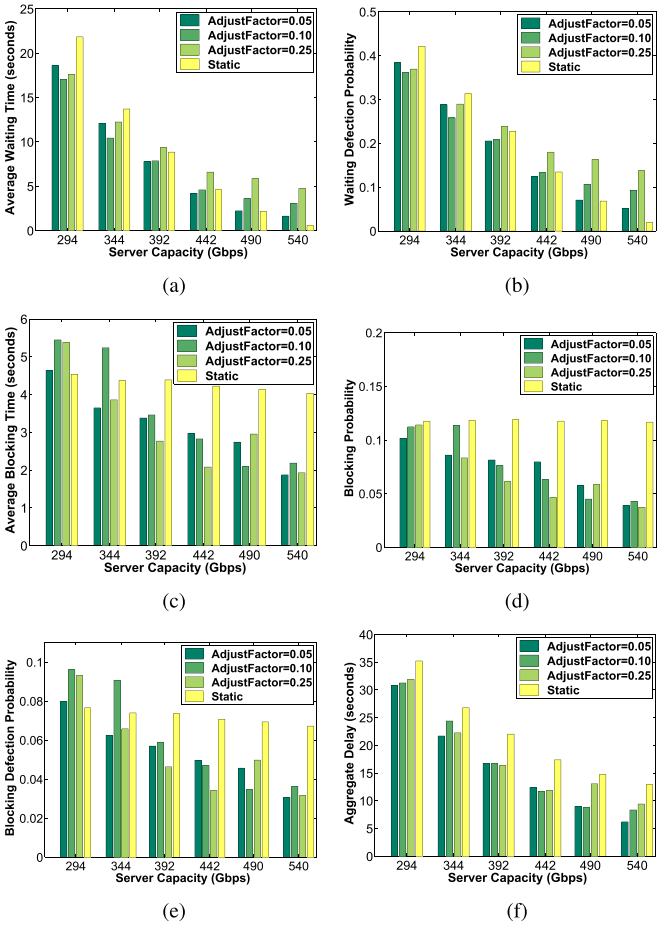


Fig. 7. Effectiveness of I-Stream provisioning policy with different AdjustFactor settings. (a) Average waiting time. (b) Waiting defection probability. (c) Average blocking time. (d) Blocking probability. (e) Blocking defection probability. (f) Aggregate delay.

and hence the I-Streams are severely underutilized. As we increase the threshold, the I-Stream utilization increases and hence the blocking metrics improve until we reach a certain threshold (about 150 s), and then the blocking metrics start to worsen. As the threshold becomes too high, a small percentage of requests hold the I-Streams for too long, preventing other requests from the opportunity of using I-Streams and leading to the degradation of the blocking metrics. We observe this behavior at all server capacities. Since I-Streams are utilized to service interactive requests, the I-Stream threshold does not have a significant impact on the waiting metrics. The aggregate delay follows the same trend as the blocking metrics because it accounts for both average waiting and blocking times and the I-Stream threshold has little impact on the waiting metrics.

B. Effectiveness of the Proposed I-Streams Provisioning Policy

Figs. 6 and 7 show the effectiveness of the proposed I-Stream provisioning policy using patching at different AdjustPeriod and AdjustFactor values, respectively. The proposed policy outperforms the static allocation of I-Streams in terms of the blocking metrics and the average aggregate delay experienced by the client. Up to 45% improvement in the average aggregate delay is achieved. At lower server capacity, dynamically adjusting resources shows a little improvement because the system favors one group of customers over another. Thus, either waiting or blocking customers suffer. Moreover, there is a cost associated with the process of switching resources. However, at higher server capacity, which is the preferred range of operation, this policy shows very significant improvements in the system performance. Using this policy with batching and ERMT techniques exhibits similar results, and thus these results are not shown for space limitations. The AdjustPeriod and AdjustFactor should be selected based on prior analysis (as that in Figs. 6 and 7). (They can also be dynamically adjusted in a manner similar to the I-Stream threshold and the number of I-Streams in the proposed I-Stream provisioning policy.)

C. Impact of the Scheduling Policy

Fig. 8 compares various scheduling policies. MCF-P outperforms other scheduling policies in terms of the average waiting time and waiting defection probability because it minimizes the cost per request. Since shorter videos are more popular and tend to have shorter patches than longer videos, MCF-P is biased toward shorter popular videos and unfair against longer videos. As longer videos are less popular and

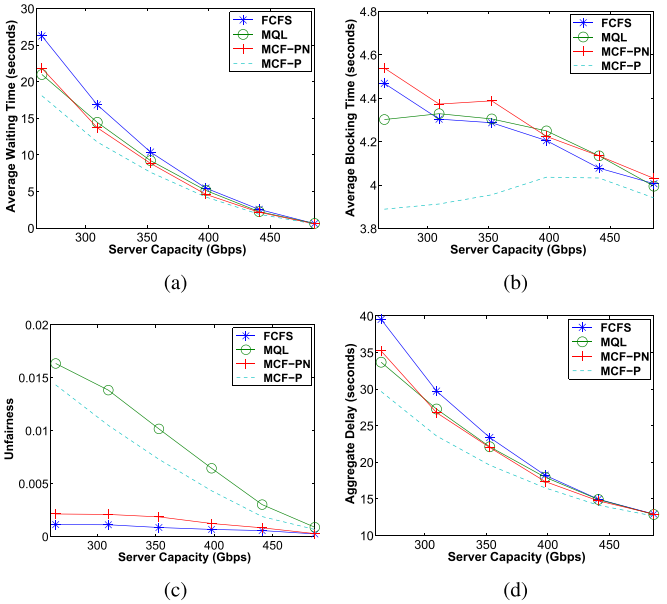


Fig. 8. Comparing effectiveness of scheduling policies. (a) Average waiting time. (b) Average blocking time. (c) Unfairness. (d) Aggregate delay.

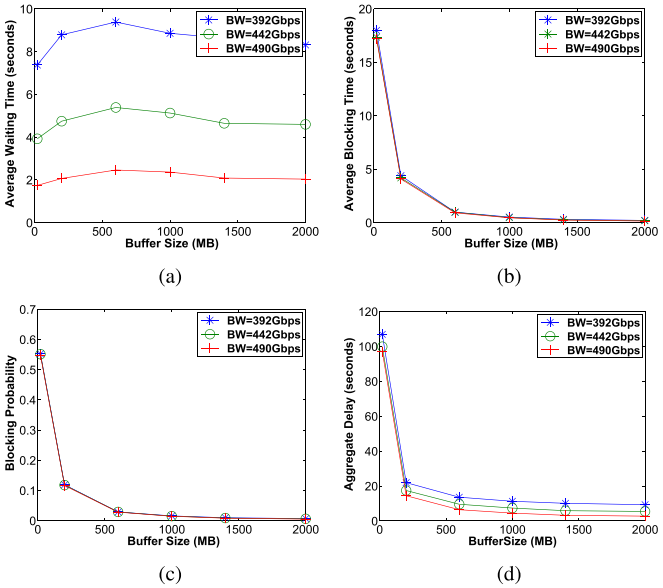


Fig. 9. Impact of client cache size (BW is the server capacity). (a) Average waiting time. (b) Average blocking time. (c) Blocking probability. (d) Aggregate delay.

receive more interactive requests than shorter videos, they tend to block more often with longer blocking times. MCF-P also outperforms other scheduling policies in terms of blocking metrics because it relatively admits shorter videos requests, which tend to block less with shorter blocking times.

At lower server capacity, MCF-P admits a high percentage of shorter videos, which tend to block less with shorter blocking times. Hence, the blocking metrics are kept relatively small. As the server capacity increases, more resources become available to entertain requests for longer videos. Thus, the blocking metrics worsen with the server capacity. This trend continues until we reach a server capacity where the relative number of requests for longer videos does not increase. After that, the blocking metrics start improving.

The main disadvantage of MCF-P is its unfairness as shown in Fig. 8(c). To overcome the unfairness of MCF-P, we experiment with normalizing the stream length required to service a request by the video length. We call this normalized scheduling policy MCF-PN. MCF-PN takes into consideration the number of requests for each video and the stream length required to service a request divided by the video length. This normalization addressed the fairness issue, while delivering decent performance in terms of waiting, blocking, and aggregate delay metrics, as shown in Fig. 8.

D. Impact of the Client Cache Size

Fig. 9 demonstrates the effect of client cache size on the waiting and blocking metrics. The main results can be summarized as follows.

1) As we increase the client cache size, there is little influence on the waiting metrics. Since the system tries to serve interactive requests from the client's own cache, the client cache plays an important role only after the system serves the client.
2) Obviously, increasing the client cache leads to a lower blocking probability and smaller blocking time.
3) The system performs well even with only 200 MB of cache, and cache sizes larger than 500 MB provide diminishing returns.
4) Devices with available buffers smaller than 100 MB are expected to have relatively high blocking probability (more than 10%) if the server capacity is not increased. Most contemporary client devices, including smartphones, are expected to be served well, considering that not all the cache has to be in the main memory. With higher server capacity, the server can better entertain low-end devices.
5) Interestingly, the average waiting time tends to be a little shorter for extremely small client caches. This behavior can be explained as follows. With a very small cache, the blocking defection rate is very high, causing many customers to stop listening to their current streams, thereby leading to many streams with no listeners. Because the server frees all such streams, these streams become available for waiting customers, leading to reduced waiting defection probability and average waiting time.

E. Impact of the Resource-Sharing Techniques

As shown in previous studies, ERMT outperforms other resource-sharing techniques in terms of waiting metrics, followed by patching, as shown in Fig. 10. Since we consider only B-Streams for merging when servicing blocked customers, batching outperforms the other techniques in terms of the blocking defection and the blocking time because all batching streams are B-Streams. However, batching poorly performs in terms of waiting metrics. patching is the best compromise among all three techniques. It performs well in terms of blocking, waiting, and aggregate delay metrics, and it has lower implementation complexity than ERMT.

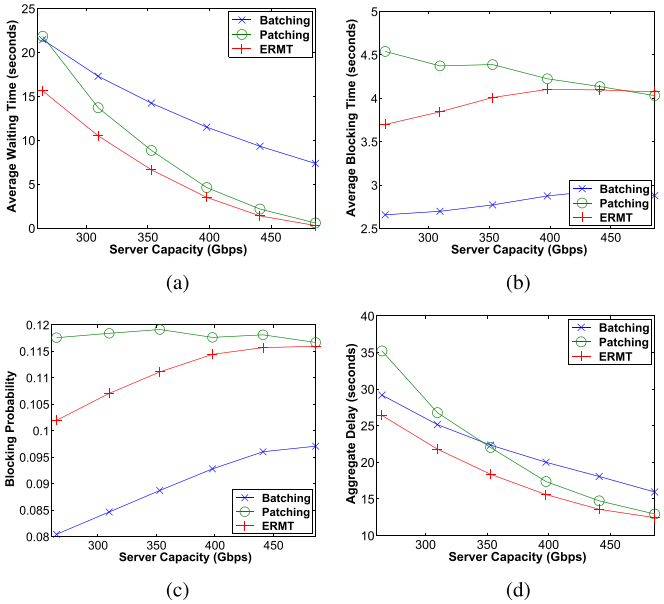


Fig. 10. Comparing effectiveness of resource-sharing techniques. (a) Average waiting time. (b) Average blocking time. (c) Blocking probability. (d) Aggregate delay.

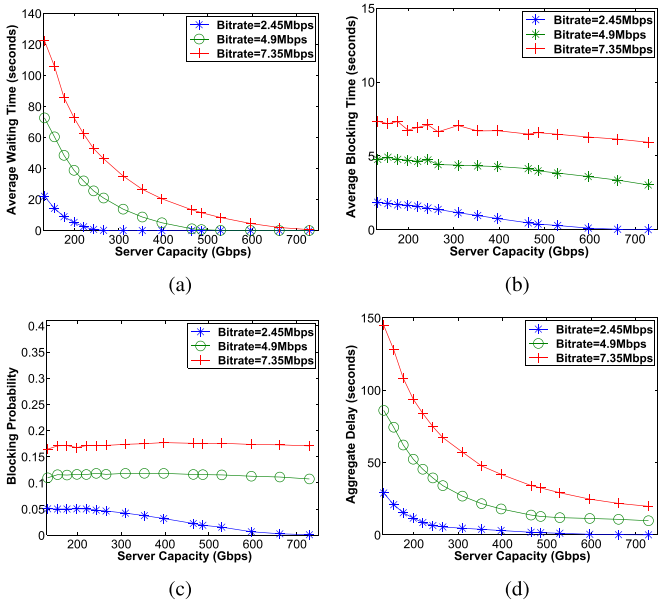


Fig. 11. Impact of video bitrate. (a) Average waiting time. (b) Average blocking time. (c) Blocking probability. (d) Aggregate delay.

F. Impact of Video Bitrate

Fig. 11 demonstrates the impact of video bitrate on the system performance. At higher video bitrates, the average waiting time follows the same trend as that at lower bitrates, but at higher server capacity. However, the average blocking time and blocking probability are significantly higher at higher video bitrates, when keeping the cache size the same for all video bitrates, as the cache can hold smaller video durations at higher video bitrates. Since the cache plays a significant role in servicing interactive requests, the blocking metrics significantly increase at higher bitrates.

VI. Conclusion

We have proposed a scalable solution for interactive NVOD systems and have analyzed its performance under realistic workload and using various resource-sharing techniques and scheduling policies. The main results can be summarized as follows.

1) ERMT outperforms other techniques in terms of waiting metrics. Unlike previous studies, which studied NVOD servers using simple workloads, at low to moderate request rates, the amount of improvement over patching is insignificant. We conclude that there are fewer opportunities for stream merging in the presence of interactive requests. Given that ERMT is of higher implementation complexity, we recommend using patching for systems with low to moderate request rates.
2) Blocking is a major issue for any NVOD server. An efficient NVOD server should continuously monitor the system performance and shift resources in order to minimize blocking. The proposed I-Stream provisioning policy first determines the I-Stream threshold and then adjusts the number of I-Streams dynamically based on the current system state. This policy reduces the blocking probability by up to 65% and the blocking defection by up to 50%. In addition, up to 45% reduction in the average aggregate delay experienced by the client is achieved.
3) I-Streams should be constrained in length to give a chance to all interactive requests for using them. Otherwise, a few interactive requests use the I-Streams in an inefficient manner, degrading system performance. The optimum I-Stream length is system dependent.
4) Deciding which group of waiting customers to serve has a significant effect on the NVOD server performance. The proposed scheduling policy (called MCF-PN) addresses the unfairness issue of MCF-P while delivering decent performance. We recommend using this policy for its performance and fairness in all systems.
5) With efficient cache management, up to 87% of all interactive requests are serviced from the client's own buffer without requiring additional server resources.

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. Comput.*, vol. 50, no. 2, pp. 97–110, Feb. 2001.

[2] V. Aggarwal, R. Caldebank, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and F. Yu, "The effectiveness of intelligent scheduling for multicast video-on-demand," in *Proc. 17th ACM Int. Conf. Multimedia*, 2009, pp. 421–430.

[3] F. Benevenuto, A. Pereira, T. Rodrigues, V. Almeida, J. Almeida, and M. Gonçalves, "Characterization and analysis of user profiles in online video sharing systems," *J. Inf. Data Manage.*, vol. 1, no. 2, pp. 261–276, 2010.

[4] B. Wu, B. Li, and S. Zhao, "Diagnosing network-wide P2P live streaming inefficiencies," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2731–2735.

[5] W. Chi, Y. Xiong, and J. Ma, "Feature analysis and performance evaluation of streaming media scheduling algorithms in patching algorithm family," in *Proc. Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, vol. 4. Dec. 2011, pp. 2332–2335.

[6] J. Choi, A. Reaz, and B. Mukherjee, "A survey of user behavior in VoD service and bandwidth-saving multicast streaming schemes," *IEEE Commun. Surv. Tuts.*, vol. 14, no. 1, pp. 156–169, Feb. 2012.

[7] C. P. Costa *et al.*, "Analyzing client interactivity in streaming media," in *Proc. 13th Int. Conf. World Wide Web Conf.*, May 2004, pp. 534–543.

[8] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. 2nd ACM Int. Conf. Multimedia*, Oct. 1994, pp. 15–23.

[9] F. Dobrian *et al.*, "Understanding the impact of video quality on user engagement," *Commun. ACM*, vol. 56, no. 3, pp. 91–99, Mar. 2013.

[10] D. Eager, M. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective video-on-demand," in *Proc. Multimedia Comput. Netw. Conf. (MMCN)*, Jan. 2000, pp. 206–215.

[11] W. Liao and V. O. K. Li, "The split and merge (SAM) protocol for interactive video-on-demand systems," in *Proc. 16th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, Apr. 1997, pp. 1349–1356.

[12] S. S. Moon, K. T. Kim, S. Lee, H. Y. Youn, and O. Song, "An efficient VoD scheme combining fast broadcasting with patching," in *Proc. IEEE 9th Int. Symp. Parallel Distrib. Process. Appl.*, May 2011, pp. 189–194.

[13] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Commun. ACM*, vol. 49, no. 1, pp. 101–106, Jan. 2006.

[14] E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," in *Proc. IEEE Int. Conf. Multimedia Comput. Syst. (ICMCS)*, Jun./Jul. 1998, pp. 66–75.

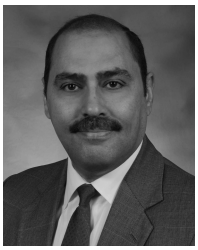
[15] B. Qudah and N. J. Sarhan, "Workload-aware resource sharing and cache management for scalable video streaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 3, pp. 386–396, Mar. 2009.

[16] M. Rocha, M. Maia, I. Cunha, J. Almeida, and S. Campos, "Scalable media streaming to interactive users," in *Proc. 13th Annu. ACM Int. Conf. Multimedia*, Nov. 2005, pp. 966–975.

[17] *Global Internet Phenomena Report*, Sandvine, Waterloo, ON, Canada, 2014.

[18] N. J. Sarhan, M. A. Alsmirat, and M. Al-Hadrusi, "Waiting-time prediction in scalable on-demand video streaming," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 6, no. 2, Mar. 2010, Art. ID 11.

[19] Z. Shen, J. Luo, R. Zimmermann, and A. V. Vasilakos, "Peer-to-peer media streaming: Insights and new developments," *Proc. IEEE*, vol. 99, no. 12, pp. 2089–2109, Dec. 2011.



Kamal K. Nayfeh received the Bachelor's degree from the Jordan University of Science and Technology, Irbid, Jordan, in 1990, and the Master's degree from Wayne State University, Detroit, MI, USA, in 1997. He is currently pursuing the Ph.D. degree with the Multimedia Computing and Networking Research Laboratory.

He has been the Software Team Leader with the Emissions and Fuel Economy Certification Laboratory, Fiat Chrysler Automobiles, Michigan, USA, since 2012. He has been a Software Engineer with Chrysler LLC, Michigan, USA, since 1998. His current research interests include video on demand servers, streaming multimedia, cache management, and interactive video operations.

Nabil J. Sarhan received the B.S. degree in electrical engineering from the Jordan University of Science and Technology, Irbid, Jordan, and the M.S. and Ph.D. degrees in computer science and engineering from Pennsylvania State University, University Park, PA, USA.

He joined Wayne State University, Detroit, MI, USA, in 2003, where he is currently an Associate Professor of Electrical and Computer Engineering and the Director of the Wayne State Multimedia Computing and Networking Research Laboratory. His current research interests include video streaming and communication, automated video surveillance, multimedia systems design, and cross-layer optimization.

Dr. Sarhan was a recipient of the IEEE SEM Outstanding Professional of the Year Award and the Wayne State University President's Award for Excellence in Teaching. He was the Chair of the Interest Group on Media Streaming of the IEEE Multimedia Communication Technical Committee in 2012 and 2014. In addition, he was the Co-Director of the IEEE Multimedia Communication Technical Committee Review Board in 2010 and 2012. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. He has also been involved in the organization of numerous international conferences in various capacities, including Chair, Technical Program Committee Co-Chair, Publicity Chair, Track Chair, Session Chair, and Program Committee Member. He has also participated as a panelist in major international conferences.