The Effectiveness of Intelligent Scheduling for Multicast Video-on-Demand

Vaneet Aggarwal
Princeton University
vaggarwa@princeton.edu

Robert Calderbank
Princeton University
calderbk@princeton.edu

Vijay Gopalakrishnan
AT&T Labs Research
gvijay@research.att.com

Rittwik Jana
AT&T Labs Research
rjana@research.att.com

K.K. Ramakrishnan
AT&T Labs Research
kkrama@research.att.com

Fang Yu
Ohio State University
yufa@cse.ohio-state.edu

ABSTRACT

As more and more video content is made available and accessed on-demand, content and service providers face challenges of scale. Today's delivery mechanisms, especially unicast, require resources to scale linearly with the number of receivers and library sizes. Unlike these mechanisms, with multicast, the load on a server is relatively independent of the number of receivers. Adopting multicast for on-demand access, however, is challenging because of the need to temporally aggregate requests. In this paper, we investigate the importance of an intelligent scheduler and a good data model for achieving good aggregation of requests into multicast groups. We examine the use of an Earliest Deadline First (EDF)-like scheduler that aims to schedule the transmission of "chunks" of video according to their "deadlines" using multicast. We show through analysis that this approach is optimal in terms of the data transmitted by the server. Using trace data from an operational service, we show that our approach reduces server bamdwidth by as much as 65% compared to traditional techniques such as unicast and cyclic multicast. Finally, our approach achieves good aggregation even when 50% of the users use a typical VoD stream-control function like skip, to view different parts of the video.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks]: Network Protocols; C.4 [Performance of Systems]

General Terms: Algorithms, Experimentation

Keywords: Multicast, VoD, Scheduling, EDF

1. INTRODUCTION

With the growth in use of Internet Protocol (IP) for video delivery, there has been a lot of emphasis on delivering video to viewers on-demand. Video-on-Demand (VoD) provides users with the flexibility to view a video of their interest, at the time of their choice ("What I want, when I want"). Popularity of services like YouTube and Hulu, and the effort of every major content provider to have a VoD service is clear evidence of the importance of VoD. These VoD services are typically implemented using a centralized server farm provisioned with a large amount of I/O and network bandwidth to handle peak usage. Each request is handled independently and is serviced using unicast. This, however, does not scale and results in the network links or I/O at the server becoming the bottleneck.

Service providers have attempted to address the problem of scale through a variety of approaches including Content Distribution Networks (CDNs) and Peer-to-Peer (P2P) techniques. CDNs consist of distributed sets of servers, close to the edge of the network. User requests are served by directing the request to the nearest set of servers. P2P approaches push the load even further to the edge of the network by utilizing the storage and bandwidth available at the peers. While studies [16, 18]) show that P2P approaches can be very effective in reducing server load, these approaches still require and heavily rely [25] on the existence of a server that assists in the delivery of content (for seeding, serving rare content, etc.). Thus, while both these approaches significantly mitigate the problem, they do not eliminate it. The fundamental limitation is the use of unicast from the server in both these approaches.

Multicast is unique in that it decouples video popularity from the population of users requesting a particular video; thus server load is mostly independent of content popularity. In this paper, we examine how IP multicast can be used to reduce load on the VoD server. The idea of using multicast for VoD, however, is not new. Several techniques, including Cyclic and Skyscraper multicast [3, 15, 10] have been proposed to take advantage of multicast. An obvious question then is: *Why have we not seen significant use of multicast for VoD delivery?* We believe that there are three main reasons for this: (a) the lack of availability of IP multicast, (b) concerns regarding the ability to aggregate requests, and (c) the data model used in the design of these systems. *Why are we re-considering multicast?* With the on-going deployment of IPTV services, IP multicast is becoming widespread. The second and third issues are somewhat inter-related. Since existing approaches think of videos as a stream, they unilaterally multicast the video. With such an approach, the locality of requests is critical. Without significant temporal locality, unilateral multicast can be wasteful. We address these latter two limitations in this paper.

The foundation to our approach is the delivery of individual segments of a video using dynamically scheduled multicast transfers. Inspired by many of the recent approaches in P2P delivery mechanisms (e.g., BitTorrent [7]), we segment each video into many small pieces that we call "chunks". Nominally, we assume these chunks to be a few tens of seconds in length. A request for a video is mapped into requests for the associated chunks. Each chunk has a corresponding "deadline" by which it has to be delivered. The server employs an intelligent, deadline-driven scheduler to schedule the transmission of these chunks. This scheduler is akin to the traditional Earliest Deadline First (EDF) scheduler [19] and aims to schedule the transmission of chunks according to their deadlines. The server exploits any slack in the deadline of these chunks to aggregate requests at the chunk granularity and serve the chunks using multicast. With the ability of clients to store a reasonable number of received chunks until they have to be played out, the scheduler can transmit, and clients can receive, chunks out of order. This enables the server to potentially satisfy multiple requests for a video using a scheduled transmission of individual chunks based on their deadlines.

In this paper, we use analysis and results from a simulator (based on a prototype implementation) that uses a request trace from a deployed VoD system to demonstrate several properties of our system:

- We prove that scheduled multicast using an unlimited number (but in practice, a manageable number) of multicast groups and a deadline-driven (EDF) scheduler is optimal in terms of the cumulative data sent from the server.
- We show that our approach reduces server load significantly compared to existing schemes. For example, with our traces, results show a reduction of 65% in server capacity compared to unicast and 58% compared to cyclic multicast when we employ our deadline-driven scheduler.
- We show that the scheduler is able to effectively aggregate requests into multicast groups, even when users perform fast forward-like functions. Our results show that even with 50% of the users performing some amount of fast forward, the reduction in multicast batching is less that 4%.

The rest of this paper is organized as follows. We discuss related work in Section 2. We present an overview of the system environment in Section 3 followed by the EDF scheduler design in Section 4. We analyze the EDF scheduler in Section 5. We present our evaluation results in Section 6 before concluding in Section 7.

2. RELATED WORK

There are a number of delivery options for VoD, including server-based unicast (as content providers have often adopted), exploiting caches at the network edge (e.g., CDNs such as Akamai) and P2P approaches (e.g., BitTorrent and PPLive). Our focus in this paper is on server-based approaches and in particular on the use of multicast for VoD delivery. As we show in this paper, our approach reduces the number of concurrent streams from the server significantly. This not only translates to reduction in bandwidth used, but also potentially in the number of servers needed by providers and CDNs.

For multicast to be effective, requests for the same video need to be batched together [21]. Multiple requests can then be served by a single multicast transmission [20], thereby reducing server load and network bandwidth used. Scheduling policies for effective batching of VoD requests have been extensively studied. Dan et al. [8] study two different strategies, First Come First Serve (FCFS) where the client at the front of the request queue is served, and Maximum Queue Length (MQL) policy in which the video with the largest number of users is chosen for service by a multicast group. The Maximum Fair Queuing policy by Aggarwal et al. [2] seeks to achieve fairness by selecting a video to serve based on the queue length, but weighted by a factor that is a function of the access frequency for the video. None of these approaches seek to meet a strict deadline for an individual user's request for a video to be served. In contrast, our work is deadline driven and studies the effectiveness of an EDF scheduler and the extent of batching achieved. We quantify the multicast savings (amount of data transmitted vs. amount of data requested) in realistic scenarios (from actual customer request traces).

Recent approaches [17, 18, 16] to deliver streaming VoD have sought to use P2P techniques for video delivery. The server delivers portions of the video that are unavailable from peers; thus peers assist servers in delivering video. However, as shown in [25], these approaches still require well provisioned server for users to experience a good viewing session. We believe that our approach is complementary to existing P2P-based schemes and that these schemes stand to gain significant reductions [13] in bandwidth usage by augmenting their schemes with server multicast.

Another broad class of techniques for delivering VoD is to use a server-initiated approach to transmission, where selected videos are periodically multicast without responding to individual user requests for a particular video. These include approaches such Pyramid [24], Skyscraper [15] and Dynamic Skyscraper [10]. Here, each video is divided into segments and the server periodically restarts the transmission of each segment regardless of user requests. Clients may prefetch multiple segments concurrently to reduce startup time. However, these approaches need to use multiple download rates. Some of these are difficult to accommodate when the client downlink is bounded (e.g., with DSL or Cable), especially when the startup delay has to be small. Gao et al. [11] study a periodic broadcast (also termed cyclic multicast) scheme and proposed a family of schemes called greedy disk-conserving broadcasting (GDB). A video is divided into non-decreasing sized segments with the goal of guaranteeing an initial latency and effective use of resources. The server periodically multicasts each segment. The shortcoming is that the startup delay can be as large as the size of the first segment of the video. Improvements to GDB avoid this initial latency by patching the portion of the video missed by a client.

Gao et al. [12] proposed a controlled multicast scheme called CIWP, which allows two clients that request the same video to share a multicast group. Unlike batching schemes where the earlier request is typically delayed, CIWP can supply instantaneous VoD service while still taking advantage of multicast group sharing. Note that this is similar in spirit to patching schemes proposed in [14] and [5]. Eager et al. [9] propose stream merging (ERMT), where a client joins the newest multicast group in order to minimize the patch stream. However, as we show in this paper, stream merging techniques are not always practical because of the need to send a patch stream to clients at a high rate. This is particularly problematic when (a) the bitrate of the video is very high compared to the available downlink capacity, and (b) when the server needs to deliver different patch streams to multiple clients at these higher rates.

Our work differs from all the above mentioned schemes in several aspects. We model the video as comprising a number of smaller fixed size chunks, as is the general trend popularized by P2P schemes. We group chunks by means of a deadline-based scheduler that initiates transmission using a multicast group to meet the deadline of user requests for the chunks. We compare the performance of multicast using our deadline driven EDF-scheduler with the cyclic multicast approach which is a hybrid between CIWP and ERMT. Another unique contribution of our work is that our analysis and results are based on utilizing traces from an operational VoD service with requests from a large number of consumers.

3. SYSTEM ENVIRONMENT

Since we primarily study the effectiveness of batching requests and serving video using multicast, our focus is on environments that are capable of delivering content through native multicast. In this section, we describe the salient features of such distribution networks. Note that this does not preclude the use of our work in other settings (e.g., YouTube, Hulu, P2P approaches, etc) when widespread multicast delivery becomes an option, either through native support or through tunneling mechanisms (such as AMT [23]).

Today, customers subscribed to IPTV services have high bandwidth links (e.g., 20 Mbps) over which they receive both standard (SD) and high-definition (HD) videos. The content is usually encoded using MPEG4 standards, resulting in a bitrate of 2 Mbps for SD video and 6-8 Mbps for HD video. The difference between the actual video bitrate and the downlink bandwidth allows for multiple streams of video to be delivered simultaneously to each customer.

As with recent approaches like P2P streaming, we break the video into small "chunks" (of a few tens of seconds in length). In this paper, we multicast to transfer individual chunks from the server. Of particular interest is the scheduling algorithm used by the server. With reasonable amount of storage at the receiver, these chunks may be delivered out-of-order and still achieve a jitter-free, in-sequence playback of the video from the user's perspective. Moreover, with reasonable sized chunks, user requests may be made for each individual chunk which may be transferred using any of a number of delivery options including IP multicast. The typical customer home (residential gateway or set top boxes) has a storage capacity in the range of 1-100 GB, some of which we believe may be used by the service provider for caching content and providing VoD service. We also utilize this storage in our algorithms to buffer chunks that come out of sequence.

Most providers allow the user to fast forward or rewind videos using "trick streams" which are essentially different files of the same video, encoded at a different rate. Hence, a user request for fast forward or rewind results in the VoD server switching to the trick stream until the user resumes "normal play". The traditional approach is to distribute the trick stream via unicast. By breaking up the video into chunks, we can in fact elide the need for trick streams and instead "jump" to an appropriate chunk of the original video and deliver it by multicast. We adopt this approach in our VoD implementation.

4. EDF SCHEDULER FOR VOD

We now describe the scheduling algorithm of the server which is an adaptation of the traditional EDF scheduler. The scheduler uses dynamically created multicast groups to transmit chunks in response to client requests for a video received at the VoD server. A request for a video is mapped by the server to requests for all the chunks of the video, with associated deadlines. Using the EDF scheduler, a chunk with the earliest deadline is transmitted by the server first. The scheduler delays transmission till the last possible moment, as specified by the chunk deadlines. Waiting provides an opportunity for the scheduler to batch the response to chunk requests, thereby reducing the need to send the same chunk to multiple customers at different times (therefore reducing the server load). The deadline to transmit the chunk would take into account variability in the delivery (jitter) and to recover from packet losses using retransmissions. We believe that this reduction in the deadline will not significantly reduce the batching efficiency. The complexity of the scheduler is O(M), where M are the number of active multicast groups.

We look at two forms of the EDF scheduler: 1) EDF-L, using a **Limited** number of multicast groups with a work-conserving scheduler, and 2) EDF-D, using an unlimited number of multicast groups and a deadline-driven **Dynamic** scheduler (i.e., not work-conserving). The schedulers have the following common properties, viz., a) chunks may be received out of sequence, b) a customer can only receive complete chunks that are transmitted subsequent to his arrival, c) a customer can tune to multiple multicast groups in a time slot and buffer the incoming chunks in his STB.

dynamically as determined by the sent out by their given deadlines.

Work Conserving Scheduler with Limited Multicast Groups (EDF-L): In this case, the server has a limited maximum number of multicast groups available and uses a work conserving scheduler to satisfy any pending chunk requests, independent of their deadlines if there is an available multicast group to send it on. Chunks are served in order of their deadlines. A transmitted chunk is received by all clients waiting to receive that chunk, even if they only need it at a later time. If at any time slot, the number of "unique" chunks transmitted is more than the number of multicast groups available, then some chunks would have to be "dropped" (since they will not meet their deadlines). Of course, this would lead to interruption in the client playout. Increasing the number of multicast groups results in fewer chunks missing their deadlines. Because of the work conserving nature of EDF-L, it seeks to always utilize the peak bandwidth out of the VoD server.

Non Work Conserving, Deadline Driven Dynamic scheduler (EDF-D): In contrast to EDF-L, with EDF-D we adapt the number of multicast groups available at every time slot. Once again, we serve the chunks based on their deadline, but we delay transmitting the chunk until its deadline (hence it is non work conserving). By waiting to serve a chunk request until its deadline, we seek to maximize the amount of batching of requests. Just like EDF-L, a chunk transmitted is received by all clients waiting to receive that chunk, in-

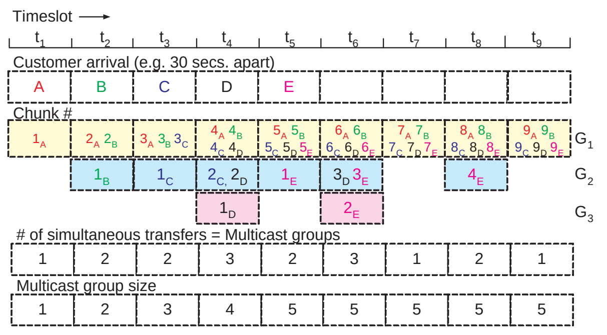

Timeslot
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
Customer arrival (e.g. 30 secs. apart)
A B C D E
Chunk #
G_1
G_2
G_3
# of simultaneous transfers = Multicast groups
1 2 2 3 2 3 1 2 1
Multicast group size
1 2 3 4 5 5 5 5 5

Figure 1: Example server aggregation with EDF-D. 1_A stands for chunk 1 being multicast to A.

dependent of when in the future it is required. The total number of multicast groups used at each time slot is the number of distinct chunks that need to be transmitted to satisfy their deadlines, thus ensuring no chunks miss their deadlines. EDF-D is designed to minimize the average bandwidth required from the VoD server. We prove in Section 5 that EDF-D guarantees the minimum cumulative number of chunks sent up to any given time t.

We use a short example to highlight the EDF-D scheduler's operation. Figure 1 shows the timeline of five users $\{A, B, C, D, E\}$ arriving one after another, a time slot apart and each requesting the same movie M. The server divides M into C_M equal sized chunks $\{1, 2, \dots C_M\}$. For example, let each time slot and chunk size be 30 seconds in length. Also assume that each receiver only has the ability to receive 3 chunks (i.e., join 3 multicast groups) simultaneously. When customer A arrives, chunk 1 is scheduled to be sent out over multicast group G_1 at time t_1. When customer B arrives at t_2, chunk 1 is sent to B on a new group G_2. Recall, that chunk 2 has to be sent to A at t_2. Since B also requires this chunk, requests from A and B can be aggregated and satisfied using one multicast over group G_1. This does not violate the deadline requirements for chunk 2 at either A or B. In a similar vein, when C requests the video at time t_3, chunk 1 is sent to C using group G_2 and chunk 3 is sent to A, B and C. At time t_4, D also requests the movie. Since A has a deadline for chunk 4 at t_4 and all 4 customers require it, the scheduler transfers chunk 4 over group G_1. Similarly, it groups the requests for chunk 2 from both C and D and transfers them over G_2 to satisfy C's deadline. Note that, if D were constrained to receive only 2 chunks simultaneously (instead of 3 in this example), then chunk 4 would be sent to D only at t_7. Proceeding in this manner, we observe that the number of multicast groups that are required after time slot 9 reaches a steady state of 1 for these 5 customers. By adding more users in subsequent time slots, the maximum number of multicast groups that are required grows slowly (sublinearly) with the number of customers. However, as desired, the number of receivers served by a multicast group increases linearly.

5. ANALYSIS OF EDF-D

This section provides an analysis of the EDF-D scheduler. It has been proved in [22] that the EDF scheduler is the optimal scheduling policy in the sense that the number of jobs failing to meet their deadline is a minimum compared to any other algorithm. The proof considers the EDF policy with a fixed number of queues. In our context, we first prove

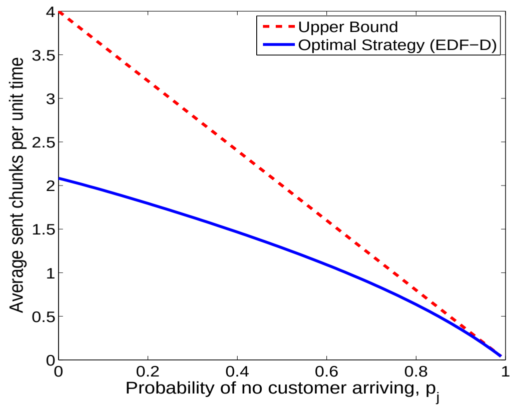


Figure 2: EDF-D bound computation based on State Transition Diagram

that the EDF-D scheduler guarantees the minimum number of chunks sent from the server when compared to any other existing scheduler. Second, we compute an upper bound for the number of chunks sent from the server.

THEOREM 1. *Let $N_D(0, t)$ represent the number of chunks sent from time 0 to t using EDF-D, and $N_X(0, t)$ represent the same for any other scheme X. Then, $N_D(0, t) \leq N_X(0, t)$.*

PROOF. Consider any customer, say c_1 requesting a movie of length L at time t_0. Its j^{th} chunk has a deadline $t_0 + j$. We prove that with EDF-D, the VoD server sends chunk j exactly once in the time duration $[t_0 + 1, t_0 + j]$ which is optimal since the customer has to receive the chunk prior to its deadline $t_0 + j$.

We prove the above by contradiction. Suppose the VoD server sends chunk j more than once in $[t_0 + 1, t_0 + j]$. Let these be two time instances in this time range, say t_1 and t_2 ($t_1 < t_2$). Since the customer c_1 had requested the movie prior to time t_1, it would get chunk j at t_1 since the chunk is sent by multicast to all waiting customers by the VoD server. Now, if the VoD server sends this chunk again at time t_2 ($t_2 \leq t_0 + j$), some other customer c_2 must have a deadline for chunk j at t_2. This means customer c_2 must have arrived at time $t \leq t_0$ (since $t_2 - j \leq t_0$) which leads to a contradiction, since this customer should also have received the chunk at time t_1. This leads us to the fact that a chunk is only sent once within its deadline duration by EDF-D. More generally, for M chunks, this process is repeated M times. Thus, EDF-D results in the minimum number of chunks sent from the VoD server. □

5.1 Bound computation

In this section we derive an upper bound for the number of chunks sent by the VoD server. Since it was proved in Theorem 1 that EDF-D sends the minimum number of chunks, we can consider any other suboptimal algorithm to give us an upper bound.

Upper bound: For each customer arrival at time $t - 1$, we consider enqueueing N chunks of movie j in the next time slot t, where N is the length of the movie. This denotes the worst case situation, since the VoD server attempts to send N chunks at every time slot as long as at least one customer has arrived. We obtain the average number of chunks sent for time interval T to be $NT(1 - p_j)$, where $1 - p_j$ denotes the probability of one or more customers arriving at a given time slot. Therefore, the average number of chunks per unit time is less than or equal to $\lim_{T\to\infty} \frac{NT(1-p_j)}{T} = N(1-p_j)$. This loose upper bound is better than cyclic multicast.

EDF-D: Figure 1 provides an example of how chunks are served in EDF-D. Depending on whether a customer arrived in a particular time slot or not, a state transition occurs. As an example, we consider requests for a movie which is four chunks long. In this case, each state has four columns, and each column lists the chunk(s) that need to be sent by its corresponding deadline. At each time slot, chunks at the head of the queue are sent by the VoD server. By enumerating all possibilities of the finite state machine for this video we obtain 44 states in all. Denoting the probability that no new customer requests this movie as p_j, we can find the transition probability matrix S for all 44 states, given by a 44×44 matrix. The steady state distribution is found by solving the system of equations given by $\pi = \pi S$, such that $\pi \mathbf{1} = 1$, where π is a 1×44 row vector of steady state probabilities and $\mathbf{1}$ is column vector of length 44 of all ones. Since we know the number of chunks sent in each state, we can compute the expected number of chunks that are sent per unit time slot. We plot this number in Figure 2 for EDF-D.

Figure 2 shows the performance of EDF-D compared to the upper bound derived above. Upon a customer arrival (i.e., $p_j = 0$), the average number of chunks sent per unit time for the upper bound is 4. This is intuitive, since the chunks of the movie are all enqueued for each customer arrival in this example upper bound. However, for the EDF-D scheduler, the corresponding average number of chunks sent per unit time is just above 2 for $p_j = 0$.

6. EXPERIMENTAL RESULTS

We first provide a characterization of customer accesses of video from a deployed VoD service. We then analyze the sensitivity of our scheduler to different parameters through detailed experiments with a trace-driven simulator as well as an implementation.

6.1 VoD Data Characteristics

We used real VoD request traces from a nationally deployed VoD service for our experiments. We were provided with data on customer requests for one month (out of several months of trace data). The trace included the length of the video and time of each customer request for a video. Customer identities and video names were anonymized for privacy and security reasons. We highlight interesting characteristics of this data, focusing particularly on properties that affect the ability to respond to requests using multicast.

Popularity: We computed the number of requests for each video, and ranked them in order of their popularity. Figure 3(a) shows the distribution of the popularity of videos, which approximates a Zipf distribution. The popularity is Zipf, but only at the waist, with the parameter (α=0.6). The most popular videos (rank 1 through 10) deviate from Zipf, with the curve being flat, suggesting that they have the same popularity. Our fit with a Zipf distribution for the popular videos is consistent with previous observations [1][6]. Cha et al.[4] also report similar observations with YouTube requests and have shown that the long tail can be modeled by a combination of Zipf and an exponential cut off distribution.

Video Length: We plot the cumulative distribution of the length of the videos that are requested by viewers in Figure 3(b). It is interesting to note that many of the videos that are requested are very short: 30% of the requested videos are 10 minutes or shorter. The median length is approximately 30 minutes, as a number of the requested videos are music videos, movie trailers, or episodes of TV shows.

Parameter	Default	Range
EDF time slot	15 sec	15 - 30 sec
Chunk length	30 sec	15 - 60 sec
Video Bitrate	2 Mbps	-
Receiver bandwidth	∞ Mbps	2 - ∞ Mbps
Cyclic multicast cycle length	30 min	1 - 140 min
Cyclic multicast popular video set	10%	1 - 20%

Table 1: Parameters values used in experiments

Popularity Evolution: Figure 3(c) shows how the popularity of a video evolves with time. The top 1%, 5% and 10% of the movies on the first day are tracked over a period of one month. By comparing the popular videos on the first day with popular videos in the remaining days, we compute the fraction of videos that stay popular throughout the month. In general, there is diminishing popularity of videos with time. The occasional peaks within the declining profile are weekends which boost the popularity of certain videos.

6.2 Experimental setup

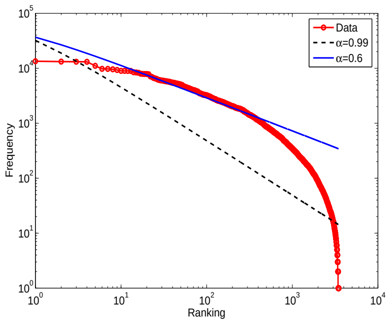
In order to evaluate our approach, we built a prototype VoD server and clients that implement our approach. The clients and the server run on GNU/Linux and play the videos using a typical media player. However, in order to study the system at scale, we built a custom simulator that uses the implementation codebase. The only difference is that we do not transmit the chunks or the messages over the network. We emulated the entire system and evaluated its performance using the trace of customer accesses. We compare our approach with a system that just uses unicast as well as one that performs cyclic multicast with unicast patching.

Input Data and Parameters: We used the user request data (from 12:00 AM to 11:59 PM) for one day in 2008 (busiest day of the month, largest number of requests) of the trace to study the performance of the different systems. We believe that the data from this day captures the entire range of situations observed in the VoD system and is sufficient to gain an in-depth understanding of the performance of our approach. We have validated many of our observations by examining the performance across multiple days. Table 1 summarizes the parameters used in our experiments.

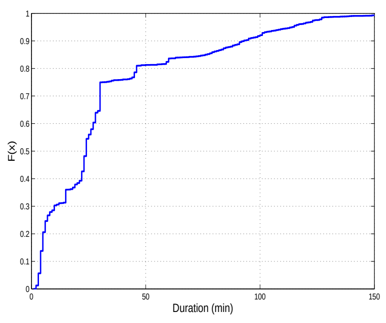
Metrics: In all our experiments, we measure the performance of the schemes using two metrics: (a) Server bandwidth used, and (b) the amount of aggregation achieved. While the former gives us an indication of the amount of work that a VoD server has to do (and hence indicative of the scaling properties of each approach), the latter allows us to understand the effectiveness of the different multicast scheduling strategies. We adopt a simple relationship between the number of multicast groups created and server bandwidth, which is 1 multicast group = 2 Mbps, assuming other overheads such as control message are negligible.

6.3 Effectiveness of EDF-L scheduler

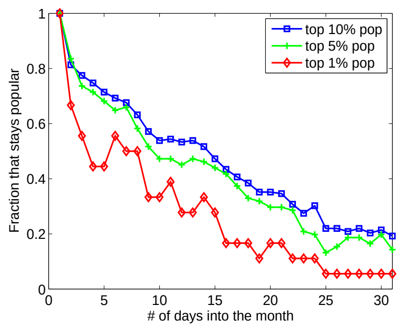
Since the EDF-L scheduler is work-conserving, it maximizes the use of available multicast groups (and hence the bandwidth available through them), even if it means transferring chunks before their deadline. This, however, could result in lost opportunities to batch subsequent requests for



(a) Popularity of videos requested



(b) Cumulative Dist. of video length



(c) Evolution of VoD popularity

Figure 3: VoD input data characteristics

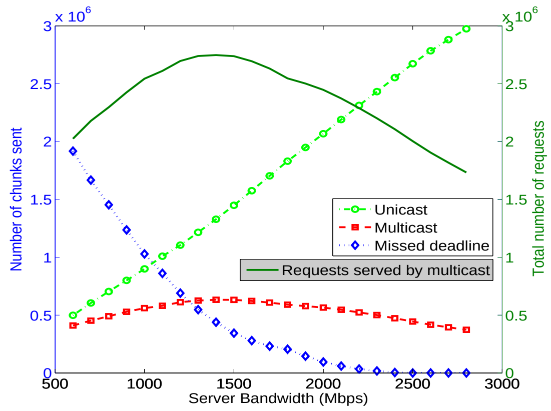


Figure 4: EDF-L: Number of multicasts, unicasts, and dropped requests.

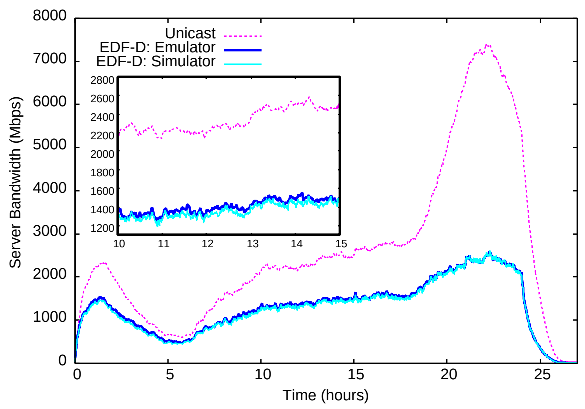


Figure 5: Server bandwidth used by EDF-D and Unicast.

the same chunk. In this experiment, we vary the amount of bandwidth available to the server (hence the number of multicast groups) and study the level of aggregation achieved. Results are shown in Figure 4. As expected, the scheduler drops fewer chunks as the available bandwidth increases. No chunk misses its deadline (i.e., dropped) when there is 2.4 Gbps bandwidth (compared to 7.4 Gbps needed by unicast in Figure 5). Note that the number of chunks multicast increases with bandwidth and later decreases. This is because initially, there is not enough capacity and the server drops (is unable to satisfy the deadline) many requests. As the available server bandwidth increases, the server is able to service more requests, and also able to aggregate more. But when there is adequate bandwidth, the server transfers any outstanding chunk requests irrespective of their deadlines, thereby losing out on the opportunity to batch later requests. We also plot the number of requests that are serviced by multicast. This number increases and then drops, again indicating the reduced aggregation with increasing bandwidth. Further, comparing this with the number of chunks sent by multicast shows that multicast is able to decrease server load by about a third.

6.4 Effectiveness of EDF-D scheduler

While EDF-L seeks to take advantage of a limited number of available multicast groups, we believe that primary limitation is the bandwidth available out of the server, rather than the number of multicast groups. We have observed that the number of multicast groups does not grow substantially with client population size, request rate or VoD library size. With a growing client population, and with a skewed distribution of requests to popular content, the effectiveness of multicast only grows. To examine the effectiveness of multicast without being constrained by the number of multicast groups (but being aware of the required bandwidth from the server), we now look at the performance of EDF-D. We first compare the effectiveness of the EDF-D scheduler with unicast in terms of required server bandwidth to serve all the user requests. We used the default simulation parameters and ran the experiments on both the simulator and the emulator. We present the results in Figure 5.

First, the peak server bandwidth needed for EDF-D, both from the simulator and the emulator, is about a *third* of what is needed by unicast (2.6 Gbps vs. 7.4 Gbps). Secondly, multicast results in less server bandwidth at all times, even during low usage periods. Finally, there is very little difference between the emulator and the simulator results, validating the accuracy of our simulation-based results. Figure 5 also shows an inset plot that examines the server bandwidth needed between 10AM and 3PM in detail. As is evident, there is a very small difference between the server bandwidth reported by the simulator and the emulator; this difference represents the practical overheads such as communication delay, protocol overheads, etc. We also analyze the amount of aggregation achieved by EDF-D in Figure 6, which shows the size of multicast groups that the server transmits to, and the number of occurrences of each sized multicast group. We see that the EDF-D scheduler is, in general, able to aggregate many requests, and is able to serve as many as 168 requests with one transmission. 38%

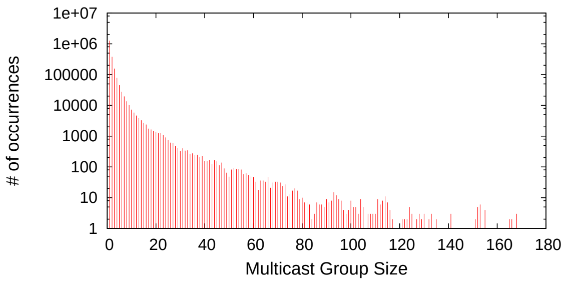


Figure 6: Aggregation achieved by EDF-D.

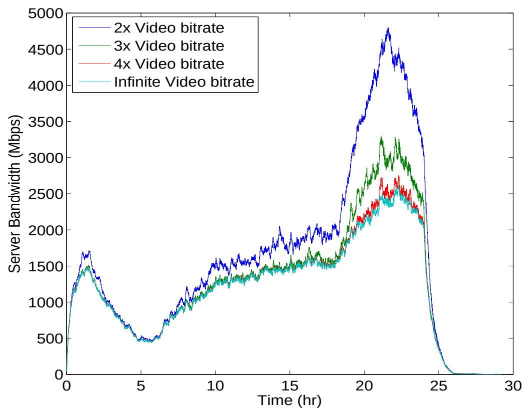


Figure 7: EDF-D – Server bandwidth required when the client has constrained downlink bandwidth.

(∼780K transfers) of the 2M transfers from the server are multicasts (at least 2 receivers). These transfers satisfy 71% of the 4.5M requests received at the server. The use of EDF-D thus reduces the overall work done by the server by 54%.

6.5 Effect of Downlink Capacity

In all the experiments so far, we have assumed that the client has infinite downlink capacity and can receive an infinite number of chunks in parallel. This however is not true in practice. Clients have a finite downlink capacity and can only receive a small number of chunks in parallel. In this experiment we vary the amount of available downlink capacity at each client from twice the video bitrate (in other words, allowing two chunks to be received in parallel) to infinity. Using the EDF-D scheduler, we then measure the server bandwidth needed. Figure 7 shows that there is a marked decrease in the server bandwidth when we increase the maximum downlink bandwidth from twice (2x) to four times (4x). Interestingly, we see that beyond four times the video bitrate, there is a diminishing benefit in terms of reducing the server bandwidth and is close to the case when the client has infinite downlink capacity.

In order to understand the reason behind this result, we measured the number of chunks received by clients in parallel, over a ten minute window in the peak period. Figure 8 shows the distribution of the number of chunks received in parallel by clients with infinite downlink capacity. The majority of the customers receive less than 4 chunks at each time slot (15 seconds in our experiments). In fact, customers receiving 4 chunks or less account for 96% of the total chunks. Thus, as we reduce the downlink from infinite capacity to four times the bitrate, the number or chunks that need to be transmitted multiple times is much less than what

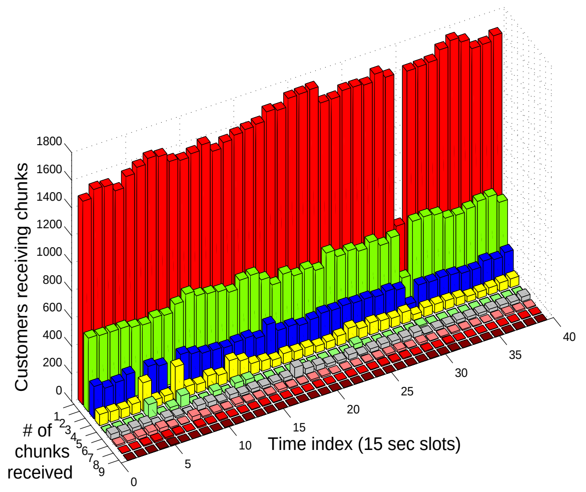


Figure 8: Distribution of the number of chunks received in parallel by clients using EDF-D

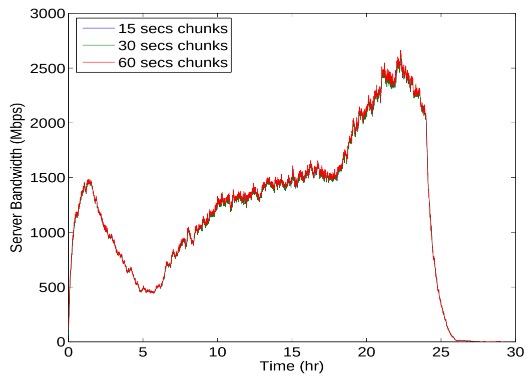


Figure 9: EDF-D – Server bandwidth required for different chunk sizes.

needs to be separately transmitted when the bandwidth decreases further, to only twice the bitrate. In fact, the cumulative amount of server resources used over the entire day (i.e., area under the curve) is 169 Tb when clients can receive 2 chunks at a time (2x), 134 Tb with 3x, 125 Tb with 4x and 123 Tb with infinite downlink capacity. This confirms that allocating a client downlink capacity of 4 times the video bitrate is sufficient to realize most of the gain.

6.6 Effect of Chunk Size

An important aspect of our design that allows good aggregation is the concept of chunking the video. Because the server aggregates at the chunk level, we are able to achieve substantial aggregation of requests. In this experiment, we study the sensitivity of EDF-D to chunk size. We vary the chunk size from 15 seconds worth of video to 60 seconds, and measure the server bandwidth required to serve all client requests. Interestingly, the results in Figure 9 show that EDF-D is almost insensitive to chunk size. The amount of server bandwidth is almost identical across all the chunk sizes. We believe similar results apply to EDF-L.

6.7 Effect of Skipping

As specified in Section 3, fast forward functionality can be supported through one of two mechanisms. A user wanting to fast forward through a video at higher speed than reg-

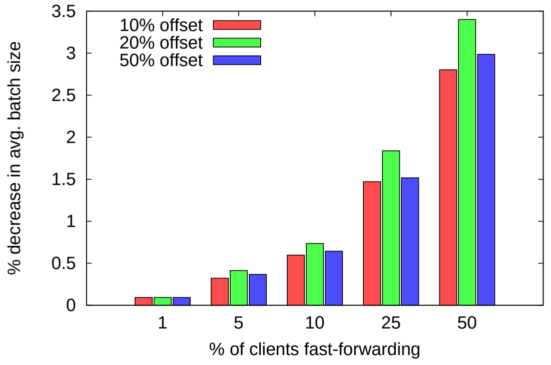


Figure 10: Percentage reduction in the amount of aggregation when clients skip portions of a video.

ular playout can be supported using the typical approach of sending a "trick file". With the existence of trick file, a fast forward is just a request for another video file and is hence handled trivially by the server (albeit with reduced batching when multicast is used). The second mode of fast forwarding, which we call "skip", is to jump to specific points in the video directly (like chapters in a DVD). This requires moving the client to a different point in the same video file, thereby breaking up the batching. It is this scenario that we analyze here.

In this experiment, we varied the number of users requesting videos that also perform a skip, from 1% to 50%. Each skip request skipped anywhere from 10% of the video to 50% of the video. We measured the decrease in the average batch size compared to the case where users do not skip any portion of the video. The results of this experiment is shown in Figure 10. As expected, skips result in a decrease in the amount of batching. That said, the decrease is negligible (max $< 3.5\%$). As expected, the amount of batching decreases as the number of users performing skips increases. Additionally, we measured the increase in server load due to skipping for all these experiments and observed that the increase in server load was negligible. In the worst case, we needed about 50 Mbps more server bandwidth.

6.8 Overhead of Using Scheduled Multicast

A question that arises with multicast being used in the manner we do in this paper is that of the overhead of setting up a tree. Note that the transfer of each chunk results in a few thousands of packets, and the cost of constructing the multicast group is amortized across this fairly large transfer. The multicast tree construction involves at most two messages to be processed by the server (to announce the group and potentially receive a join from one of the clients) and each of the clients (to issue a join that is processed by intermediate routers and to receive a message from the server announcing the group). Routers process the joins, store state for the group and forward the join upstream on the first join for the group. The processing time of each of these messages at each entity is only a few milliseconds. Our implementation experience indicates that a join/leave operation terminating on an on-tree node takes less than 10 milliseconds of processing time. Given a chunk size of 30 seconds (60 Mbits, assuming 2 Mbps video), we believe that the dynamic creation of multicast groups on a per-chunk basis is well within acceptable limits. We have experimented with chunk sizes going down to a few seconds. Decreasing the chunk size below 2 to 3 seconds results in the overheads (requesting chunks, multicast join etc.) become significant. We found that maintaining chunk sizes of the order of 10 seconds was sufficient to amortize the overheads adequately.

6.9 Comparison with Cyclic Multicast

To compare our EDF-based "scheduled multicast" approach with existing multicast mechanisms for supporting "near-on-demand" video distribution, we examined the performance of cyclic multicast with unicast patching, CIWP [12]. We compare the amount of server bandwidth required as well as the extent of batching (number of receivers actively receiving a particular multicast transmission).

With cyclic multicast, a subset of the movies (typically based on popularity) are unilaterally multicast by the VoD server, irrespective of whether individual users have requested that video or not. When the video is considered a single segment, the video is multicast repeatedly, and multiple copies of the same content may be multicast, offset by a certain amount. Alternatively, the video may be broken up into multiple segments, with each segment typically a fixed stream length (e.g., a 2 hour movie may be broken up into 4 equal segments of 30 minutes each). For a particular user request of one of those popular movies, the user is directed to the multicast group being used for the first segment. When the user joins the multicast group, the multicast of the segment may already be in progress. To meet the "near-on-demand" desire for viewing of the content by the user, the VoD server delivers the prefix portion of the segment missed by the user as a unicast. This portion delivered by unicast is termed "patching". This patch stream is transferred at the same rate as the multicast stream (thus using just a little over twice the playout rate of the client downlink bandwidth). Note that our implementation of cyclic multicast is in fact a hybrid approach where the newly arriving client joins the multicast stream that has been initiated most recently, so that the amount of content unicast with a patch stream is minimized (as proposed in ERMT [9]). The only difference is that ERMT creates a stream upon a request from a client in contrast to the cyclic multicast approach we have modeled.

While there exists more recent work like Skyscraper, Pyramid or Permutation Pyramid broadcast, we do not compare with them because we believe that these schemes are difficult to deploy because of practical constraints. These schemes reduce the startup latency of cyclic multicast by having the client temporarily download a large number of streams simultaneously. However, in order to be effective, these schemes require much higher bandwidth than is feasible on our intended deployments. For instance, with a 1 minute startup delay on a 1.5Mbps video stream, skyscraper shows that all of these schemes require clients to periodically download at rates higher than 125Mbps. It is difficult to accommodate such technology in typical client environments (e.g. DSL, Cable) where the downlink bandwidth is limited.

Using the VoD request trace, we evaluated the performance of cyclic multicast using a trace-driven simulation. The cyclic multicast stream for each popular video is chosen to be a fixed size and multicast repeatedly (the stream size is varied in our experiments to examine the sensitivity to this parameter). As before, the video playout rate is 2Mbps.

We examined the load on the server as increasing fractions of the most popular movies are cyclically multicast,

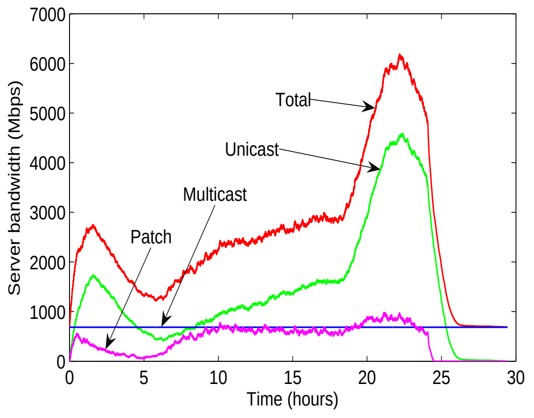


Figure 11: Cyclic multicast server bandwidth: variation w/time on busy day

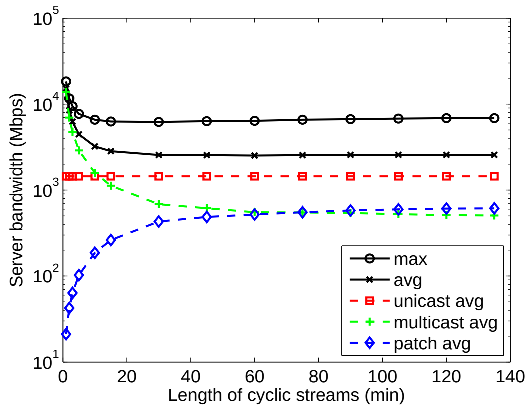


Figure 12: Cyclic multicast server bandwidth vs. length of cyclic stream

ranging from 1% to 10%, and for increasing stream lengths, from 1 minute to 2 hours (few movies are longer than this in our library). We show results for the case where 10% of the movies are cyclically multicase because of the more gradual and predictable drop-off in popularity of these movies (see Figure 3(c)) over a period of 30 days. The observed predictability makes it easier for a provider to set engineering rules to determining which movies should be set up for cyclic multicast. Figure 11 shows the server bandwidth for cyclic multicast streams for the top 10% of the movies cyclically multicast, with a cycle stream length of 30 minutes, which yielded the minimum peak server bandwidth.

The figure also breaks down the total server bandwidth into its sub-components of multicast, patching and unicast (for unicast delivery of the not-so-popular movies that are not cyclically multicast). Multicast bandwidth is constant because it is independent and oblivious of the user requests. It is interesting to note that patching (delivered on a unicast basis) requires almost the same bandwidth as multicast, and therefore is a significant load on the server. The unicast bandwidth demand for the remaining 90% of the not-so-popular movies dictates the behavior of the total server bandwidth requirement, especially the peak server bandwidth. The peak server bandwidth is about 6.2 Gbps.

Comparing cyclic multicast with the EDF-based "scheduled multicast" approach, we find that cyclic multicast places a server load of at least (across the parameter range we studied) 6.2 Gbps in comparison to less than 3 Gbps with

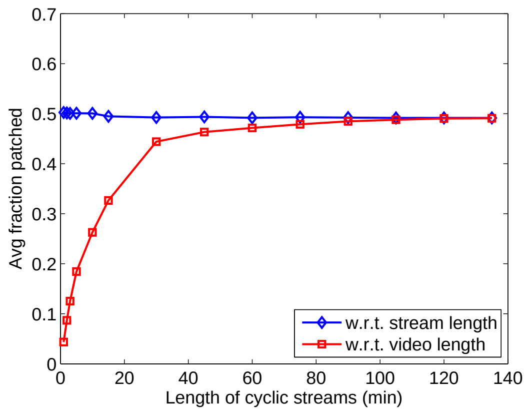


Figure 13: Proportion of Patch Stream: variation w/length of cyclic stream

scheduled multicast, for our traces.

We then examine how the server bandwidth demand varies as the cyclic stream length is changed for a given percentage (top 10%) of videos being cyclically multicast. In Figure 12, we observe that the peak server bandwidth demand reduces rapidly with increasing stream length because the number of concurrent multicast groups drops quickly as the stream length gets larger. The figure also breaks down the components of the server bandwidth, looking at the average multicast, unicast and patch bandwidth requirements. The average multicast demand dominates for small stream lengths, but just like the peak server bandwidth, drops quickly for larger stream lengths. Unicast patching, meant to support a low video startup latency, increases and contributes a significant proportion (as much as multicast) when the stream length increases. Unicast bandwidth demand for the not-so-popular videos is independent of the cyclic stream length and stays constant. The peak server bandwidth reaches a minimum for a stream length of 30 minutes and increases when the patch streams get larger.

As shown in both Figures 11 and 12, the server bandwidth needed for patching is quite significant. To understand the cause of this, we measure the extent of patching with respect to the length of the cyclic stream as well as the video length. Figure 13 shows that on average the server has to patch about half of the first stream for each request for a popular video. When stream length is small, the fraction of the complete video patched is small. However, when stream length becomes approximately the length of the video (roughly for streams longer than 30 minutes), about half the video is patched. To keep the patch bandwidth small, we need small stream sizes.

It is critical to understand the effectiveness of batching with cyclic multicast. Figure 14 shows the average number of receivers batched together for a given stream. The average batch size reduces as the percentage of popular movies cyclically multicast increases. As expected, batching increases with larger stream lengths. However, the percentage of popular videos cyclically multicast plays a more dominant role.

In summary, we observe that a small cyclic stream length imposes a much heavier server load, contributed by the multicast streams. But, a larger cyclic stream length increases patching bandwidth, thus requiring us to balance these two conflicting requirements on the server. Finally, cyclically multicasting a high percentage of popular videos reduces the effectiveness of batching, with the potential for streams

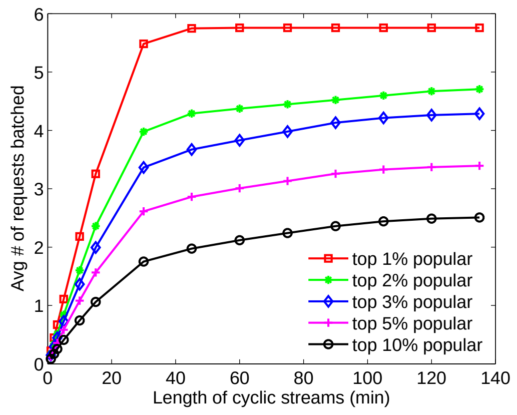


Figure 14: Batching effectiveness of cyclic multicast: Avg. # of receivers of each individual cyclic stream

to have very few users actively receiving the stream.

The batching efficiency is very low with cyclic multicast, with an average of between 2 and 6 receivers for each of the popular movies cyclically multicast. On the other hand, EDF-D, because of its ability to adapt, is better at using the multicast groups (see Fig. 6). The average batching of receivers is approximately 2.176, but this is measured across all the movies in the library. This suggests that a VoD service built on top of multicast as the basis can gainfully exploit the adaptation incorporated in EDF-D scheduled multicast, rather than "blindly" multicasting streams using cyclic multicast.

7. CONCLUSIONS

In this paper we study the effectiveness of intelligent scheduling for multicast video-on-demand delivery. Multicast is attractive since it consumes a fixed amount of server bandwidth regardless of the number of simultaneous requests to a particular video. This is well suited for live streaming but conventional wisdom has been that it is not as suitable for on-demand access due to insufficient temporal locality in user requests for a particular video. In this paper, we proposed a scheduled multicast approach in conjunction with a "chunked" video data model and showed that it can achieve a substantial amount of aggregation in delivering chunks via multicast groups to multiple customers.

We present two versions of our scheduler based on earliest deadline first policy, namely, EDF-L and EDF-D. Adaptive scheduling reduces the server bandwidth requirements (EDF-D). We show through analysis that the EDF-D scheduler is *optimal* in the number of chunks sent from the server. By simulation and emulation using real operational VoD traces, we quantify the performance improvement for EDF based schedulers compared to cyclic multicast and unicast. By sending multiple chunks in parallel, aggregation is greatly improved with scheduled multicast while at the same time reducing the server bandwidth requirement. Using EDF-D we save 65% of the server capacity when compared to unicast and 58% compared to cyclic multicast. We also show that skipping and fast forwarding videos do not significantly diminish our ability to aggregate requests.

8. REFERENCES

[1] S. Acharya, B. Smith, and P. Parnes. Characterizing User Access To Videos On The World Wide Web. In *Proc. of MMCN*, San Jose, CA, January 2000.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On Optimal Batching Policies for Video-on-Demand Storage Servers. In *Proc. of IEEE ICMCS*, Hiroshima, Japan, June 1996.

[3] K. V. Almeroth, M. H. Ammar, and Z. Fei. Scalable Delivery of Web Pages Using Cyclic Best-Effort Multicast. In *Proc. of IEEE INFOCOM*, San Francisco, CA, March 1998.

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. of ACM IMC*, San Diego, CA, October 2007.

[5] Y. Chai, Z. Du, and S. Li. A New Scheduling Algorithm for Distributed Streaming Media System based on Multicast. In *Proc. of ICDCS Workshops*, Beijing, China, June 2008.

[6] X. Cheng, C. Dale, and J. Liu. Statistics and Social Network of YouTube Videos. In *Proc. of IWQoS*, Enschede, Netherlands, June 2008.

[7] B. Cohen. Incentives to Build Robustness in BitTorrent. In *Proc. of P2PECON*, Berkeley, CA, June 2003.

[8] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In *Proc. of ACM Multimedia*, San Francisco, CA, October 1994.

[9] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and Efficient Merging Schedules for Video-on-Demand Servers. In *Proc. of ACM Multimedia*, Orlando, FL, November 1999.

[10] D. L. Eager, M. K. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery. In *IEEE Transactions on Knowledge and Data Engineering*, pages 742–757, October 2001.

[11] L. Gao, J. Kurose, and D. Towsley. Efficient Schemes for Broadcasting Popular Videos. In *Proc. of ACM NOSSDAV*, Cambridge, United Kingdom, July 1998.

[12] L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. In *Proc. of IEEE ICMCS*, Florence, Italy, June 1999.

[13] V. Gopalakrishnan, B. Bhattacharjee, K. K. Ramakrishnan, R. Jana, and D. Srivastava. CPM: Adaptive Video-on-Demand with Cooperative Peer Assist and Multicast. In *Proc. of IEEE INFOCOM*, Rio de Janerio, Brazil, April 2009.

[14] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proc. of ACM Multimedia*, Bristol, England, September 1998.

[15] K. A. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. In *Proc. of ACM SIGCOMM*, pages 89–100, Cannes, France, September 1997.

[16] C. Huang, J. Li, and K. Ross. Can Internet Video-on-Demand Be Profitable? In *Proc. of ACM SIGCOMM*, Kyoto, Japan, August 2007.

[17] G. Huang. Experiences with PPLive. In *Proc. of ACM SIGCOMM - P2P-TV Workshop*, Kyoto, Japan, August 2007.

[18] Y. Huang, T. Fu, D. M. Chiu, J. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proc. of ACM SIGCOMM*, Seattle, WA, August 2008.

[19] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[20] H. Ma and K. G. Shin. Multicast Video-on-Demand Services. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 31–43, 2002.

[21] S. Sheu, K. A. Hua, and W. Tavanapong. Chaining: A Generalized Batching Technique for Video-on-Demand Systems. In *Proc. of IEEE ICMCS*, Ottawa, Canada, June 1997.

[22] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms. *Springer*, 1998.

[23] D. Thaler, M. Talwar, A. Aggarwal, L. Vicisano, and T. Pusateri. http://tools.ietf.org/id/draft-ietf-mboned-auto-multicast-05.txt. *IETF*, October 2005.

[24] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting. *Multimedia Systems*, 4(4):197–208, 1996.

[25] C. Wu, B. Li, and S. Zhao. Diagnosing network-wide p2p live streaming inefficiencies. In *IEEE INFOCOM 2009*, Rio De Janeiro, Brazil, April 2009.