

# Server Selection in Large-Scale Video-on-Demand Systems

NIKLAS CARLSSON and DEREK L. EAGER

University of Saskatchewan

Video on demand, particularly with user-generated content, is emerging as one of the most bandwidth-intensive applications on the Internet. Owing to content control and other issues, some video-on-demand systems attempt to prevent downloading and peer-to-peer content delivery. Instead, such systems rely on server replication, such as via third-party content distribution networks, to support video streaming (or pseudostreaming) to their clients. A major issue with such systems is the cost of the required server resources.

By synchronizing the video streams for clients that make closely spaced requests for the same video from the same server, server costs (such as for retrieval of the video data from disk) can be amortized over multiple requests. A fundamental trade-off then arises, however, with respect to server selection. Network delivery cost is minimized by selecting the *nearest* server, while server cost is minimized by directing closely spaced requests for the same video to a *common* server.

This article compares classes of server selection policies within the context of a simple system model. We conclude that: (i) server selection using dynamic system state information (rather than only proximities and average loads) can yield large improvements in performance, (ii) deferring server selection for a request as late as possible (i.e., until just before streaming is to begin) can yield additional large improvements, and (iii) within the class of policies using dynamic state information and deferred selection, policies using only “local” (rather than global) request information are able to achieve most of the potential performance gains.

Categories and Subject Descriptors: C.4 [Computer Systems Organization] Performance of Systems

General Terms: Performance

Additional Key Words and Phrases: Performance analysis, modeling, video-on-demand, content distribution networks, server selection

## ACM Reference Format:

Carlsson, N. and Eager, D. L. 2010. Server selection in large-scale video-on-demand systems. *ACM Trans. Multimedia Comput. Commun. Appl.* 6, 1, Article 1 (February 2010), 26 pages.  
DOI = 10.1145/1671954.1671955 <http://doi.acm.org/10.1145/1671954.1671955>

## 1. INTRODUCTION

Popular video sharing systems, such as YouTube, are faced with enormous scalability problems. For example, in June 2006 it was estimated that YouTube served over 100 million video requests per day

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

Authors' addresses: N. Carlsson, D. L. Eager, Department of Computer Science, University of Saskatchewan, Saskatoon, SK S7N 5C9, Canada; email: {carlsson, eager}@cs.usask.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2010 ACM 1551-6857/2010/02-ART1 \$10.00

DOI 10.1145/1671954.1671955 <http://doi.acm.org/10.1145/1671954.1671955>

ACM Transactions on Multimedia Computing, Communications and Applications, Vol. 6, No. 1, Article 1, Publication date: February 2010.

and stored over five million videos, with an additional 65,000 videos being uploaded to the YouTube site each day [Gee 2006; USA Today 2006].<sup>1</sup> YouTube currently employs server replication, including use of one or more Content Distribution Networks (CDNs), to handle this heavy load. So as to maintain control of its content, YouTube aims to disallow downloading of videos, requiring users that wish to view the same video again to request it anew from a YouTube server.

It is not clear as to what extent such an approach to large-scale video on demand can economically scale up with increasing popularity, quality, and/or duration of the videos being streamed. YouTube, for example, imposes a 100MB limit on video file size. This is two orders of magnitude smaller than what would be required for a high-quality, full-length movie.

Server costs can be reduced by synchronizing the video streams for clients that make closely spaced requests for the same video from the same server, so that the server is delivering data from the same position in the video file at each point in time to each of the clients that have been so “batched” together [Aggarwal et al. 1996; Dan et al. 1994; Dykeman et al. 1986; Wong 1988]. This allows the cost of retrieving video data from disk, and other associated costs, to be amortized over multiple requests. In particular, each block of video data needs to be fetched from disk only once to serve all of the clients in the batch. In this article, use of unicast network delivery is assumed, but if multicast (IP or application level) was available and employed this approach could reduce network as well as server cost.

Batching of client requests so as to reduce server costs complicates the task of deciding where to forward each client request. Network delivery cost is minimized by selecting the *nearest* server, while server cost is minimized by directing closely spaced requests for the same video to a *common* server (regardless of the diversity of locations of the requesting clients). Intermediate policies might serve some requests at the closest server, and others at a common server along with other requests for the same video, depending on the current states of the servers and the currently pending requests.

Prior work on the server selection problem in content delivery systems has assumed individual rather than batched service [Carter and Crovella 1997; Jamin et al. 2000, 2001; Johnson et al. 2006; Qui et al. 2001; Ratnasamy et al. 2002; Zegura et al. 2000], or has considered some form of batched service, but with multicast delivery [Almeida et al. 2004; Fei et al. 2001; Guo et al. 2002]. Among these latter studies, Fei et al. [2001] consider systems in which a long-lived video stream is being multicast concurrently by replicated servers and the objective is to direct clients to servers so as to minimize the total network bandwidth usage. They show that the server selection problem in this scenario is NP-complete, and compare a number of heuristic policies. Guo et al. [2002] design and evaluate server selection techniques for replicated video-on-demand servers, each with a fixed number of channels. Several heuristic techniques are proposed and shown to outperform a basic policy that always directs requests to the closest server. Unlike Fei et al. [2001] and Guo et al. [2002] (but similarly to the work in this article), Almeida et al. [2004] assume that each server devotes *varying* resources, on demand, to the service of interest, rather than statically allocating fixed resources. They consider the problem of server placement and selection/routing with a weighted sum of network and server bandwidth usage as the objective function to be minimized, and show that, in the assumed context, use of multicast-based delivery techniques can result in optimal solutions that are very different from those for systems using nonbatched, unicast delivery.

Rather than proposing a specific server selection policy for the class of video-on-demand systems of interest in this article, we attempt only to determine the *type* of server selection policy likely to

<sup>1</sup>A complete wildcard (\*) search carried out in March 2008 found that approximately 78 million files were available at the site [www.youtube.com](http://www.youtube.com).

be the most suitable, recognizing that in practice the implemented policies are likely to have many details tailored to the specific system under consideration. We compare different classes of policies in the context of a simple, very abstract system model that allows us to accurately delimit the performance that may be achievable with policies from each class.

The policy classes considered range from those with simple policies, using very limited state information, to classes with more complex policies, requiring much more state information. While it is obvious that policies with more state information should be able to outperform policies with less state information (at least when neglecting the overhead of collecting such information), the magnitudes of the performance differences are not obvious a priori. It is important to determine these magnitudes so that, in general, design and implementation efforts can be focused on the simplest policy classes that are able to yield most of the potential performance improvements.

The first and most basic distinction that we consider is between server selection policies that use *dynamic* state information (for example, numbers of pending requests), and (simpler) policies that use only *static* (or semistatic) client-server proximity and average request rate information. Our results indicate that use of dynamic state information has the potential to yield large reductions in client startup delay, for fixed total delivery cost, in many cases by a factor of two or more.

Among policies using dynamic state information, a second distinction can be made between policies that *defer* server selection decisions until just before streaming is to begin, and those (simpler) policies that make a server selection decision immediately upon request arrival. We find that deferred selection can potentially yield substantial performance improvements, again by a factor of two or more in some cases, although only for fairly narrow ranges of model parameter values.

Finally, among policies using dynamic state information and deferred selection, a third distinction is between “local state” policies that base their server selection and scheduling decisions on the currently outstanding “local” client requests, and “global state” policies that use information concerning all current requests. We find that relatively simple local state policies appear able to achieve most of the potential performance gains.

The remainder of the article is organized as follows. Our system model is described in Section 2. Section 3 addresses the question of whether use of dynamic state information can yield major performance improvements. Section 4 considers the extent to which performance can potentially be improved in dynamic policies by deferring server selection decisions, rather than making such decisions immediately upon request arrival. Section 5 focuses on the class of policies using dynamic state information and deferred selection, and considers the extent to which policies using only “local” state information can realize the full potential of this policy class. Supporting evidence for our conclusions, obtained using more detailed network topology models, is presented in Section 6. Conclusions are presented in Section 7.

## 2. SYSTEM MODEL

We consider a video-on-demand system with  $N$  servers. Clients are divided according to network location into  $M$  groups, such that all of the clients in a group can be considered to have approximately the same network proximity to each server. For simplicity, in the following we assume that  $M = N$ . Given this assumption, the client groups and servers are indexed such that for the clients of group  $i$ , the nearest server is server  $i$ . Server  $i$  is called the “local” server for client group  $i$ , while all other servers are called “remote” servers for this group.

Each of the  $N$  servers is assumed to use a batching technique wherein the video streams for clients that make closely spaced requests for the same video are synchronized, so that the server is delivering data from the same position in the video file at each point in time to each of the clients in the batch. This synchronization is achieved by delaying service to some clients. A client cannot join a batch for which streaming of the video to the respective clients has already begun.

We assume that each server devotes *varying* resources, on demand, to video file delivery, rather than statically allocating fixed resources. This allows us to consider only the requests for a single representative video, and to focus on the trade-off between choosing the nearest server versus choosing a common server at which multiple requests for that video may be served concurrently, rather than additional load balancing issues. Requests for the representative video from the clients of each group  $i$  are assumed to be Poisson at rate  $\lambda_i$ , with the servers/groups indexed from 1 to  $N$  in nonincreasing order of the group request rates. Poisson arrivals have been observed in some measurement studies of video-on-demand systems (e.g., Almeida et al. [2001]). The performance of batching and related delivery techniques improves with more bursty arrival processes (e.g., Eager et al. [2000]). Therefore, performance models assuming Poisson arrivals will typically be conservative for burstier workloads.

It is assumed that client requests are for the entire video, and that once delivery to a batch of clients has commenced, each stream is continued (with synchronization maintained) until the respective client has received the entire video. Methods for handling situations where batches are broken apart (owing to client use of skip-ahead or fast-forward operations, for example), have been studied in prior work on batching (e.g., Dan et al. [1994, 1995]); we do not expect that consideration of such complexities would fundamentally change our conclusions regarding the server selection problem. It should also be noted that previous work has found VCR functionality to be used relatively infrequently by on-demand users in some contexts [Costa et al. 2004; Johnsen et al. 2007]. For example, Costa et al. [2004] observed that 89% of video clips, and 95% to 98% of audio streams delivered by two of Latin America's service and content providers did not require VCR operations at the server. Similarly, Johnsen et al. [2007] found that 99% of the news-on-demand sessions at Norway's largest newspaper did not result in any VCR operations.

The performance metrics that we consider are the maximum client startup delay, denoted by  $D$ , and the total delivery cost per request, denoted by  $C$ . The client startup delay (or waiting time) is defined as the time from request generation until streaming begins. We consider batching policies that impose a bounded client startup delay, yielding the metric  $D$ . The total delivery cost per request is given by the sum of the average *server cost* per request and the average *network cost* per request.

The server cost of streaming the representative video to a batch of clients is modeled as consisting of two components. One of these components includes costs such for retrieving the video data from disk, and is independent of the number of clients in the batch. This component is denoted by  $L$  in the following. The other component is assumed to be a per-client cost, and therefore scales linearly with the number of clients in the batch. Since this second component is a function only of the request arrival rates, and is therefore the same for all of the server selection and batching strategies considered here, it is neglected in the following.

The network cost of streaming the representative video to a group  $i$  client from a server  $j$  is given by  $c_{ij}L$ , where the constant  $c_{ij}$  expresses this cost relative to the per-batch server cost. Use of batching does not reduce the network costs, since we assume unicast delivery. For simplicity, in the following it is assumed, unless stated otherwise, that  $c_{ii} = 0$ , and  $c_{ij} = c$  for some  $c$  such that  $0 < c < 1$ , for all  $i \neq j$ . With the server cost of serving a single batch equal to  $L$ , this condition is satisfied whenever the network cost of retrieving video data from a remote server is less than the server cost of streaming the video. As evidenced in part by the growing interest in cloud/utility computing, for example, recent trends (such as the increasing importance of power expenditure, which could be expected to be much smaller for delivery over the current optical-fiber-based wide-area networks than for streaming video from disk at a server) suggest that this is a case of practical importance. (Note that for  $c \geq 1$ , it is always optimal to select the local server, while for  $c \rightarrow 0$  it is always optimal to select a common server.)

Table I. Notation

Symbol	Definition
$N$	Number of servers (assumed equal to the number of client groups)
$\lambda_i$	Video request rate from the clients of group $i$ ; groups indexed so that $\lambda_i \geq \lambda_j$ for $i \leq j$
$\lambda$	Total request rate, summed over all client groups
$L$	Server cost for streaming the video to a batch of clients
$\beta_i$	Rate at which batches of clients are served by server $i$
$c$	Network cost of streaming the video from server $j$ to a group $i$ client, $i \neq j$ , relative to the per-batch server cost
$r_i$	Average fraction of requests from group $i$ clients that are served by a server other than server $i$
$C$	Total delivery cost per request
$D$	Maximum client start-up delay (waiting time)

Using the notation in Table I, the preceding definitions and assumptions yield

$$C = \left( \sum_{i=1}^N \beta_i L \right) / \lambda + \left( \sum_{i=1}^N \lambda_i r_i \right) cL / \lambda, \quad (1)$$

where the first term gives the average server cost per request, and the second term gives the average network cost per request. The average server cost is equal to the total rate at which batches are served in the entire system (i.e.,  $\sum_{i=1}^N \beta_i$ ), times the server cost  $L$  associated with serving each batch, and divided by the total request rate  $\lambda$ . Note that the rate  $\beta_i$  that batches are served by a server  $i$  decreases as more requests are included in each batch. Similarly, the average network cost is equal to the rate at which requests are served remotely (i.e.,  $\sum_{i=1}^N \lambda_i r_i$ ), times the network cost  $cL$  associated with each such request, divided by the total request rate  $\lambda$ . Here,  $r_i$  is the fraction of requests from client group  $i$  that are served by a server other than server  $i$ .

In Section 6, this model is generalized to cases in which the number of servers  $N$  may be less than the number of client groups  $M$ , and for general  $c_{ij}$ , resulting in the second term in Eq. (1) becoming  $\sum_{i=1}^M (\lambda_i / \lambda) \sum_{j=1}^N r_{ij} c_{ij} L$ , where  $r_{ij}$  denotes the average fraction of requests from group  $i$  clients that are served by server  $j$ . The first term in Eq. (1) remains the same.

Note that there is a trade-off between the maximum client startup delay and the delivery cost. This trade-off arises from the fact that delivery cost is reduced with larger batch sizes, and that larger batches can be achieved by making clients wait longer for arrivals of other requests that can be served at the same time. In particular, assuming a fixed request load, the rate at which batches need be served monotonically decreases with the maximum client delay. Referring to Eq. (1), we note, for example, that if every request is served without delay at the local server, then  $C = L$ . In this case  $\beta_i = \lambda_i$ . In contrast, with a maximum startup delay  $D \rightarrow \infty$ ,  $C \rightarrow 0$  can be achieved by subjecting clients to arbitrarily long delays so as to create arbitrarily large batches of local requests at each server.

In light of this trade-off, policy comparisons can be carried out either by comparing maximum client startup delays for a fixed total delivery cost, or comparing total delivery costs for a fixed maximum client startup delay.

Although we assume unicast delivery, and our network cost model is not able to capture the network cost reductions possible with multicast, our work may still be applicable for some multicast environments. This is since the network cost reductions with multicast may be relatively small compared to the other cost components, owing to the relatively large number of nonshared links in a typical multicast delivery tree [Chuang and Sirbu 2001; Phillips et al. 1999; Fahmy and Kwon 2007].

### 3. DYNAMIC VS. STATIC POLICIES

This section compares server selection policies that use *dynamic* state information (for example, numbers of pending requests), and (simpler) static policies that use only client-server proximity and average



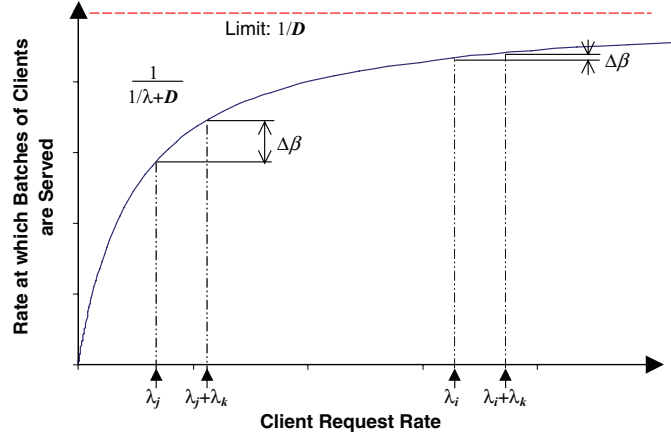


Fig. 1. Server load example.

request rate information in making server selection decisions. In a static policy either all requests from a given client group are served by the same server (in general, dependent on the group), or server selection is probabilistic. In either case, with Poisson requests from each client group, request arrivals at each server will also be Poisson. Section 3.1 reviews prior analysis results for a single server with Poisson request arrivals. In Section 3.2, these results are applied to determine a tight bound on the achievable performance with static policies. Section 3.3 accurately delimits the achievable performance with dynamic policies. Performance comparisons are presented in Section 3.4.

### 3.1 Analysis for Single-Server Systems

For a fixed maximum client startup delay (waiting time)  $D$ , the average server cost per request is minimized by a policy in which the server starts streaming to a batch of waiting clients whenever the waiting time of the client that has been waiting the longest reaches  $D$ . Since the average time duration from the beginning of service of one batch until the beginning of service of the next is  $D + 1/\lambda$  with this policy, the average server cost per request is given by the following expression [Carlsson et al. 2006; Tan et al. 2002].

$$\left( \frac{L}{D + 1/\lambda} \right) / \lambda \quad (2)$$

### 3.2 Delimiting the Achievable Performance with Static Policies

Note that in expression (2), the average server cost per request is a monotonically decreasing, convex function of the request arrival rate at the server. Thus, in a static policy, if all requests that are served by a remote server are served by the server with the highest rate of requests from its local client group (i.e., server 1), the average server cost per request will be minimized. This fact is illustrated in Figure 1, which shows the rate at which batches of clients are served for two servers  $i$  and  $j$ , with original loads  $\lambda_i$  and  $\lambda_j$ , both before and after their request load is inflated by the requests from a client group  $k$  with request rate  $\lambda_k$ . Clearly, the rate at which batches are served (and thus the rate at which server cost is incurred) increases less at server  $i$ , with the higher original load.

Furthermore, since we assume that  $c_{ij} = c$  for all  $i \neq j$ , serving such requests at server 1 incurs no greater network cost than serving them at any other remote server(s). Finally, the convexity of expression (2), and our assumptions regarding network costs, imply that in an optimal static policy

either all requests from a client group are served by a remote server (namely, server 1), or none is, and the former case can hold only if all requests from client groups with equal or lower request rate are also served remotely. Thus, in an optimal static policy there is an index  $k$  ( $1 \leq k \leq N$ ) such that all requests from group  $i$  clients, for  $i \leq k$ , are served by the local server, while all requests from group  $j$  clients, for  $j > k$ , are served by server 1. Note that for homogenous systems in which the client groups have identical request rates, in the optimal static policy either all requests are served by the local server or all requests are served at a common server.

Given the form of the optimal static policy, from Eq. (1) and expression (2) a tight lower bound on the total delivery cost per request that is achievable with a static policy, for fixed  $D$ , is given by

$$\min_{k=1,2,\dots,N} \left\{ \left( \frac{L}{D + 1/(\lambda_1 + \sum_{i=k+1}^N \lambda_i)} \right) / \lambda + \left( \sum_{i=2}^k \frac{L}{D + 1/\lambda_i} \right) / \lambda + \left( c \sum_{i=k+1}^N \lambda_i L \right) / \lambda \right\}. \quad (3)$$

Here, the first two terms (within the outer brackets) give the average server cost per request, and the third term gives the average network cost per request (as incurred owing to the requests from clients local to servers  $k + 1$  through  $N$ , which receive service remotely from server 1).

### 3.3 Delimiting the Achievable Performance with Dynamic Policies

Dynamic server selection policies use information about the current system state. To delimit the performance of such policies, we are interested in the best possible performance by policies with global state knowledge. Determining an optimal *online* dynamic policy (using a Markov decision process, for example) appears to be a difficult and perhaps intractable problem. For example, suppose that there is a waiting request from some client group  $i$ , when there is a remote server  $j$  about to begin streaming to some batch of local clients. The optimal choice between joining this batch and being served by the remote server, versus continuing to wait for a stream to be initiated at the local server, in general depends not only on the network cost  $c$  but also on the complete system state and on the statistics of the request arrival process. We are, however, able to accurately delimit the achievable performance with dynamic policies through a combination of results for optimal *offline* performance, with a given number of servers and client groups, and results for optimal offline performance in a limiting case as the number of servers and client groups grows without bound.

Consider first the problem of determining optimal offline performance with a given number of servers and client groups. An algorithm is developed that takes as input a request sequence (indicating both the arrival time and the client group of each request) and a maximum client startup delay  $D$ , and finds the minimum total delivery cost for serving all of the requests in the sequence. This algorithm is based on the following observation. Define the deadline of a request as the time at which the request waiting time would equal the maximum client delay. Then, at any request deadline  $t$ , the minimum network cost incurred by the corresponding request is determined solely by the service initiations that occur within the interval  $[t-D, t]$ , that is, from the request arrival time to its deadline. In particular, the minimum network cost is zero if and only if the local server starts service to a batch of clients during this interval, and otherwise is  $c$ .

This observation enables the following algorithm structure. First, a window of duration  $D$  is advanced through the given request sequence, with the right endpoint of the window moving at each advance to the next request deadline. Second, each potential choice of batch service initiations within the current window defines a “state”. When the window advances, the set of states changes, as earlier batch service initiations may now be outside of the window and some new batch service initiations may be added. Each state has an associated minimum total delivery cost (that expresses the minimum cost to have

reached that state, given the arrival sequence). Third, the cost of a new state (as created when the window advances) is calculated as the minimum of the costs of the alternative prior states (before the advance of the window) that result in this new state, *plus* the “transition” cost to get from the prior state to the new state. The transition cost between these two states consists of: (i) the network cost associated with the request whose deadline defines the right endpoint of the new window (according to whether or not the local server begins service to a batch in the new state), and (ii) the server cost for any new batch service initiations (associated with the new state). Finally, when the window advances to include the deadline of the last request in the request sequence, the minimum total delivery cost for the input request sequence and maximum client startup delay is given by the minimum over all current states of the associated total delivery cost.

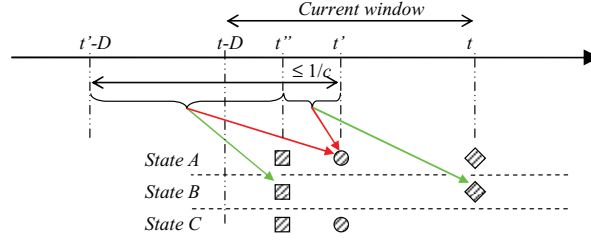
The feasibility of this approach depends on being able to tightly constrain the potential choices of batch service initiation times and locations, and thus the number of states associated with the current window, in a manner that still allows discovery of the minimum total delivery cost. Assuming Poisson arrivals, or any other arrival process for which no two events (request arrivals or deadlines) occur simultaneously, the constraints that we employ are as follows.<sup>2</sup>

- (1) A server *may* begin service to a batch of clients at a time  $t$ , *only if* time  $t$  is a request deadline.
- (2) Server  $i$  *may* begin service to a batch of clients at the deadline  $t$  of a client group  $i$  request, *only if* server  $i$  did not begin service to a batch of clients during the interval  $(t - D, t)$ .
- (3) Server  $i$  *may* begin service to a batch of clients at the deadline  $t$  of a client group  $j$  request, for  $i \neq j$ , *only if* there is no server  $k$  ( $k$  may equal  $i$  or  $j$ ) that began service to a batch of clients during the interval  $(t - D, t)$ .
- (4) Server  $i$  *may* begin service to a batch of clients at the deadline  $t$  of a client group  $j$  request, for  $i \neq j$ , *only if* there have been at least two arrivals of client group  $i$  requests in the interval  $(t - D, t)$  (and that thus could belong to the served batch).
- (5) *Some* server *must* begin service to a batch of clients at a request deadline  $t$  *if* there have been no batch service initiations in the interval  $(t - D, t)$ .
- (6) A server *may not* begin service to a batch of clients at a request deadline  $t$  *if*: (a) some server  $i$  previously began service to a batch of clients at the deadline  $t'$  of a client group  $i$  request, with  $t - D < t' < t$ ; (b) at most  $1/c$  arrivals of group  $i$  requests occurred in the interval  $[t' - D, t')$ ; and (c) the batch service initiation prior to the one at time  $t'$  occurred at a time  $t''$  with  $t - D < t'' < t' < t$ .
- (7) A server *may not* begin service to a batch of clients at a time  $t$  *if*: (a) some server  $i$  previously began service to a batch of clients at the deadline  $t'$  of a client group  $i$  request, with  $t - D < t' < t$ ; (b) the most recent deadline of a group  $i$  request previous to time  $t'$  occurred at a time  $t''$  with  $t - D < t'' < t' < t$ ; (c) at most one arrival of a group  $i$  request occurred in the interval  $(t'', t')$ ; and (d) no batch service initiation occurred at time  $t''$ , but this was not prevented by the constraints (and thus, there is a state in which server  $i$  begins service to a batch of clients at time  $t''$  rather than at  $t'$ ).

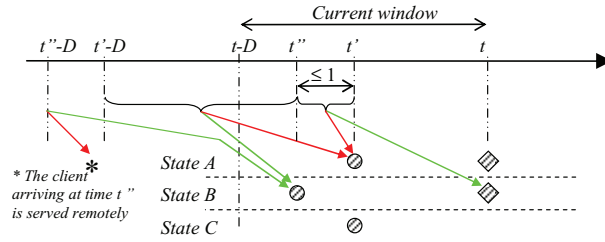
Constraints (1)–(4) specify when a server *may* serve a batch of requests, and constraint (5) specifies when *some* server *must* begin service. In the following we motivate why each of these constraints is valid. First, any service initiation not satisfying constraint (1) could be postponed, with no greater delivery cost. With respect to a schedule in which constraint (2) is violated, the request with deadline  $t$  could have been served with the earlier batch, and the new batch service initiation postponed, with no greater (and possibly reduced) total delivery cost. Similarly, when constraint (3) is violated, the request

<sup>2</sup>This list is followed by explanations of each of the constraints.





(a) Batch service initiations and deadlines distinguishing between the states involved in constraint (6). State A cannot yield a lower cost than state B. There is therefore no reason to explore state A further. State C, on the other hand, could result in a lower total cost, and must therefore be explored further.



(b) Batch service initiations and deadlines distinguishing between the states involved in constraint (7). Again, state A cannot yield a lower cost than state B, but state C requires further exploration.

Fig. 2. Illustration of constraints (6) and (7) used by the offline algorithm. Circles represent batch service initiations by server  $i$ ; squares and diamonds represent batch service initiations by other servers. Arrows are used to indicate with which batch each client  $i$  request is served. The tail of each arrow indicates the time (or time interval) at which the request(s) arrived. Any additional batch service initiations prior to time  $t$  that are not shown in the figures are identical for the compared states.

with deadline  $t$  could have been served earlier by server  $k$ , and the batch service initiation at server  $i$  postponed, with no greater cost. Constraints (1)–(3) imply that each server may begin service to a batch of clients at most once during any time period of duration  $D$ , and servers may begin service only at request deadlines. Note that there is never a benefit to initiating streams at the same server more frequently. When constraint (4) is violated, the batch could have been served by server  $j$  instead, with no greater total delivery cost. Finally, constraint (5) ensures that the startup delay of a client does not exceed  $D$ .

Constraint (6) essentially says that the decision to begin service to a batch of clients at time  $t'$  could be necessary in an optimal schedule (and thus the partial schedule including this batch service initiation possibly fruitful to pursue further), only if there is no batch service initiation at time  $t$ . To see this, note that in comparison to a schedule with batch service initiations at times  $t''$ ,  $t'$ , and  $t$ , the cost would be no greater if the batch service initiation at time  $t'$  had not occurred, and each of the requests that belonged to the batch served at time  $t'$  were served instead with either the batch at time  $t''$  or a batch at time  $t$ , which is possible owing to the time separation of at most  $D$  between time  $t''$  and time  $t$ . This constraint is illustrated in Figure 2(a).

Constraint (7) essentially says that beginning service to a batch of clients at server  $i$  at time  $t'$  and not at time  $t''$  could be necessary in an optimal schedule only if there is no batch service initiation at time  $t$ . To see this, note that in comparison to a schedule with batch service initiations at times  $t'$  and  $t$  but not at time  $t''$ , the cost would be no greater if each of the requests that belonged to the batch served at time  $t'$  and that arrived prior to  $t''$  are served instead by server  $i$  at  $t''$ , and the other requests that belonged to this batch are served instead at time  $t$ . This constraint is illustrated in Figure 2(b).

Table II. Average and Maximum Number of States Using the Optimal Offline Algorithm ( $N = 16$ ,  $c = 0.5$ ,  $L = 1$ ,  $\lambda_i = 1$  for all  $i$ )

$D(C)$	Constraints (1)–(5) only	Constraints (1)–(6)	Constraints (1)–(5), (7)	Constraints (1)–(7)
0.1 (0.6526±0.0006)	6.518±0.091 2,300	4.115±0.025 69	6.141±0.073 1,285	4.045±0.024 63
0.5 (0.4645±0.0006)	2,040±190 5,072,540	98.6±1.1 3,619	666±21 349,799	86.86±0.85 2,247
1.0 (0.3999±0.0008)	— >8,000,000	2,250±190 475,843	19,610±660 1,661,760	1,181±35 106,531
1.5 (0.3446±0.0007)	— >8,000,000	— >8,000,000	— >8,000,000	13,940±460 1,342,120

Although constraints (6) and (7) are somewhat more complex than the others, they can greatly reduce the number of states that need to be considered. This is illustrated in Table II, which shows 95% confidence intervals for the average number of states associated with the current window, and the observed maximum number, for algorithm variants using different subsets of the constraints, with  $N = 16$ ,  $c = 0.5$ ,  $L = 1$ ,  $\lambda_i = 1$  for all  $i$ , and various values of the maximum client delay  $D$ .<sup>3</sup> For each algorithm variant and value of  $D$ , 10 runs were performed, each on a different randomly generated request sequence with 25,000 request arrivals.

Although additional constraints are possible, at the cost of increased complexity in the implementation, constraints (1)–(7) were found to be sufficient to allow use of the optimal offline algorithm for a large portion of the parameter space. The algorithm can become too costly when  $D$  is large and  $1/c$  is not substantially greater than  $\lambda_i D$  (implying that there are many request deadlines, and thus many possible batch service initiation times, within a window, and that constraint (6) becomes less effective), and/or there are a large number of servers. Fortunately, in those cases in which the algorithm is too costly, consideration of a simple limiting case yields a lower bound on the total delivery cost that is empirically tight. To delimit the performance of dynamic policies we use a combination of the offline algorithm and this asymptotic bound.

Our asymptotic bound is derived by consideration of the case in which there are a sufficiently large number of servers and client groups, that whenever it would be optimal for a request to be served from a server other than the local server, the corresponding client is always able to join a batch of clients about to begin service from some remote server (prior to the request deadline). The minimum total delivery cost for this case can be determined with small computational cost by a variant of the optimal offline algorithm in which each server and its associated client group is considered in isolation, without explicit consideration of the remote servers. Note that this lower bound on the total delivery cost is tight not only when there is a sufficiently large number of servers, but also when almost all requests would be served by the local server in an optimal policy.

### 3.4 Performance Comparisons

In this section, we apply the results from Sections 3.2 and 3.3 to compare the potential performance with static versus dynamic server selection policies. Without loss of generality, the unit of cost is chosen to be the server cost of streaming the video to a batch of clients, and the unit of time is chosen to be the average time between requests from a client group when the total request rate is divided evenly among the client groups. With these choices of units,  $L = 1$  and  $\lambda = N$ .

We first consider the cost components for the optimal static policy separately, as these results will shed light into our subsequent comparative results in which only total cost is considered. Figures 3(a),

<sup>3</sup>The particular algorithm implementation used for these results could accommodate 8,000,000 current states.

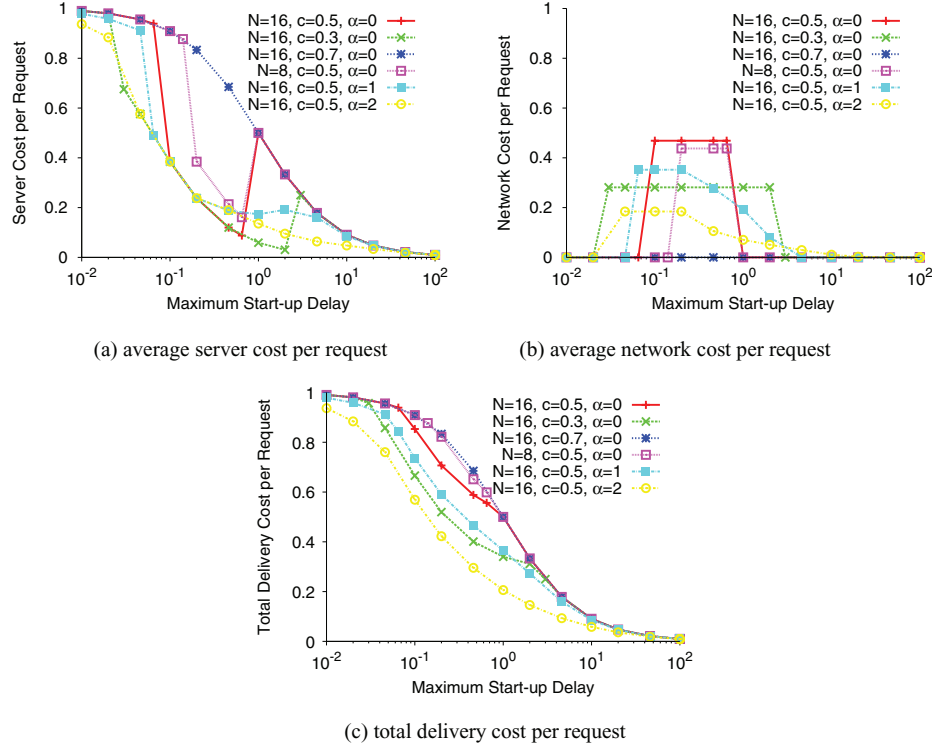


Fig. 3. Delivery costs for example scenarios with the optimal static policy.

3(b), and 3(c) show the average server cost per request, the average network cost per request, and the total delivery cost per request, respectively, as functions of the maximum client startup delay. Note that with our choice of time unit, the latter is given relative to the average time between requests from a client group when request rates are homogeneous; that is, a client delay of 2 means that the client delay is twice this average interarrival time. Further, the total server load, as measured by batch service initiations per unit time, with our normalized units can be found by multiplying the values in Figure 3(a) by  $N$ ; thus, curves for server load are identical (except for this scaling) to those shown in Figure 3(a). Results are shown for a number of example scenarios. The default scenario has  $N = 16$  servers,  $c = 0.5$ , and equal request rate  $\lambda/N$  from all client groups. Each other scenario differs in one characteristic. Two of the scenarios have heterogeneous client group request rates, in which the request rates are Zipf distributed with parameter  $\alpha$  and the request rate  $\lambda_i$  from client group  $i$  is equal to  $\Omega/i^\alpha$  with the normalization constant  $\Omega = N / \sum_{j=1}^N (1/j^\alpha)$ .

Note that for the homogeneous scenarios, either all clients or only group 1 clients receive service from server 1, and that in the former case the average network cost per request is  $c(N - 1)/N$ . The homogeneous scenario with  $c = 0.7$  has no values of the maximum client delay at which remote service is beneficial (with the optimal static policy). For the heterogeneous scenarios, for some maximum client delay values, some client groups receive service remotely at server 1, while others do not. For example, the curves for the scenario with  $\alpha = 1$  include three plotted points where all clients receive service at server 1, and three plotted points where client groups 3–16, 5–16, and 10–16, respectively, receive service remotely at server 1. As illustrated in Figure 3(c), the heterogeneous scenario with  $\alpha = 2$  achieves

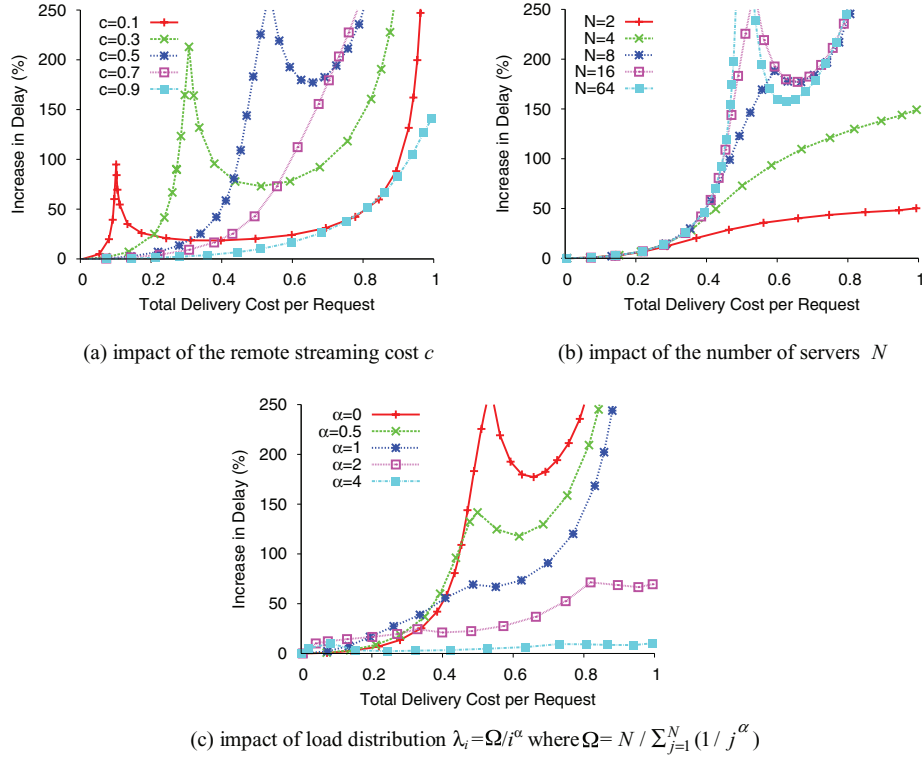


Fig. 4. Best potential performance with static policies relative to that with dynamic policies. Default parameters:  $L = 1$ ,  $c = 0.5$ ,  $N = 16$ ,  $\lambda_i = 1$  ( $\alpha = 0$ ).

the lowest total delivery cost per request, owing to its highly skewed request rates; in particular, approximately 63% of the requests are from client group 1 in this scenario.

Figure 4 compares the potential performance with static versus dynamic server selection policies. Rather than considering the minimum total delivery cost potentially achievable with a given maximum client startup delay, here we (equivalently) consider the lowest maximum client startup delay potentially achievable with a given total delivery cost. Specifically, this figure shows the lowest potentially achievable maximum client startup delay for static policies expressed as a percentage increase over that with dynamic policies, as a function of the total delivery cost per request. We have chosen this form for this and a number of subsequent figures, even though total cost is an output of our models rather than an input parameter, since we believe it may be most natural to view the server selection problem as one in which the goal is to improve the client-perceived performance (i.e., reduce the maximum startup delay) under a given cost constraint. This form is also most similar to that used in much prior resource scheduling work in which capacity or load is shown on the x-axis, with delay on the y-axis.

For the case of dynamic policies, the optimal offline algorithm from Section 3.3 was run on 10 randomly generated request sequences, each with 25,000 request arrivals, and the results averaged, for each set of parameters for which this algorithm was found feasible. For the other parameter sets, for which the optimal offline algorithm was not feasible, a similar methodology was followed, but using our asymptotic bound (in which each server and its associated client group is considered in isolation).<sup>4</sup>

<sup>4</sup>In this case, owing to the relatively low execution cost, each of the 10 runs for each parameter set had 200,000 request arrivals.

We use the same default scenario as before, with  $N = 16$  servers,  $c = 0.5$ , and homogeneous client group request rates, and vary one characteristic at a time. Figures 4(a) and (b) show results for homogenous request rate scenarios with  $c$  values between 0.1 and 0.9, and 2–64 servers, respectively. Figure 4(c) considers scenarios in which request rates are Zipf distributed with parameter  $\alpha$  (i.e.,  $\lambda_i \propto 1/i^\alpha$ ). For the case of  $\alpha = 0$ , all client groups have the same request rate; for  $\alpha = 4$ , in contrast, 92% of the requests are from client group 1.

The primary observation from these figures is that dynamic policies have the potential to yield substantially better performance than static policies over large regions of the parameter space. In a substantial number of cases, the lowest potentially achievable maximum client startup delay with static policies is over 100% higher than with dynamic policies; that is, higher by more than a factor of 2. These performance benefits are especially pronounced when  $c$  is between 0.3 and 0.7. While the peak benefit grows with the number of servers  $N$ , even for scenarios with only 4 servers the minimum achievable maximum startup delay with static policies is over 100% higher than with dynamic policies for a substantial region of the parameter space. As can be expected, the performance differences are the smallest for the highly heterogeneous scenarios, since there is little scope for improvement with complex policies when most requests are from a single client group.

Note the presence in many of the curves of a local maximum in the performance difference, at an intermediate value of the total delivery cost per request. As can be seen from Figures 3(b) and 3(c) for those curves whose parameter set is also used in those figures, these peaks correspond to points where the optimal static policy changes between one in which all requests are served by the local server, and one in which all requests are served by some common server. For the cases in which all client groups have the same request rate, the total delivery cost per request is approximately equal to the value of  $c$  at these points.<sup>5</sup>

It is possible to obtain the asymptotic limits of the curves in Figure 4 as the total delivery cost per request (in our normalized units) approaches one from the left, since in this case the delay  $D$  is so small that the probability that a client could be batched with more than one other client becomes negligibly small. The optimal static policy in this case is for each request to be served by the local server. The optimal dynamic policy in this case is for each request to be served by the local server if no previous request is waiting for service at the time of arrival of the request. In the rare event that there is such a previous request, the cost is minimized if the two clients are batched together. In the Appendix, these optimal policies are analyzed for the case of homogeneous client group request rates. The asymptotic limit of the percentage increase in delay with the optimal static policy, in comparison to that with the optimal dynamic policy, is derived as  $(N - 1)(1 - c) \times 100\%$ .

Note that Figure 4 shows the *potential* performance improvements with dynamic policies, but that these improvements may not be practically realizable. The next two sections consider the question of how complex a dynamic policy needs to be to achieve the full potential of this policy class.

#### 4. DEFERRED SELECTION VS. AT-ARRIVAL SELECTION

A basic distinction among dynamic policies is whether server selection occurs immediately when a request is made (“at arrival”), or whether server selection may be deferred for some period of time (at most, by the maximum client startup delay  $D$ ). Note that at-arrival server selection is a simpler approach, but deferred selection may offer the potential for improved performance, since server selection may take into account subsequent request arrivals. This section considers the question of how much

<sup>5</sup>Note that these peaks occur in regions of the parameter space in which the optimal offline algorithm is feasible; only well to the left of each peak did it become necessary to use the variant in which each server and its associated client group is considered in isolation.



performance can potentially be improved by use of deferred server selection, rather than at-arrival selection.

Section 4.1 determines an optimal at-arrival server selection policy, and a corresponding tight bound on the achievable performance with at-arrival server selection. Section 4.2 presents performance comparisons between these results and the results for general dynamic policies from Section 3.3.

#### 4.1 Delimiting the Achievable Performance with At-Arrival Server Selection Policies

Consider first the case in which all client groups have the same request rate  $\lambda_i = \lambda/N$ . If there are one or more clients waiting for service by server  $i$  when a new request from client group  $i$  arrives, the new client should join this batch. Suppose that there are no such waiting clients. Since all groups have the same request rate, in an optimal at-arrival policy a remote server would never be selected for a newly arriving request unless there is at least one client already waiting for service by that server, and thus the next client to begin waiting for service from server  $i$  can only be from group  $i$ . Therefore, if a remote server is selected for the new request, the same state with respect to client group and server  $i$  (a newly arriving group  $i$  request, and no waiting clients at server  $i$ ) will be entered again after a time of expected duration (with Poisson arrivals)  $1/\lambda_i$ , and a cost of  $cL$  will have been incurred. On the other hand, if server  $i$  is selected, the same state (a newly arriving group  $i$  request, and no waiting clients at server  $i$ ) will be entered after a time of expected duration  $D + 1/\lambda_i$  (the expected cost is minimized if a batch is not served until time  $D$  after formation), and a cost of  $L$  will have been incurred. Comparing these two scenarios, it is optimal to select the local server if and only if there is no remote server with at least one waiting client or  $(cL)/(1/\lambda_i) \geq L/(D + 1/\lambda_i)$ , or equivalently  $c \geq 1/(\lambda_i D + 1)$ ; otherwise, it is optimal to select such a remote server.

With the optimal at-arrival policy, if  $c \geq 1/(\lambda_i D + 1)$  all requests receive service from the local server, and the total delivery cost per request is given by  $L/(\lambda_i D + 1)$ . If  $c < 1/(\lambda_i D + 1)$ , the total delivery cost per request is given by  $\frac{L+cL(N-1)(\lambda/N)D}{\lambda D+1}$ , where the numerator gives the expected total cost to serve all of the clients in a batch, and the denominator gives the expected number of such clients. This expression uses the fact that there is one batch served per renewal period (of average duration  $D + 1/\lambda$ ), and a fraction  $(N-1)/N$  of all clients making requests during the time  $D$  (not including the client initiating the batch) receive service remotely.

Consider now the general case in which client groups may have differing request rates. Suppose that when a client group  $i$  request arrives there are no clients waiting for service by any server. Recalling that servers/groups are indexed from 1 to  $N$  in nonincreasing order of the client group request rates, analogously to the optimal static policy there is an optimal index  $k$  ( $1 \leq k \leq N$ ), such that for  $i \leq k$ , the new client begins a batch that will receive service by the local server, while for  $i > k$ , the new client begins a batch that will receive service by server 1.

For client groups  $i$  with  $2 \leq i \leq k$ , the optimal server selection policy is as described in the case of homogeneous groups; that is, decisions are based on the condition  $c \geq 1/(\lambda_i D + 1)$ . For requests from group  $i$  with  $i > k$ , it is optimal to select a remote server that already has a waiting client (if any), and otherwise to select server 1. Finally, consider requests from group 1. If there is at least one client that is waiting for service by server 1, or if there are no clients waiting for service by any server, it is optimal to select server 1. The case in which there are no previous clients that are waiting for service by server 1, but at least one client waiting for service by some remote server, is more complex than with homogeneous groups, however, when  $k < N$ . This added complexity is owing to the possibility of clients from other than group 1 beginning new batches to be served by server 1, which increases the desirability of selecting server 1 in this case. Note though, that when  $k < N$  we must have  $1 + \lambda_i D < \lambda_1 D$  for some client group  $i$  (namely, each group  $i$  for  $i > k$ ). (This observation comes from the fact that it can only be

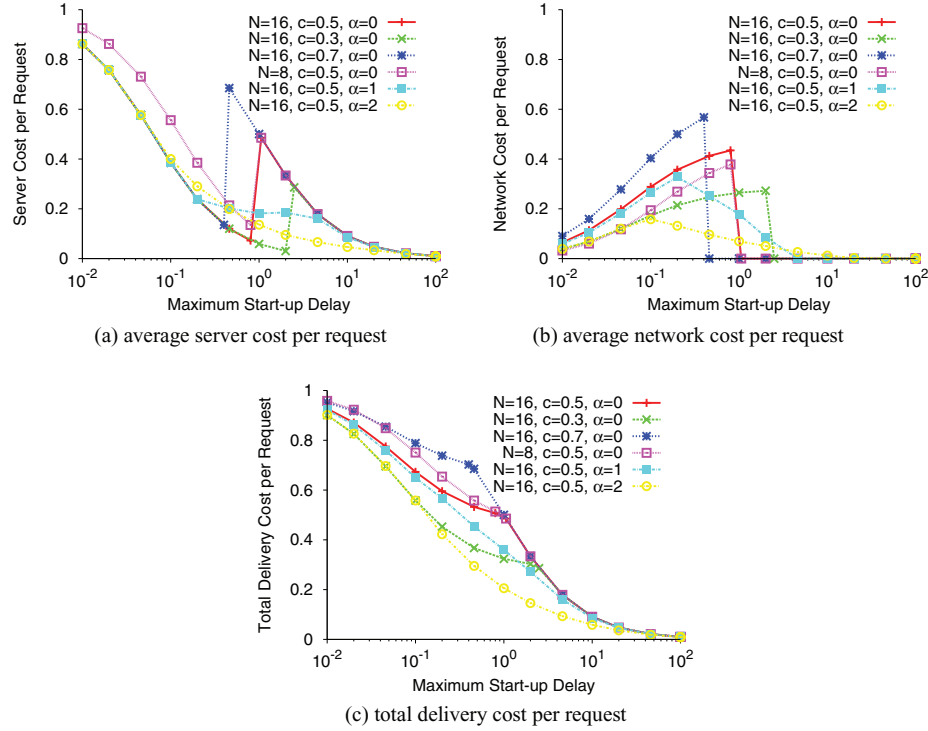


Fig. 5. Delivery costs for example scenarios with the optimal at-arrival policy.

desirable for a group  $i$  client to begin a new batch to be served by server 1, rather than by server  $i$ , if the expected number of group 1 clients that will be served in that batch exceeds the expected number of group  $i$  clients.) This observation implies that  $\lambda_1 D > 1$ , and therefore that  $c \geq 1/(\lambda_1 D + 1)$  for  $c \geq 1/2$ . Since it is even more desirable than with homogeneous groups to select server 1 for a newly arriving request from group 1, in the event that there are no previous clients that are waiting for service by server 1 but at least one client waiting for service by some remote server, for  $c \geq 1/2$  (and  $k < N$ ) we must have that it is optimal to select server 1 in this case. For the results shown in Section 4.2 for client groups with differing request rates, we have chosen  $c = 1/2$ , and simulation is used to determine the optimal index  $k$  and the minimum total delivery cost according to the optimal policy.

#### 4.2 Performance Comparisons

Similarly as in Section 3.4, we first consider the cost components for the optimal at-arrival policy separately, as these results will give insight into our subsequent comparative results in which only total cost is considered. Figures 5(a), 5(b), and 5(c) show the average server cost per request, the average network cost per request, and the total delivery cost per request, respectively, as functions of the maximum client startup delay. Again, the total server load, as measured by batch service initiations per unit time, with our normalized units can be found by multiplying the values in Figure 5(a) by  $N$ . Results are shown for the same example scenarios used for Figure 3. Comparing Figures 3(b) and 5(b), it is interesting to note that the optimal at-arrival policy is more able to fruitfully use remote service to achieve request batching for much smaller startup delays, and for higher values of  $c$  (e.g.,  $c = 0.7$ ) than the optimal static policy.

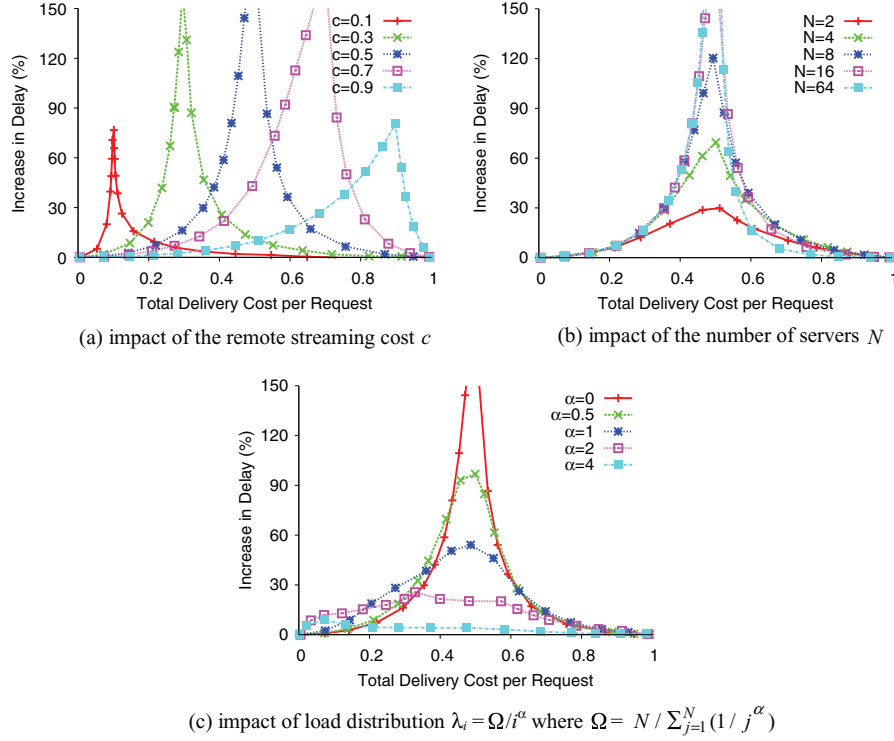


Fig. 6. Best potential performance with at-arrival policies relative to that with general dynamic policies. Default parameters:  $L = 1$ ,  $c = 0.5$ ,  $N = 16$ ,  $\lambda_i = 1$  ( $\alpha = 0$ ).

Figure 6 applies the results from Sections 4.1 and 3.3 to compare the potential performance with at-arrival versus general dynamic server selection policies, for the same scenarios used for Figure 4. Figures 6(a) and 6(b) show results for scenarios with homogeneous client group request rates, while Figure 6(c) shows results for heterogeneous request rate scenarios. It is apparent that use of deferred selection can potentially yield substantial performance improvements, by a factor of two or more in some cases, although only for fairly narrow ranges of model parameter values. In particular, large potential performance improvements are seen only when the total delivery cost per request is approximately the same as the network cost when a request is served remotely, equal to  $cL$ . In such regions, the potential performance improvements are maximized as the client groups become more homogeneous, as the number of servers and client groups increases, and for values of  $c$  (in our normalized units) between 0.3 and 0.7. As shown in Figures 5(b) and 5(c), in the case of homogeneous client groups, the point at which the total delivery cost per request equals  $cL$  is exactly the point at which the optimal at-arrival policy changes between one in which all clients receive service from the local server, and one in which some clients are served by a remote server.

## 5. LOCAL STATE VS. GLOBAL STATE

Dynamic server selection policies use information about the current system state. A key part of this information concerns what clients are waiting for service. We define “local state” policies as those that make server selection decisions for client group  $i$  requests, and service scheduling decisions for

server  $i$ , based on the currently outstanding group  $i$  requests. “Global state” policies, in contrast, are defined as those that use information concerning the current requests from other client groups, in addition to the local group. Note that both types of policies may also use other types of state information, in particular concerning the times at which servers are scheduled to serve batches. In general, local state policies require less state information and are simpler than global state policies. For example, global state policies may require information about all requests in the system to be sent to a central scheduler, or to every server in the system. In contrast, with the local state policy considered here, the other servers must only be informed about the time and server at which a batch service initiation has been scheduled. This section considers the question of how much performance can potentially be improved by use of global state server selection policies, rather than local state policies.

Section 5.1 describes a candidate local state server selection policy. Section 5.2 presents a candidate online global state policy. Section 5.3 compares performance with the local state policy to the limits on achievable performance from Section 3.3, and to the performance with the candidate global state policy. Section 5.3 concludes with a summary of our findings.

### 5.1 Candidate Local State Policy

In our candidate local state policy, server selection uses the following two rules. First, when a server  $i$  initiates service of a batch, all currently waiting group  $i$  clients are served. Second, when a remote server initiates service of a batch at a time  $t$ , a waiting group  $i$  client that requested the video at a time  $t_a$  receives this service *if*: (a) for each waiting group  $i$  client that requested the video at time  $t'_a \leq t_a$ , there are fewer than  $1/c - \lambda_i(t'_a + D - t)$  waiting group  $i$  clients with request times no earlier than  $t'_a$ ; *and* (b) there is no batch service initiation that has been scheduled (by time  $t$ ) at any server within the time interval  $(t, t_a + D]$ . This rule ensures that a group  $i$  client is served remotely only if it is unlikely that there will be at least  $1/c$  local clients with which it can be served locally, at its deadline, and it cannot defer its decision further without the risk of causing an additional batch service to be necessary. With at least  $1/c$  waiting local clients at the deadline it would be beneficial to serve a batch locally.

A batch service initiation is scheduled (for a time possibly in the future) at a server  $i$  whenever one of the following events occurs: (a) the waiting time of a group  $i$  client reaches the maximum duration  $D$ ; (b) a request arrives from a group  $i$  client, and the number of group  $i$  clients waiting for service reaches at least  $1/c$ ; *or* (c) a request arrives from a group  $i$  client when there is no future batch service initiation that has been scheduled at any server, and the number of waiting group  $i$  clients reaches at least  $\max[(2/3)(1/c), 2]$ . The motivation for scheduling a batch at the last of these events is to increase the likelihood that when batches are served with fewer than  $1/c$  clients from any one client group, the server that serves the batch is one for which there are a relatively large number of clients from the local client group.

When a server  $i$  schedules a batch service initiation, the time of this service initiation is chosen as  $t_a + D$ , where  $t_a$  denotes the earliest request time of the group  $i$  clients in the batch *if*: (a) there is a future batch service initiation that has been scheduled by some other server; (b) the most recent batch service initiation was by server  $i$ ; *or* (c) the most recent batch service initiation occurred later than time  $t_a$ . Otherwise, the time of the batch service initiation is chosen as the maximum of the current time, and  $t_{last} + D$ , where  $t_{last}$  denotes the time of the last batch service initiation at any server. While there is typically no advantage to initiating service of a batch earlier than at time  $t_a + D$ , use of this rule reduces the likelihood that some other server with fewer waiting local clients needs to start serving a batch to satisfy an earlier deadline.

## 5.2 Candidate Online Global State Policy

To give additional intuition for the potential performance differences between at-arrival policies and general dynamic policies, as well as between global state and local state policies, this section considers the performance achieved by *online* global state policies.

The specific global state policy presented here assumes that either all, or none of, the waiting clients for a client group  $i$  are served whenever service commences for a new batch at some server. Various policies in which some waiting group  $i$  clients are served as part of the batch, while other waiting group  $i$  clients are not, were investigated but not found to yield noticeable performance improvements.

Service to a batch of clients is initiated only when some request deadline is reached. When server  $i$  initiates service of a batch at a time  $t$ , all currently waiting group  $i$  clients receive this service. For any client group  $j$  with waiting clients at this time  $t$ , define  $t_j$  to be the time at which the longest waiting client from this group requested the video. All of the waiting group  $j$  clients are served as part of this batch if and only if, for each client group  $k$  with  $t_k \leq t_j$  (including  $j$ ) there are fewer than  $1/c - \lambda_k(t_k + D - t)$  waiting group  $k$  clients.

Whenever the deadline of a group  $i$  client request is reached, if there are at least  $1/c$  waiting group  $i$  clients in total then server  $i$  initiates service of a batch. If there are less than  $1/c$  waiting group  $i$  clients, the best candidates for the server that will serve a batch at this time (the second of these candidates may be server  $i$ ) include: (i) server  $j$ , for  $j$  such that there are at least  $1/c$  waiting group  $j$  clients, and among the waiting clients of such groups, a group  $j$  client is the one with the earliest upcoming deadline, and (ii) server  $k$ , for any  $k$  such that group  $k$  clients will receive service with this batch, and there is no other such client group with a greater number of waiting clients.

A choice is made between these two candidates, when both exist (note that the second candidate must exist), by estimating which candidate would result in the smaller delivery cost. With the first candidate, a batch that would otherwise have to be served by server  $j$  at a later point in time is served earlier, with the addition of other clients, and possibly reducing the total number of batches served at a cost saving of  $L$ . On the other hand, clients that make requests after the new service initiation time, but prior to when the batch could have been served, will now have to be served as part of some other batch; the expected cost increase is approximated by  $\delta_j C\lambda/N$ , where  $\delta_j$  denotes the time until the earliest upcoming deadline of the waiting group  $j$  clients and where  $C\lambda/N$  is the (measured) average per-server rate at which total delivery cost is incurred. Also, if the first candidate is chosen rather than the second candidate, the  $n_k$  waiting group  $k$  clients will receive service remotely rather than locally, with cost increase of  $n_k cL$ . The policy makes the first choice if the cost savings of  $L$  exceed the estimated cost increases, and otherwise makes the second choice.

## 5.3 Performance Comparisons

Figures 7(a), 7(b), and 7(c) show the average server cost per request, the average network cost per request, and the total delivery cost per request, respectively, as functions of the maximum client startup delay, for the candidate local state policy. The corresponding figures for the candidate global state policy are quite similar and are therefore omitted. Performance results for both policies were obtained by simulation, with the results for each parameter set taken as the average values over 10 simulation runs on randomly generated request sequences, each with 200,000 request arrivals.

As illustrated in Figure 7(b), the extent of use of remote service varies much more gradually, without abrupt transition points, with these deferred selection dynamic policies than with the optimal static and optimal at-arrival policies. This is due to the greater flexibility provided by use of dynamic, deferred selection. For example, unlike with a static policy, some requests from a particular client group may be served locally while others are served remotely, and unlike at-arrival policies, server choice may depend



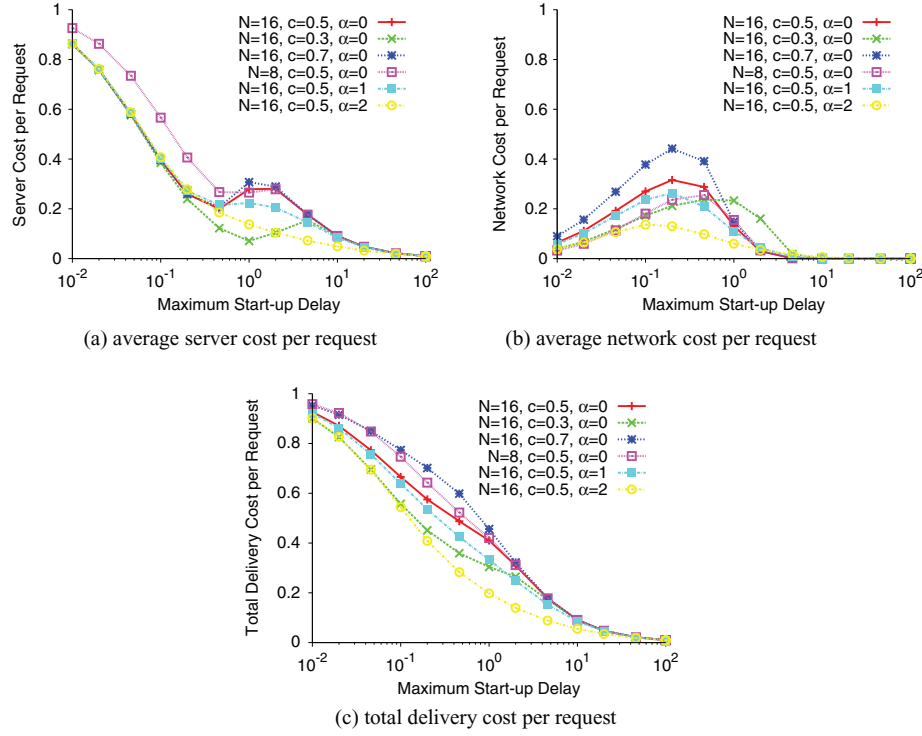


Fig. 7. Delivery costs for example scenarios with the candidate local state policy.

not only on just the at-arrival system states, but also on how state evolves over the near-term future, while the server choice is being deferred.

Figures 8 and 9 compare the performance of the candidate local state policy to the best potential performance with general dynamic policies, determined as described in Section 3.3, and to the performance of the candidate (online) global state policy, respectively. The same example scenarios are used as for Figures 4 and 6.

Interpretation of the results of Figure 8 is complicated by the fact that we delimit the best potential performance using the optimal *offline* performance. Thus, this figure leaves open the question of what portions of the performance gaps illustrated in the figure are owing to use of local state versus global state, and what portions are owing to use of online versus offline policies. Some insight into this question is provided by the results in Figure 9, which shows significantly smaller performance gaps between the candidate local state and candidate global state policies. The performance gaps between the candidate local state policy and the *optimal* online performance will be intermediate in size to the gaps shown in Figures 8 and 9. Based on these and other policy comparisons [Carlsson 2006], we conjecture that the performance gaps between the candidate local state policy and the optimal online performance are closer to those shown in Figure 9 than to those of Figure 8.

To summarize our findings thus far, performance comparisons among all of the various policy classes that we consider are presented in Figure 10, for various example parameter settings. These settings correspond to a subset of the example scenarios already considered in Figures 4, 6, and 8. Figures 10(a) and 10(b) show results for different network costs, Figures 10(c) and 10(d) for different numbers of servers, and Figures 10(e) and 10(f) illustrate the impact of heterogeneity. Although there are

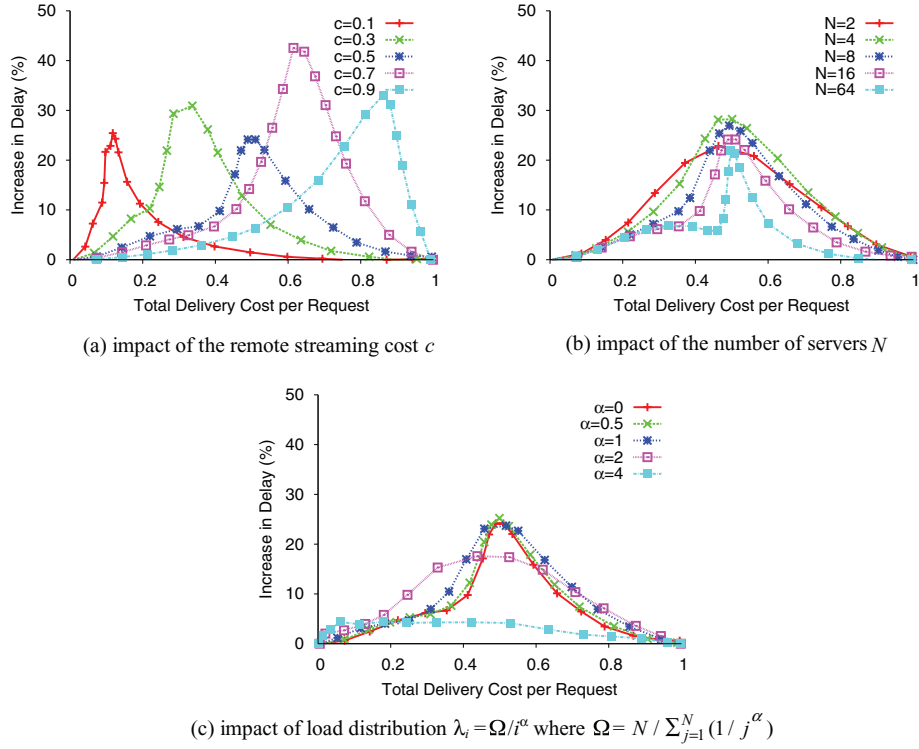


Fig. 8. Performance with local state policy relative to the best potential performance with general dynamic policies. Default parameters:  $L = 1$ ,  $c = 0.5$ ,  $N = 16$ ,  $\lambda_i = 1$  ( $\alpha = 0$ ).

substantial potential performance benefits to the use of dynamic rather than static server selection policies (as shown in Section 3), and deferred rather than at-arrival policies (as shown in Section 4), within the class of deferred, dynamic policies, simpler local (versus global) state policies appear able to achieve close to optimal performance.

## 6. EVALUATION USING MORE DETAILED TOPOLOGY MODELS

This section compares the policy classes defined previously using network topology models in which finer-grained proximity distinctions can be made than just “local” versus “remote.” Specifically, we use the GT-ITM topology generator [Zegura et al. 1996] to generate transit-stub network topologies, on which we evaluate candidate server selection policies from the considered policy classes. Each randomly generated topology consists of a transit (i.e., backbone) domain, and multiple stub domains. While such topologies do not model all the complexity of the Internet, they have been found useful in previous work. We associate one client group with each node located in a stub domain, and place either one or zero servers randomly within each stub domain. The client groups are assumed to have equal request rates, which yields heterogeneity with respect to the total request rate from client groups within each stub domain, owing to their differing numbers of nodes.

We consider first scenarios in which each stub domain has exactly one server. For this case, we can directly apply the static, at-arrival, local state, and online global state policies previously defined and used in our policy class comparisons, with the “local” server defined as the server in the same stub domain. Note, however, that the policies that were optimal with our simple model are no longer

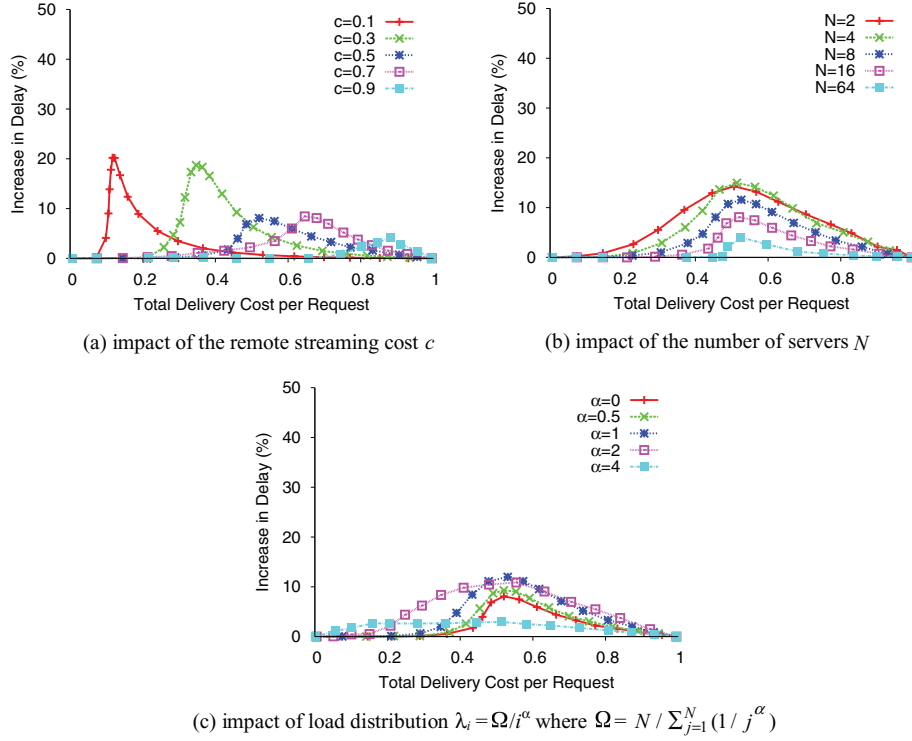


Fig. 9. Performance with local state policy relative to the performance with the candidate global state policy. Default parameters:  $L = 1$ ,  $c = 0.5$ ,  $N = 16$ ,  $\lambda_i = 1$  ( $\alpha = 0$ ).

optimal in this more complex setting, and are considered simply as “representatives” of their policy class.

Network costs depend on the distances between the servers and the client groups in the generated topologies. To capture the fact that routing across domain boundaries is typically more expensive than routing within domains, we assign a link cost for each intradomain link that is one-fourth of that for each interdomain link. Link costs are normalized so that the appropriate value of  $c$  when we apply our representative server selection policies is 0.5. Of course, now  $c$  gives only an average value, since the paths between different client groups and servers will, in general, have different costs (as given by the sum of the costs of the links on these paths).

For each of our representative policies, Table III shows the total delivery cost per request, average server cost per request (with  $L = 1$  as before), average network cost per request, maximum client startup delay, and percent increase in startup delay relative to the global state policy, for a number of scenarios in which each stub domain has exactly one server. Each of the results for the “ts96/12/0.5” topology class is an average from 10 simulation runs using randomly generated topologies with  $M = 96$  client groups spread over 12 stub domains; for each run 1,000,000 request arrivals were simulated. Similarly, each of the results for the “ts192/24/0.5” topology class is an average from 10 simulation runs using randomly generated topologies with  $M = 192$  client groups spread over 24 stub domains. During each simulation run, binary search was used to adjust the value of the maximum client startup delay (reported as an output of the simulation), so as to achieve the desired total delivery cost.

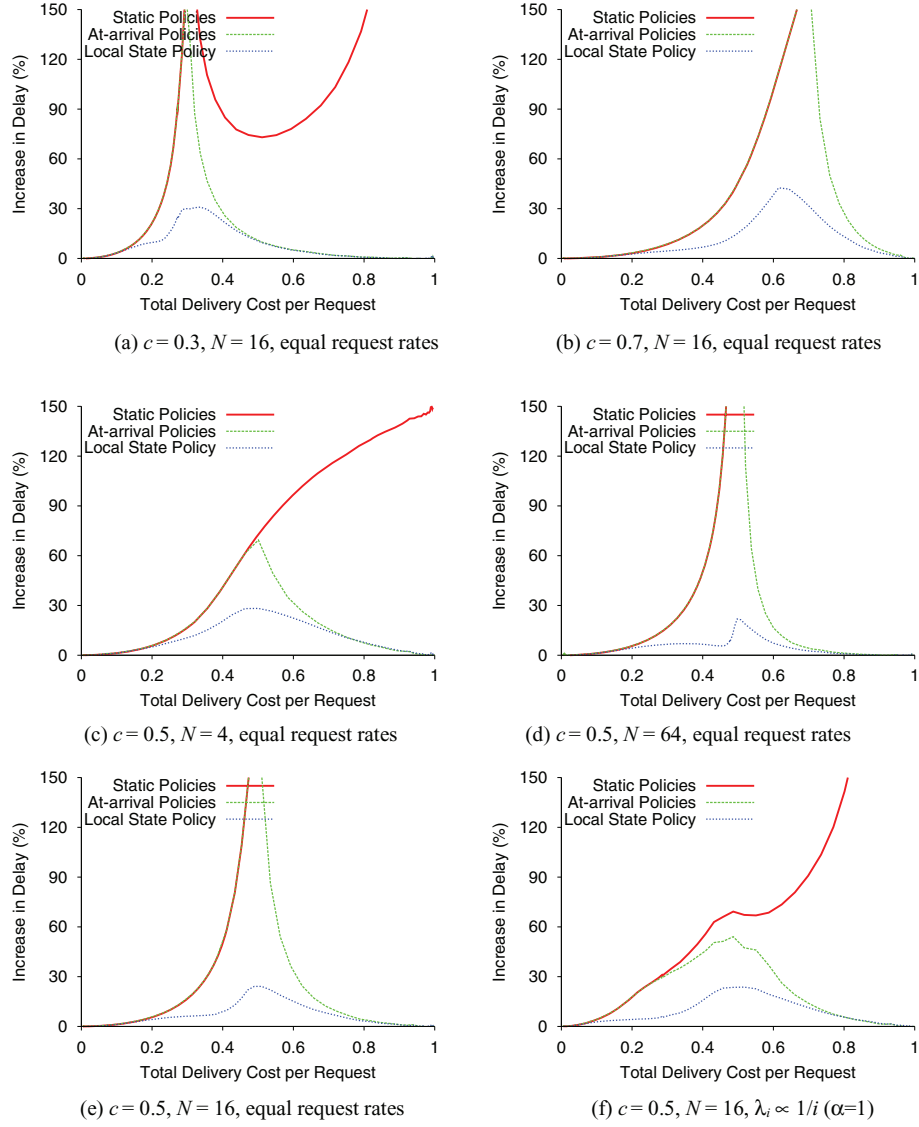


Fig. 10. Best potential performance with static policies and with at-arrival policies, and actual performance with a local state policy, relative to the best potential performance for general dynamic policies.

The results shown in Table III are consistent with our previous conclusions. For example, in the considered cases with “medium” (0.5) and “high” (0.8) total delivery costs, the static policy is significantly outperformed by the dynamic policies. Also, there are cases with significant performance differences between the at-arrival policy and the policies that defer selection decisions, while the performance differences between the local state and global state deferred selection policies are quite small.

We now consider scenarios in which not all stub domains have a server. As before, a server is considered “local” if it is in the same domain as the client group, and “remote” otherwise. However, there are now client groups that do not have any local server, requiring extensions of our representative policies

Table III. Example Scenarios with Transit-Stub Topologies Generated Using the GT-ITM Topology Generator

Topology class	Policy	Total delivery cost	Server cost	Network cost	Start-up delay	Increase in delay (%)
ts96/12/0.5	Static	0.3	0.231	0.069	3.22	12.98
	At-arrival	0.3	0.237	0.063	3.11	8.88
	Local state	0.3	0.231	0.069	2.86	0.19
	Global state	0.3	0.230	0.070	2.85	-
ts96/12/0.5	Static	0.5	0.389	0.111	1.23	117.36
	At-arrival	0.5	0.226	0.274	0.939	65.69
	Local state	0.5	0.227	0.273	0.599	5.64
	Global state	0.5	0.229	0.271	0.567	-
ts96/12/0.5	Static	0.8	0.337	0.463	0.165	164.34
	At-arrival	0.8	0.563	0.237	0.0648	3.59
	Local state	0.8	0.573	0.227	0.0643	2.85
	Global state	0.8	0.573	0.227	0.0625	-
ts192/24/0.5	Static	0.3	0.245	0.055	3.00	9.23
	At-arrival	0.3	0.251	0.049	2.92	6.26
	Local state	0.3	0.240	0.060	2.75	0.02
	Global state	0.3	0.240	0.060	2.76	-
ts192/24/0.5	Static	0.5	0.300	0.200	0.961	133.24
	At-arrival	0.5	0.110	0.390	0.772	87.47
	Local state	0.5	0.188	0.312	0.427	3.66
	Global state	0.5	0.189	0.311	0.412	-
ts192/24/0.5	Static	0.8	0.345	0.455	0.0821	157.66
	At-arrival	0.8	0.562	0.238	0.0326	2.17
	Local state	0.8	0.568	0.232	0.032	1.52
	Global state	0.8	0.568	0.232	0.0319	-

to accommodate such groups. In general we try to modify the policies as little as possible. For client groups with a local server, server selection operates as before. Server selection for each client group  $m$  without a local server is done quite similarly to that for the local client groups of the server closest to  $m$ .

Specifically, with our extended candidate static policy, a request from a client group  $m$  with no local server is directed to the same server that would serve a client request from a local client group of the server closest to  $m$ .<sup>6</sup> With our extended candidate at-arrival policy, a request from such a client group  $m$  is served by the closest server that has committed to serve a batch at the time of the request arrival, if any. If no such batch is scheduled, a batch is scheduled at the same server as would be chosen if the request was from a client group local to the server closest to  $m$ . With our extended candidate local and global state policies, the request is served with the first batch whose service is initiated while the respective client is waiting for service, such that there is no other batch whose service has been scheduled at a server closer to the client group and at a time prior to the request deadline. If there is no batch whose service is initiated while the client is waiting, the request is served at the request deadline by the same server as would be chosen if the request was from a client group local to the server closest to  $m$ . Note that in the case of the local state policy this would be the closest server itself.

Figure 11 shows performance comparisons among the candidate policies for scenarios in which either: (i) all stub domains have a server, or (ii) only half of the domains have a server. The same methodology and topology classes are used as used to generate Table III. With the exception that the cost to the closest

<sup>6</sup>When using expression (3), the ordering of the servers is according to the request rates of the local client groups only, but the expression is evaluated with the request rate of each client group  $m$  without a local server added to the request rate of the client groups that are local to the server closest to  $m$ .



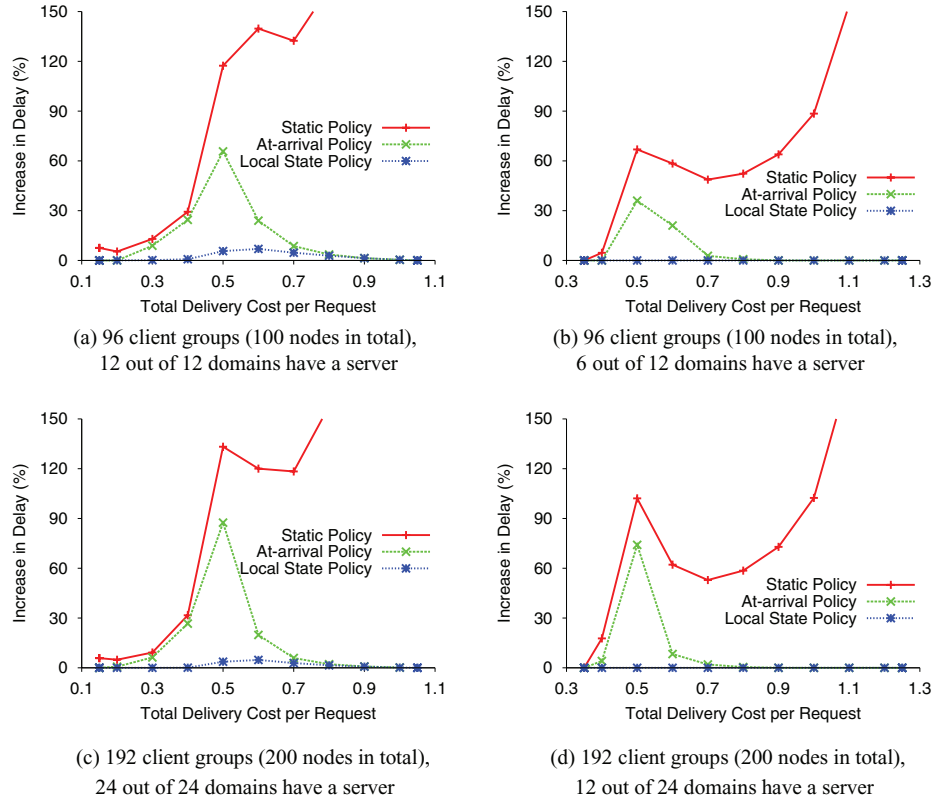


Fig. 11. Performance relative to the candidate global state policy for example scenarios using the GT-ITM topology generator.

server (and hence the minimum achievable delivery cost) increases for scenarios with fewer servers, and thus the performance differences among the policies decrease somewhat, the relative performance differences are similar to those observed previously. We recognize that the transit-stub topologies used here are not as complex as the topology of the Internet; yet, these topologies model what is found in the real world closer than our abstract model, which incorporates only first-order location information (local versus remote).

## 7. CONCLUSIONS

This article has considered the server selection problem in large-scale video-on-demand systems employing both server replication and request batching. Rather than proposing a specific server selection policy, different classes of policies are compared in the context of an abstract system model that allows us to accurately delimit the performance that may be achievable with policies from each class. The considered policy classes have differing complexities and require differing amounts of state information to be communicated among the servers. While it is obvious that policies with more state information are able to outperform policies with less state information, our results provide insights regarding the magnitudes of the potential performance benefits of more complex policies.

Our findings suggest that server selection using dynamic system state information (rather than only proximities and average request rates) can potentially yield large improvements in performance. Within the class of dynamic policies, use of deferred rather than at-arrival server selection has the potential to

yield further substantial performance improvements, although only for fairly narrow ranges of model parameter values. Finally, within the class of deferred, dynamic server selection policies, “local state” policies appear able to achieve reasonably close to the best possible performance. (These results are summarized in Figure 10.) Simulation results using Internet-based transit-stub topologies are provided as supporting evidence that these conclusions are more generally applicable.

Open problems include determination of the true optimal “online” performance, and bounding the performance of the different policy classes under more complex proximity and cost models.

## APPENDIX: ASYMPTOTIC ANALYSIS OF DYNAMIC VS. STATIC SERVER SELECTION

Assuming that the probability that a client could be batched with more than one other client is negligibly small, the optimal static policy is for each client to be served by the local server. Under this assumption, a client request for the video at a time  $t$  will result in a new batch that is served at time  $t + D$ , if and only if no other request for the video occurred within  $(t - D, t]$ ; otherwise, the new client will be batched with the already waiting client. Using this observation, the total delivery cost per request in a system with identical client group request rates (i.e.,  $\lambda_i = \lambda/N$ ) can be calculated as

$$C = e^{-(\lambda/N)D} L \approx L \left(1 - \frac{\lambda}{N} D\right), \quad (\text{A.1})$$

where a Taylor expansion has been used to obtain the final expression.

As described in Section 3.4, the optimal dynamic policy is for each request to be served by the local server if no other client is waiting for service at the time of the request. In the rare event that there is such a waiting client, the cost is minimized if the two clients are batched together. Similar to the preceding analysis, the total delivery cost per request can be calculated as

$$C = e^{-\lambda D} L + (1 - e^{-(\lambda/N)D}) \frac{N-1}{N} cL \approx L \left(1 - \lambda \left(1 - \frac{N-1}{N} c\right) D\right). \quad (\text{A.2})$$

Using the asymptotic approximations in (A.1) and (A.2), the maximum client startup delay  $D$  using the optimal static and the optimal dynamic policy, respectively, can be derived as

$$D_{\text{static}} \approx \frac{N}{\lambda} \left(1 - \frac{C}{L}\right); \quad D_{\text{dynamic}} \approx \frac{1}{\lambda(1 - c(N-1)/N)} \left(1 - \frac{C}{L}\right). \quad (\text{A.3})$$

The percentage increase in delay with the optimal static policy, in comparison to that with the optimal dynamic policy  $((D_{\text{static}} - D_{\text{dynamic}})/D_{\text{dynamic}} \times 100\%)$ , can then be easily derived as  $(N-1)(1-c) \times 100\%$ .

## REFERENCES

- AGGARWAL, C., WOLF, J., AND YU, P. 1996. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS'96)*. 253–258.
- ALMEIDA, J. M., EAGER, D. L., VERNON, M. K., AND WRIGHT, S. J. 2004. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE Trans. Multimedia* 6, 2, 356–365.
- ALMEIDA, J. M., KRUEGER, J., EAGER, D. L., AND VERNON, M. K. 2001. Analysis of educational media server workloads. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'01)*. 21–30.
- CARLSSON, N., EAGER, D. L., AND VERNON, M. K. 2006. Multicast protocols for scalable on-demand download. *Perform. Eval.* 63, 8–9, 864–891.
- CARLSSON, N. 2006. Scalable download protocols. Ph.D. thesis, University of Saskatchewan, Saskatoon, SK, Canada.
- CARTER, R. L. AND CROVELLA, M. E. 1997. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'97)*. 1014–1021.
- CHUANG, J. AND SIRBU, M. 2001. Pricing multicast communication: A cost based approach. *Telecomm. Syst.* 17, 3, 281–297.

- COSTA, C. P., CUNHA, Í. S., VIEIRA, A. B., RAMOS, C. V., ROCHA, M. M., ALMEIDA, J. M., AND RIBEIRO-NETO, B. A. 2004. Analyzing client interactivity in streaming media. In *Proceedings of the International World Wide Web Conference (WWW'04)*. 534–543.
- DAN, A., SHAHABUDDIN, P., SITARAM, D., AND TOWSLEY, D. 1995. Channel allocation under batching and vcr control in video-on-demand systems. *J. Parall. Distrib. Comput.* (Special Issue on Multimedia Processing and Technology), 30, 2, 168–179.
- DAN, A., SITARAM, D., AND SHAHABUDDIN, P. 1994. Scheduling policies for an on-demand video server with batching. In *Proceedings of the ACM International Conference on Multimedia (MM'94)*. 15–23.
- DYKEMAN, H. D., AMMAR, M. H., AND WONG, J. W. 1986. Scheduling algorithms for videotex systems under broadcast delivery. In *Proceedings of the IEEE International Conference on Communications (ICC'86)*.
- EAGER, D. L., VERNON, M. K., AND ZAHORJAN, J. 2000. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of the Annual Multimedia Computing and Networking Conference (MMCN'00)*. 206–215.
- FAHMY, S. AND KWON, M. 2007. Characterizing overlay multicast networks and their costs. *IEEE/ACM Trans. Netw.* 15, 2, 373–386.
- FEI, Z., AMMAR, M. H., AND ZEGURA, E. W. 2002. Multicast server selection: Problems, complexity and solutions. *IEEE J. Select. Areas Comm.* 20, 7, 1399–1413.
- GUO, M., AMMAR, M. H., AND ZEGURA, E. W. 2002. Selecting among replicated batching video-on-demand servers. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'02)*. 155–163.
- JAMIN, S., JIN, C., JIN, Y., RAZ, D., SHAVITT, Y., AND ZHANG, L. 2000. On the placement of Internet instrumentation. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'00)*. 295–304.
- JAMIN, S., JIN, C., KURC, A., RAZ, D., AND SHAVITT, Y. 2001. Constrained mirror placement on the Internet. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'01)*. 31–40.
- JOHNSON, F. T., HAFSØE, T., GRIWODZ, C., HALVORSEN, P. 2007. Workload characterization for news-on-demand streaming services. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC'07)*. 314–323.
- JOHNSON, K. L., CARR, J. F., DAY, M. S., AND KAASHOEK, F. 2006. The measured performance of content distribution networks. *Comput. Comm.* 24, 2, 202–206.
- LEE, G. 2006. Will all of us get our 15 minutes on a YouTube video? *The Wall St. J. Online*, 8/30/06.
- QIU, L., PADMANABHAN, V. N., AND VOELKER, G. M. 2001. On the placement of web server replicas. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'01)*. 1587–1596.
- PHILLIPS, G., SHENKER, S., TANGMUNARUNKIT, H. 1999. Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. (SIGCOMM'99)*. 41–51.
- RATNASAMY, S., HANDLEY, M., KARP, R., AND SHENKER, S. 2002. Topologically-aware overlay construction and server selection. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'02)*. 1190–1199.
- ROST, S., BYERS, J., AND BESTAVROS, A. 2001. The cyclone server architecture: Streamlining delivery of popular content. In *Proceedings of the International Workshop on Web Content Caching and Distribution (WCW'01)*. 147–163.
- TAN, H., EAGER, D. L., AND VERNON, M. K. 2002. Delimiting the range of effectiveness of scalable on-demand streaming. In *Proceedings of IFIP W. G. 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation (Performance'02)*. 387–410.
- USA Today. 2006. YouTube serves up 100 million videos a day online. *USA Today*, 8/16/06.
- WONG, J. W. 1988. Broadcast delivery. *IEEE* 76, 12, 1566–1577.
- ZEGURA, E. W., AMMAR, M. H., FEI, Z., AND BHATTACHARJEE, S. 2000. Application-layer anycasting: A server selection architecture and use in a replicated Web service. *IEEE/ACM Trans. Netw.* 8, 4, 455–466.
- ZEGURA, E. W., CALVERT, K., AND BHATTACHARJEE, S. 1996. How to model an Internetwork. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'96)*. 594–602.

Received April 2008; revised July 2008; accepted September 2008.