Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 50 (2006) 3608–3621
Computer Networks
www.elsevier.com/locate/comnet

Optimal server allocations for streaming multimedia applications on the Internet

Padmavathi Mundur *, Poorva Arankalle

Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, United States

Received 3 May 2005; received in revised form 2 March 2006; accepted 6 March 2006
Available online 5 April 2006

Responsible Editor: D. Hutchison

Abstract

In this paper, we address the server selection problem for streaming applications on the Internet. The architecture we consider is similar to the content distribution networks consisting of geographically dispersed servers and user populations over an interconnected set of metropolitan areas. Server selection issues for Web-based applications in such an environment have been widely addressed; the selection is mostly based on proximity measured using packet delay. Such a greedy or heuristic approach to server selection will not address the capacity planning problem evident in multimedia applications. For such applications, admission control becomes an essential part of their design to maintain Quality of Service (QoS). Our objective in providing a solution to the server selection problem is threefold: first, to direct clients to the nearest server; second, to provide multiple sources to diffuse network load; third, to match server capacity to user demand so that optimal blocking performance can be expected. We accomplish all three objectives by using a special type of Linear Programming (LP) formulation called the Transportation Problem (TP). The objective function in the TP is to minimize the cost of serving a video request from user population x using server y as measured by network distance. The optimal allocation between servers and user populations from TP results in server clusters, the aggregated capacity of each cluster designed to meet the demands of its designated user population. Within a server cluster, we propose streaming protocols for request handling that will result in a balanced load. We implement threshold-based admission control in individual servers within a cluster to enforce the fair share of the server resource to its designated user population. The blocking performance is used as a trigger to find new optimal allocations when blocking rates become unacceptable due to change in user demands. We substantiate the analytical model with an extensive simulation for analyzing the performance of the proposed clustered architecture and the protocols. The simulation results show significant difference in overall blocking performance between optimal and sub-optimal allocations in as much as 15% at moderate to high workloads. We also show that the proposed cluster protocols result in lower packet loss and latencies by forcing path diversity from multiple sources for request delivery.

Keywords: Streaming multimedia; Video-on-Demand; Transportation Problem; Optimal server clusters; Quality of Service; Performance analysis

* Corresponding author. Tel.: +1 410 455 3019; fax: +1 410 455 3969.
E-mail addresses: pmundur@csee.umbc.edu (P. Mundur), apoorva1@csee.umbc.edu (P. Arankalle).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.03.002

1. Introduction

Streaming applications for video such as pay per view and Video-on-Demand (VoD) have proliferated in the recent years spurred by the phenomenal increase, about 58% in the last year or two [1] in home-based Internet users. Industry-based streaming solutions consist of Real Networks' Real Media [2], Microsoft's Windows Media Services [3], Macromedia's Streaming Shockwave, and Apple's Quicktime Streaming [4]. Akamai [5] has moved closer to finding an efficient solution by using Content Delivery Networks (CDN) that put content closer to end users. However, we argue that moving the content to the edge alone will not work for multimedia applications where we use admission control as a mechanism to provide QoS. For such applications, capacity planning to realize better performance and resource utilization becomes a prominent issue. In this paper, we address system design issues to provide intelligent aggregation of server resources and request handling protocols that result in optimal blocking performance, server utilization and balanced loads. The analytical models and performance evaluation presented in this paper provide solutions to enhance the pragmatic approach provided by the CDNs. VoD-like applications require stricter QoS for longer duration than Web applications. Unlike Web servers, streaming servers are specially designed to deliver content in a predictable, delay-sensitive manner. These servers will use admission control and other related techniques to prevent overload and maintain QoS. Capacity planning to realize optimum server utilization in such an environment is the topic addressed in this paper.

In the spirit of CDNs, we consider metropolitan areas that are close together and content server nodes scattered throughout the area. We propose a special type of LP formulation called the Transportation Problem (TP) to obtain optimal server allocation to each metropolitan user population and aggregate those individual server nodes into clusters to serve each of the metropolitan areas. Organizing server clusters in this way provides distinct advantages: in an earlier work [31], we showed that a cluster of servers streaming different parts of the video to the client will result in better performance in terms of packet loss and latency. Using multiple sources that are physically separate, the data transfer is forced on multiple network paths and therefore, reduces the possibility of congestion. Employing a cluster of servers will also help reduce data granularity and achieve balanced load and fault tolerance. More importantly, organizing individual servers into clusters gives us a framework for monitoring blocking performance and detecting suboptimal server allocations when user demand changes.

We implement threshold-based admission control in individual servers within a cluster to enforce the fair share of the server resource to its designated user population. An arriving request will be rejected or *blocked* during admission control if there is no available capacity or if the number of requests from the user population exceeds its threshold. The blocking performance is used as a trigger to find new optimal allocations when blocking rates become unacceptable due to change in user demands and the current allocations are no longer optimal. In addition to providing such an architectural solution, we design and employ streaming protocols within server clusters for their efficient operation. The network context for our analysis is the best-effort IP-based networks. While we do not suggest any network related modifications, the proposed cluster architecture is expected to perform better by forcing path diversity in the network. Our work on streaming protocols and others suggest that such use of the network results in better performance in terms of packet loss and latency.

The TP formulation proposed in this paper is ideally suited for finding optimal server allocations for the server selection problem. The basic idea behind the TP formulation is to assign server capacities to user population demands by considering the cost of providing service from server i to a request from user population j. In mathematical programming problems where TP is applied, the cost is typically represented as distribution cost from manufacturing plant i to warehouse j. For our purposes, we use the round trip time (RTT) as the cost of serving a client request from server i to user population j. The network path metric RTT has long been used as a distance measure for proximity analysis in server selection for Web services. Using RTT as the cost metric in our TP formulation, the servers that are close to the user population are grouped together to provide service for that population.

The main contribution of this paper is the combined approach of providing a clustered server architecture based on optimal server allocation and protocols that efficiently operate those clusters. The proposed TP formulation in addition to addressing dynamic server selection problem, also

provides solution to capacity planning issues. A greedy or heuristic approach to server selection will not address the capacity planning problem for multimedia applications. The protocols proposed for operating server clusters make use of centralized and distributed approaches to handle requests resulting in better performance as shown through simulation experiments. One of the strengths of this work is its simulation. We model realistic Internet topologies using topology generators such as Tiers. We generate multimedia and background traffic using results from network monitoring tools such as Tstat. Using *ns* we implement an extensive simulation to evaluate the proposed clustered server architecture consisting of several server clusters obtained as a result of the application of TP. We demonstrate through simulation experiments, the blocking performance for optimal and suboptimal server allocations to user demands. We show that we can dynamically reconfigure the server clusters to match the changes in user demands by using the TP. The simulation results show significant difference in overall blocking performance between optimal and suboptimal allocations in as much as 15% at moderate to high workloads. The proposed cluster protocols are shown to result in lower packet loss and latencies by employing multiple sources and path diversity.

The paper is organized as follows. In Section 2, we present related work citing in particular research on modeling Internet topology and traffic distributions. In Section 3, we present the TP formulation for optimal server allocation and details on server cluster construction and reorganization. In Section 4, simulation setup and results are discussed. We conclude the paper in Section 5. In Appendix 1, we provide a brief description of the Transportation Simplex Method as background material.

2. Related work

Though companies like Akamai, Inktomi, or RealNetworks offer streaming applications in the Web context, we are not aware of any published works from them to indicate the use of analytical solutions we present in this paper. While they may offer low bitrate, Web-based, small interface streaming solutions, we address issues related to more persistent and pervasive VoD type of streaming application. For such applications, admission control and blocking performance become an essential part of their design. Whenever denial of service is used to prevent server overload for maintaining QoS, resource allocation issues become prominent. The analytical solution presented in this paper addresses the issue of matching server resources with user population demands in an optimal way. The distributed architecture by design results in an efficient use of the best-effort IP network.

Our work in this paper differs from others in that we provide an optimization technique for capacity planning. We also incorporate realistic Internet Topologies in our extensive simulation to analyze the performance of the clustered architecture. For Internet simulation, we use the Tiers topology generator mechanism proposed by Doar [20]. In [35], the authors compare two classes of Internet topology generators—degree-based versus structural—and conclude that Tiers topology generator adequately generates networks that mimic the hierarchical nature of the Internet when used for generating small scale regional networks like we do in this paper. Degree-based generators are more appropriate when generating large scale networks to reflect the power law nature of the Internet. Modeling Internet topology is also extensively discussed in [21–27]. Mellia et al. [21] present measurement techniques for traffic on the Internet using tools such as Tstat which we use in our own simulation. Research in [28] and others indicate the possibility of representing Web traffic using distributions such as M/Pareto for sessions and Poisson for arrival patterns. Trace collection for video traces is illustrated in Fitzek and Reisslein [29]. Optimization techniques including the TP are discussed in [32].

Much of server selection work is in the context of Web services and clustering refers to client groups being formed and directed to an appropriate Web server. For instance, in [6] a system called the *Webmapper* is presented to solve the problem of client assignment in a distributed system of content servers. Other research for server selection and load balancing in the context of Web services focuses on selecting from among a few mirrored servers and downloading in parallel different document segments [7–12]. Though these works are not directly relevant to streaming architectures, their measurement and probing strategies may be used in an actual deployment of our proposed architecture. For instance, in [13], a variety of probing strategies to measure and disseminate server and network path metrics with low overhead is suggested for Internet-wide service. Since in our architecture we employ RTT as a cost metric for service, tools like

IDMaps [14] or King [15] may be used to measure RTT between the client and the server in a real implementation. Tools such as IDMaps measure and disseminate distance information on the global Internet. Application level services can use this information to estimate the distances between any two IP addresses.

In [16], the authors present server selection techniques using Multiple Description Coding (MDC) video servers. They demonstrate the advantage of path diversity when combined with multiple sources and MDC. They use GT-ITM transit-stub model to generate the network topology. In [17], the authors present a framework for streaming video from multiple senders simultaneously to a single client. Similar to our prior work, they adopt this approach to force path diversity. However, their simulation environment and network configuration are limited. Same authors in an earlier work [18] employ a set of mirrored servers using rate allocation and packet partitioning algorithms to achieve high throughput. Other such multiple source streaming architectures can be found in the context of peer-to-peer networks, most notably the work on CoopNet [19] where clients actively participate in distributing the content and lessening the load on the server. There are a number of other works in the context of VoD that use hierarchical architectures and server models, including our own prior work [30]. Most of them do not explicitly model the network topologies or they use reservation-based protocols. Many of the resource allocation strategies in those works are based on greedy approach such as least loaded servers or redirection from one level to the next in a hierarchical architecture.

3. Distributed streaming architecture

Companies like Akamai push web content to the edge of the network by strategically placing servers such that the client can select a server with the shortest RTT. For our streaming architecture, we choose a similar environment consisting of several individual video servers geographically dispersed over a few interconnected metropolitan areas serving requests over the Internet. Our objective for this architectural setup is threefold: first, to direct client requests to a server that is close in network distance as measured by RTT. Second, to provide multiple sources so that different segments of the video can be streamed in quick succession from those sources to result in better performance; multiple sources also help in server load balancing and fault tolerance as well as path diversity on the network. Third, to match the server capacity to user demand so that optimal blocking performance can be expected. We accomplish the first and the third objective by providing a solution for server cluster formation using the TP formulation. We propose protocols to operate the cluster of servers to achieve the second objective.

Prior to formulating the TP, we use naturally occurring metropolitan areas to define our user group or population. The individual servers are assumed to be geographically dispersed in the metropolitan region under consideration. The TP formulation will provide the technique to match the aggregate demand from user groups to the overall server capacity. All servers serving the same user group will form a cluster and client requests originating from that user group will be directed to one of the servers in the cluster. It is possible that a server is shared between multiple user groups in which case, each user group has access only to its allocated capacity on that server.

An organization of clusters of servers for three metropolitan regions is shown in Fig. 1. In our proposed model, this organization is the result of solving the TP. Cluster c_1 consists of servers 1, 2, 3 and serves user community x; cluster c_2 consists of servers 3, 4, 5 and serves user community y. Notice that server 3 appears in both clusters which means that its capacity is shared between user communities x and y. Three other servers in cluster 3 serve requests from user community z.

3.1. Optimal server allocation model

Using the TP formulation, we allocate individual servers to meet the demands of the user groups. The result of this optimal allocation of server capacity is the server clusters, each cluster designated to serve a user population. We quantify three parameters to formulate the transportation matrix: server capacity, user community demand, and the cost of serving a request from ith server to jth user community. User group demand is estimated using the number of active clients: each client requiring one stream capacity from the server. Server stream capacity is determined as the number of simultaneous requests the server can serve based on its disk bandwidth. Lastly, we define the cost matrix based on server's proximity to user community using RTT values. The model parameters are shown in Table 1.

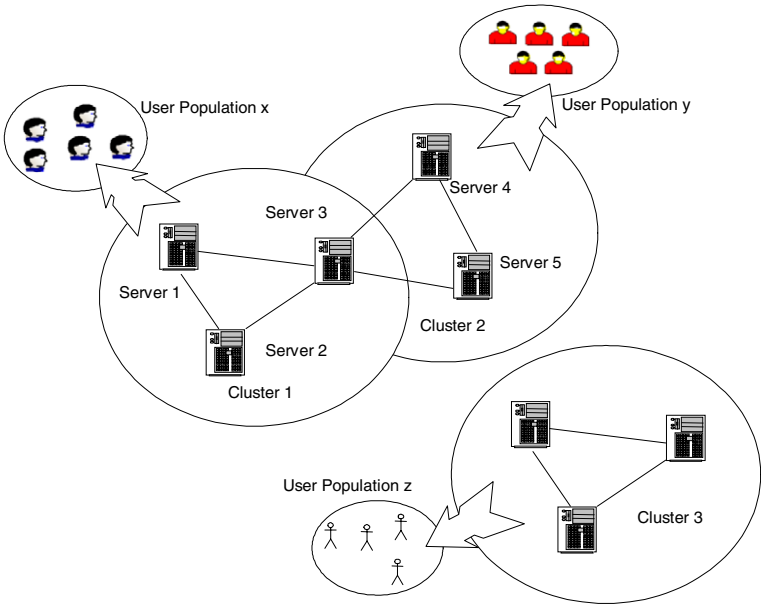

Fig. 1. Clustered server architecture.

Table 1
Transportation problem model parameters

m	Number of servers
n	Number of user groups
s_i	ith server stream capacity (number of requests)
d_j	jth user group demand (number of requests)
c_{ij}	Cost of serving a request from server i to user community j

Let Z be the total allocation cost and $x_{ij}(i = 1,2,\ldots,m;\ j = 1,2,\ldots,n)$ be the number of stream capacity *allocated* between server i and user group j; the linear programming formulation of this problem becomes

$$\text{Minimize} \quad Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij} = s_i \quad \text{for } i = 1,2,\ldots,m,$$

$$\sum_{i=1}^{m} x_{ij} = d_j \quad \text{for } j = 1,2,\ldots,n, \quad \text{and}$$

$$x_{ij} \geqslant 0 \qquad \text{for all } i \text{ and } j.$$

As seen in the TP formulation above, both server capacity, s_i and user group demand, d_j are expressed as the number of simultaneous requests at a known average bitrate: for the server, it is the disk bandwidth which will determine the overall capacity; for the user group, the estimated number of users wanting to use the service at the same time will constitute the overall demand. This type of request to request matching using average bitrate is a simplification. Within the TP formulation, it is easy to represent both server capacity and user group demand in units of Mbps which will allow us to represent requests of different bitrate. The cost c_{ij} is the average RTT value between pairs of user communities and servers. We find *average* RTT values by taking a random sampling of RTT values from client nodes in each group to individual servers by periodically probing the network to get these measurements.

Average RTT as the cost metric in our model is well justified. As Internet applications have proliferated in recent years, so has the number of mirrored Web servers that can serve a client request. Research on server selection problem for Web services typically redirect a client request to the nearest server as determined by the packet delay between the host and the client. RTT is used as a basic measure of latency between two Internet hosts by using *ping* or *traceroute*. Since many applications rely on such proximity information, proposals for Internet-wide services for providing that information have resulted in tools like IDMaps [14]. The proposed architecture in this paper will use RTT as a distance measure to find nearest servers but it also extends the idea to include server capacities in a common framework using the TP formulation.

Table 2
Transportation matrix

	Group 1	Group 2	Group 3	Max capacity
S1	276	350	399	95
S2	265	426	475	70
S3	329	274	383	40
S4	457	298	511	40
S5	417	422	301	55
Demand	100	100	100	300

Table 3
Optimal allocation

Server	Group 1	Group 2	Group 3
S1	30	20	45
S2	70	0	0
S3	0	40	0
S4	0	40	0
S5	0	0	55

Table 2 is an illustration of the average RTT-capacity-demand matrix used for TP formulation. The TP matrix shown is called a balanced transportation problem where we have the maximum capacity equal to the overall demand. The aggregate server capacity is 300 simultaneous requests. The user demand as shown in Table 2, is assumed to be the same for each user group and adds up to the same as the maximum server capacity. This is not a limiting assumption; the requests from the user group can be generated in any other proportion as illustrated in the simulation later. Optimal server allocation to each user group is shown in Table 3 found after applying the Transportation Simplex method to solve the TP [see Appendix 1 and [32] for details on solving TP].

Server clusters generated from TP formulation in Table 2 are:

- S1 and S2 serving User Group 1;
- S1, S3, and S4 serving User Group 2;
- S1 and S5 serving User Group 3.

This allocation will be used later in the simulation to analyze blocking performance.

3.2. Server admission control

As is often the case with multimedia applications, each server in the cluster will implement admission control to prevent overload. A capacity based admission control scheme will deny service to an incoming request if there are not enough server resources to serve the request. In addition, the servers in our clustered architecture implement threshold-based admission control and block incoming requests beyond a threshold. Threshold-based admission control scheme allows us to share a server among different types of requests by limiting access to each type of request within a threshold capacity. We presented an analytical model for computing blocking probabilities under threshold-based admission control in [33]. In our clustered server architecture, it is possible that a server is shared by two or more user groups in which case we need to implement a sharing policy. For instance, server S1 in Table 3 is shared among three clusters each serving requests from user groups 1, 2, and 3; requests from each group are limited to the optimal allocation of the capacity of server S1 for that group. For instance, requests beyond a threshold of 30 from user group 1 will be blocked even if there is available capacity in S1. The idea is to conserve capacity for requests from groups 2 and 3 to provide them their fair share. In general, the admission control test at server i is represented as follows. An arriving request from user group j will be admitted at server i if both of these conditions are met:

$$\mathrm{Ncurrent}_j + 1 \leqslant \mathrm{Nallocated}_j$$

(threshold-based admission control),

$$\sum_{j=1}^{n} \mathrm{Ncurrent}_j + 1 \leqslant \mathrm{Ncapacity}$$

(capacity-based admission control),

where $\mathrm{Ncurrent}_j$ is the number of current requests at server i from user group j, $\mathrm{Nallocated}_j$ is the number of requests allocated to server i for user group j, Ncapacity is the maximum capacity at server i. In general, the threshold-based rule will suffice in our model where the allocated capacities in a server add up to its overall capacity. However, during the transition phase as explained in the next section, capacity-based rule has to be applied to make sure that the servers are not overloaded.

3.3. Dynamic cluster reorganization

In addition to enforcing sharing, admission control helps us to quantify blocking performance. We monitor the blocking performance with respect to user groups and use that as a guide to maintain optimal allocation of servers to user groups. Blocking performance worse than an acceptable level is seen

as an indicator of change in user demand or network conditions. This will be used as a trigger to reorganize the server clusters. We run the TP algorithm again during reorganization but with changed user demand to find new optimal allocations. New allocations take effect dynamically by a change of admission control policies in each server.

An important issue to consider during reorganization is the handling of requests that are already in service if there are differences in allocation before and after reorganization at a particular server with respect to a user group. Let N_{old} be the number of requests from the old allocation prior to reorganization and N_{new} be the number of requests to be allocated to the server after reorganization. Two cases arise at each server:

1. The old allocation is less than or equal to the new allocation; $N_{old} \leqslant N_{new}$.
2. The old allocation is more than the new allocation. $N_{old} > N_{new}$.

$|N_{old} - N_{new}|$ requests denote the maximum number of requests from old allocation that could have already been received at the server and in progress at the time of reorganization. For case 1, this number is 0 and for case 2, the number lies between 0 and $|N_{old} - N_{new}|$.

In the first case, the transition from old allocation to new allocation is smooth and can be assumed immediately because the server capacity is not overcommitted. In the second case, all or some of $|N_{old} - N_{new}|$ requests may still be in service at the server. An instantaneous transition to the new organization will disrupt service for those requests from the old allocation. A transition phase to allow for the completion of previously scheduled requests is essential. During this phase, the server in question will be overcommitted in excess of a maximum of $|N_{old} - N_{new}|$ requests and therefore blocking will continue to rise. Reduced blocking due to reorganization is seen only after a certain time lapse; in fact, we will see the benefits of new configuration after $|N_{old} - N_{new}|$ requests have been completed. Later in our simulation, we show this behavior through experiments. No reorganization is triggered during the transition phase to prevent oscillations.

As an alternative to the transition phase, those requests from old allocation could be migrated to the servers in the new allocation. However, we do not consider that option in our analysis because of additional overhead and possible disruption in QoS incurred in managing such migration of requests already in progress.

3.4. Cluster protocols

We present in this section two protocols for request handling in the server clusters. The first protocol, called the Centralized Control Protocol (CCP) uses a centralized load distribution algorithm. All client requests are received by the central server, which distributes them evenly within the cluster using the global state information about server loads. In the second protocol called the Distributed Control Protocol (DCP), a token passing scheme is employed to split and distribute *each* arriving client request equally across the servers in the cluster. The distribution achieves diffusion of the network traffic across several different routes. We evaluate both protocols using average packet loss and latency as the metrics. Our simulation results show a significant decrease in packet loss and latency with the two protocols. These protocols were first introduced in [31].

3.4.1. Centralized control protocol (CCP)

In this protocol, a central server maintains the global state and is aware of the load on each server as communicated periodically by individual servers. The precision of the load information in the central server depends on how often the video servers send messages about their load information to the central server. Irrespective of the streaming protocol which is usually UDP, TCP is used for interserver communication to prevent loss of messages from the video server to the central server and vice versa. On an incoming request for a video stream, the central server forwards the request to the least loaded server in the cluster. This protocol therefore, can be characterized as a *single-stream single-server* protocol as exactly one server serves each client request in contrast to the DCP protocol described next. Fig. 2a illustrates this protocol. If a video server fails to send any message about its state information to the central server, it is assumed to be inactive. No client requests are forwarded to the video server until it becomes active again and resumes sending status messages.

3.4.2. Distributed control protocol (DCP)

Most centralized solutions suffer from the overhead of having a single point of failure. Limited resource availability at the central server can also be an overhead under high load. On the other hand,

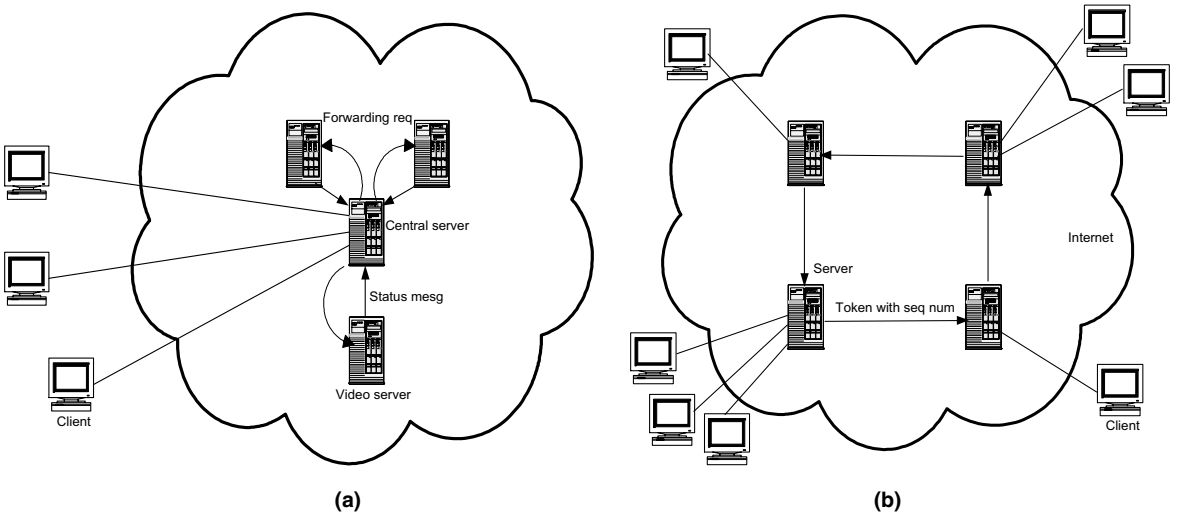

Fig. 2. (a) CCP protocol, (b) DCP protocol.

they have the advantage of a simple implementation. To contrast the CCP protocol, we present a second protocol that uses a distributed control architecture. DCP employs a token passing scheme to split and distribute each client request across the servers in the cluster; request to an individual server now refers to a segment of the same video. Each server in the cluster would then stream the appropriate video segment to the client (see Fig. 2b).

Let the number of servers in a cluster be K. When a new client request is received by a server, it breaks up the requested stream into K segments. Let each segment take n seconds to stream. Each segment has a sequence number i, $1 \leqslant i \leqslant K$. The server which receives the request will stream the first segment. A period D seconds before it completes transmission of the first segment, it hands over control to another server, chosen at random. The second server processes the second segment, and hands over control to a third server, and so on. While forwarding the request, a sequence number and a server-list are also forwarded. The *sequence number* indicates the segment that is to be processed by the next server. The *server-list* contains a list of servers which have already processed segments of this stream. While choosing a new server, a server not on the server-list is chosen. While forwarding the request, each server appends its id to the server-list. This protocol can be characterized as *multiple-server*, single-stream since a single stream is served in several segments and each request is processed by several servers serving request segments. When a new server does not respond, a different server not present in the server-list is chosen. In case servers not in *server-list* fail to respond, a server already in *server-list* will be chosen. The protocol can be described by the two principal events shown below.

Event1: Connection request from client to a server.

1. Start processing segment 1, ($i = 1$).
2. D seconds before completion, send to a server chosen at random, {Connection Req, Seq num $i + 1$ (next segment), server-list (server-ids)}.

Event2: (Connection req, seq num i, server-list}) received from another server.

1. Start processing segment i.
2. Update server-list to server-list + server id.
3. If ($i = K$), end of stream. Otherwise,
4. D seconds before completion, choose a server not in server-list, and forward {Connection req, seq num $i + 1$, server-list} to the server.

The advantage of the DCP protocol is a greater distribution of load over the network and its lower packet loss as seen later in simulation results. However, the disadvantage is that it works best when all servers have equal capacities so that an incoming request can be equally distributed among them. When individual server capacities are different in a cluster, a combination of both DCP and CCP has to be implemented.

4. Performance evaluation

We present details of the simulation of the proposed streaming architecture in this section. Performance is evaluated using blocking measurements, packet loss and latency as evaluation metrics. The architecture for our simulation study consists of five video servers labeled S1,...,S5. The delivery network is the Internet, consisting of hierarchical structure of LANs, MANs, and WANs. The video streams are served using UDP but control messages between server nodes are sent using TCP. The five servers are clustered with respect to each user population using the transportation problem (see Tables 2 and 3). Within a cluster, CCP or DCP protocol as appropriate is used. Threshold-based admission control will enforce the rightful share of a server for each user population. We generate both multimedia streaming traffic and background traffic to emulate realistic network topologies. The evaluation metrics used are request blocking percentage at server nodes per user group, average packet loss and packet latencies. We first present results on blocking analysis and show performance with and without server reconfiguration. In Section 4.2, we present performance analysis for the cluster protocols, CCP and DCP. All results are averaged over five trials; confidence interval analysis for 95% confidence level is conducted over the sets of five experimental values. Each trial is run for a length of 300–500 h simulation time with average requests per hour ranging from 500 to 1200. This combined with 5 replications gives us enough samples to generate a statistically significant average performance. The graphs are shown with error bars that show half-widths of the confidence intervals. The error bars are quite narrow as seen in the graphs indicating that the results we present are statistically significant.

We use ns-2.26 [34] for implementing the simulation. The Internet topology is simulated using Tiers1.1 topology generators. Tiers is found to be an appropriate topology generator when modeling regional or geographically limited networks as we do in our analysis [35]. Realistic values for different network parameters such as nodes/LAN, LANs/MAN, and MANs/WAN are used. In order to model background traffic, we note a few general characteristics of Web traffic: traffic on the Internet is characterized by a large portion of TCP traffic and a relatively smaller proportion of UDP traffic. TCP traffic, notably, Telnet and FTP, have a Poisson arrival pattern. Telnet and FTP sessions can be modeled by a Pareto distribution with heavy tail. Background traffic can be modeled as superpositions of ON/OFF periods expressed using Pareto distributions [28]. We use all of the above factors to generate realistic background traffic. Results from Tstat, a network monitoring tool described in [21] are used to provide quantitative information regarding the proportion of TCP and UDP flows. Results from Doar [20] are used to apply various values for the network topology. The intranetwork redundancy parameter refers to the number of edges between nodes of the same type. The simulation parameters are shown in Table 4.

4.1. Simulation results—blocking analysis

We construct three experiments for analyzing the blocking performance for the distributed streaming architecture. The metrics used are the percentage of blocked requests from each user population and for the overall system.

- For the first experiment, we use the transportation matrix shown in Table 2. The combined stream capacity of the 5 servers is 300 requests. We assume that each user population generates requests equally in 1:1:1 proportion. The purpose of this experiment is to illustrate optimal resource allocation from TP and determine blocking performance.
- For the second experiment, we use the transportation matrix shown in Table 5. The combined stream capacity of the servers is still 300 requests.

Table 4
Simulation parameters

Parameter	Value
Number of WANs	$N_w = 1$
Number of MANs/WAN	$N_m = 3$
Number of LANs per MAN	$N_l = 15$
Number of WAN nodes	$S_w = 5$
Number of MAN nodes	$S_m = 10$
Number of LAN nodes	$S_l = 20$
Intranetwork redundancy	$R_w = 3$, $R_m = 2$, $R_l = 1$
Internetwork redundancy	$R_{mw} = 3$, $R_{ml} = 1$
Total number of nodes	Approximately 1000
Total number of links	Approximately 2200
Link capacity	10 Mbps
Number of servers	5
Streaming request duration	Average 15 min; exponential
Streaming request arrival pattern	Poisson distribution
Multimedia traffic	100 kbps CBR
Background traffic	M/Pareto ON/OFF periods
Simulation time	300–500 h

Table 5
Transportation matrix

	Group 1	Group 2	Group 3	Max capacity
S1	279	360	372	95
S2	269	398	448	70
S3	333	290	357	40
S4	461	311	485	40
S5	421	394	273	55
Demand	80	140	80	300

However, the mix of requests from the three user populations is changed to 1:1.75:1 proportion. The purpose of this experiment is to show that the allocation from Table 3 is no longer optimal as indicated by high overall blocking.

- For the third experiment, we find the optimal server allocation for the change in demand and show that the blocking performance goes down with reconfigured server clusters. We also use this experiment to explain the transition between old and new allocations.

The RTT values for the cost matrix in TP formulation are found using a random sampling of individual RTT values between pairs of client and server nodes by implementing the *ping* utility in our simulation. The RTT values shown in Tables 2 and 5 are on the high side as these values represent the average over a wide range of samples from clients. Within the transportation matrix RTT values are however, used to represent the distance or proximity of a client to a server node. While absolute values have no impact, they collectively represent a cost for providing service between a client and a server node. A more accurate measurement will be needed if RTT values are used as indicators of network congestion.

In the first experiment, we analyze blocking performance using the optimal allocation from Table 3. Fig. 3a shows the blocking performance for the overall system. The number of requests per hour range from 600 to 1200 and average duration of each request is 15 min generated using exponential

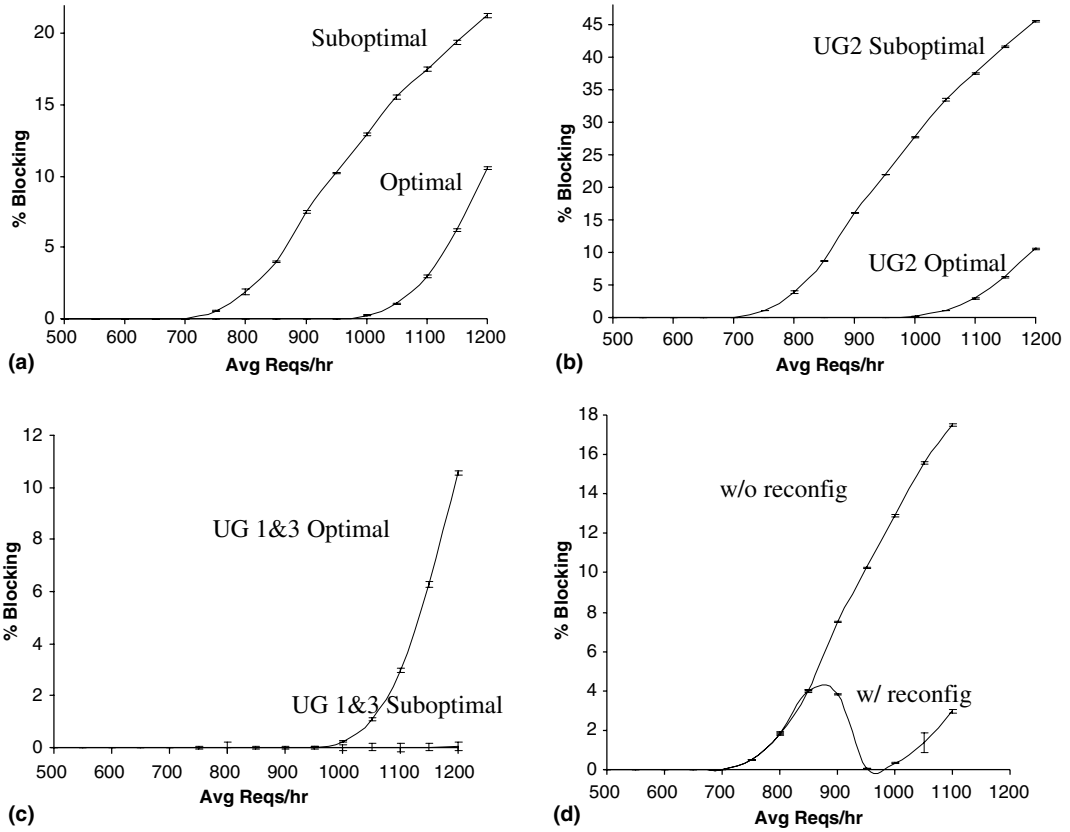

Fig. 3. (a) Overall blocking performance for the clustered architecture, (b) blocking performance for user group 2, (c) blocking performance for user groups 1 and 3, (d) blocking performance with and without reconfiguration.

distribution. The system shows very low blocking until about 1100 requests per hour after which it increases steadily. Fig. 3a and b show blocking performance for user groups 1, 2, and 3. Notice that in Table 2, we set up the demand for all user groups to be equal and therefore, we expect that their blocking performance will also be the same as illustrated in Fig. 3a and b for the curve labeled *optimal*.

Note: Performance curve labeled *optimal* indicates favorable server clusters; performance curve labeled *suboptimal* indicates server clusters no longer catering to the change in demand.

The results of the second experiment where we change the user demand are shown by the curve labeled *suboptimal* in Fig. 3a–c. The performance shows the need for server reconfiguration to find new optimal allocation. As seen in Table 5, the demand from user groups 1 and 3 go down from 100 to 80 and the demand from user group 2 goes up by about 40. Within the simulation, we generate requests in the following proportion, 1:1.75:1, with user group 2 generating 1.75 times more requests than user groups 1 and 3. The RTT cost matrix is also updated using the newly probed values. As seen from Fig. 3a–c, the allocation from Table 3 is no longer optimal and results in high blocking for user group 2. Blocking for user groups 1 and 3 is close to 0 as the servers allocated to them are underutilized. Overall blocking is also high as shown in Fig. 3a (curve labeled *suboptimal*) and 3d (curve labeled *w/o reconfig*).

The third experiment involves running the transportation solution on the new matrix in Table 5 resulting in the allocation shown in Table 6, with the old allocation shown in parenthesis. Fig. 3d shows blocking performance with and without server reconfiguration. Reconfiguration is triggered when the overall blocking performance exceeds a threshold; in our simulation, we use 3.5% blocking at 850 streams per hour as the threshold to trigger reconfiguration; if the allocation is optimal, blocking at this workload should be 0 as shown in Fig. 3a.

Fig. 3d shows the performance comparison between old and new allocation. Without reconfiguration and using the old allocation, some of the servers are over utilized and some others underutilized resulting in high blocking. The reconfiguration being optimal for the new demand pattern should result in decreased blocking. However, we observe a transition phase where some of the requests from old allocation are still in progress and need to be completed before the new allocation takes effect as explained in Section 3.3. During this transition phase, blocking continues to rise and finally, starts to decrease. This behavior is demonstrated in Fig. 3d. Without reconfiguration, overall blocking at a workload of 1100 requests/h is at 17.5% whereas with reconfiguration blocking at the same workload is brought down to 2.35%.

Table 6
New optimal allocation

Server	Group 1	Group 2	Group 3
S1	0 (30)	95 (20)	0 (45)
S2	70	0	0
S3	10 (0)	5 (40)	25 (0)
S4	0	40	0
S5	0	0	55

4.1.1. Transition phase analysis

A snapshot of server requests in progress at servers S3 and S1 at the end of 300 h with 850 streams per hour when the reconfiguration is triggered is shown in Fig. 4a and b. Server S3 under new allocation has fewer requests from user group 2 than it had before. However, requests allocated from previous configuration are still in progress and as a result

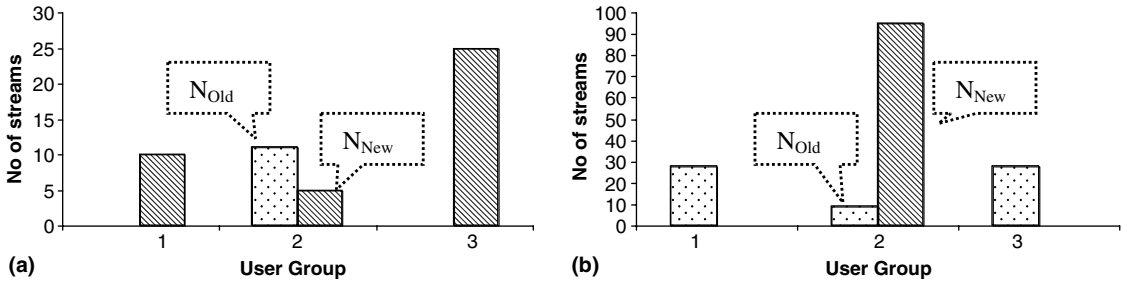

Fig. 4. (a) Snapshot of new allocation and requests in progress from old allocation at Server 3. (b) Snapshot of new allocation and requests in progress from old allocation at Server 1.

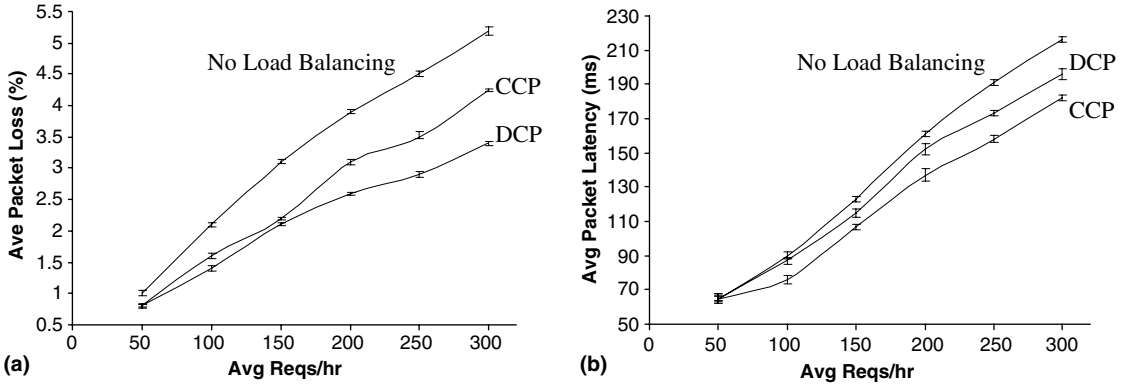

Fig. 5. (a) Packet loss with cluster protocols. (b) Packet latencies with cluster protocols.

blocking of over committed newly arriving requests goes on at this server. This explains the hump in Fig. 3d. During the transition phase, we could migrate requests at the over committed server to other underutilized servers. Such request migration is not considered in this paper since that will entail migrating requests already in progress and the overhead incurred as a result and the possibility of disruption of QoS. A snapshot of Server S1 in Fig. 4b shows the new allocation to be higher than the old allocation for user group 2. However, all requests shown to be in progress must be completed before the blocking performance for the new allocation (0 for user group 1, 95 for user group 2, and 0 for user group 3) materializes. During the transition phase, there is about 0.5% increase in blocking from the threshold of 3.5% before we start to see the reduction due to reconfiguration even as we increase the workload (see Fig. 3d).

4.2. Simulation results—protocols analysis

In this section, we present the performance analysis for the CCP and DCP protocols within a cluster. The metrics used are percent packet loss and packet latency in milliseconds (ms). A cluster of 4 identical video servers is used for this analysis and the average number of requests from 100 clients is varied in the range of 50–300 requests per hour. The clients generate requests based on Poisson distribution. Network related parameters are as shown in Table 4. The two protocols are compared against a base scenario where no attempts are made to distribute the traffic across servers. The server which receives a client request processes it in its entirety. The base scenario indicates a situation where there is no load balancing while both CCP and DCP offer better load balancing by design.

Fig. 5a shows the average packet loss as the number of streams is increased. The distributed protocol performs best, since it achieves the highest degree of load distribution across the network. Each client request is diffused across several different routes, and thus no particular route gets a higher amount of traffic than others. Packet losses due to local congestion conditions are minimized. The centralized control protocol performs slightly worse since the global state is updated only periodically and therefore, the load distribution is not ideal. Fig. 5b shows the average packet latencies for the same workload as before. The distributed control protocol shows slightly worse performance than the centralized protocol since each stream is served by all servers, both near and far from the client. Both protocols perform better than the base scenario. For more details on other experiments, we refer the readers to [31].

CCP and DCP differ in performance and neither protocol proves superior to the other in both metrics, packet loss and latency. It is easier to implement the CCP protocol in the distributed multiple cluster architecture as there is no guarantee that the optimal allocation will result in identical capacity servers in a cluster. The DCP protocol works best when the cluster of servers is identical in individual server capacity. The blocking performance will be unaffected by either CCP or DCP.

5. Conclusion

We presented an analytical model for matching user demands with server capacities dynamically. By incorporating RTT as a cost metric in our

analytical model, we provided a server selection scheme that takes proximity into consideration. Clustered servers are already used in CDNs by RealNetworks and other providers for Web-based services. Such current streaming solutions can leverage the proposed clustered architecture and experience significantly better performance. Realistic modeling of the Internet and traffic patterns in an extensive simulation strengthens this paper. The design and analysis of streaming protocols for within the cluster provides a comprehensive end-to-end architecture for streaming applications. Our simulation results for blocking analysis and cluster performance show that the proposed schemes perform better than other schemes where server allocations are not optimal.

Appendix 1. Background on transportation problem

The Transportation Problem (TP) is a special type of Linear Programming problem that is used with the distribution of resources from supply centers, called *Sources*, to receiving centers called *Destinations*. This formulation can be viewed as a simplified version of facility location problem where the facility locations with their capacities are already known and the task is to determine a minimum cost assignment of receiving centers to these facilities. In general, source i ($i = 1,2,\ldots,m$) has a supply of s_i units to distribute to the destinations, and destination j ($j = 1,2,\ldots,n$) has a demand for d_j units to be received from the sources. A basic assumption in the TP formulation is that the cost of distribution, c_{ij} is directly proportional to the number distributed between source i and destination j. We have adapted this formulation for our problem by representing the servers with a known stream handling capacity as the Sources and User Groups with their stream requests as the Destinations. The cost to be minimized is the RTT which is used as a distance measure between the server and the user groups.

The special structure of the general TP formulation lends itself to a solution procedure based on the Simplex method that automatically results in an integer-based solution to the allocation problem. As is explained in [32], the special structure of the TP results in great computational savings in the Simplex method.

Constructing an initial basic feasible solution can be done using one of three methods—Northwest corner rule or Vogel's approximation method or Russell's approximation method. The initial feasible solution is iteratively improved to get the optimal solution using either the Vogel's approximation or Russell's approximation method.

Northwest corner rule: Start in the northwest corner of the transportation simplex tableau by selecting x_{11}. If x_{ij} is the last basic variable selected, move one column to the right to choose $x_{i,j+1}$ if source i has any supply remaining. Otherwise, move one row down to select $x_{i+1,j}$. The initial feasible solution obtained by this method could be far from optimal solution and may result in more number of iterations than the other two methods.

A summary of the Transportation Simplex Method [32] is presented below:

Initial step: Obtain a basic feasible solution from one of three alternatives mentioned above.

Optimality test: Derive the u_i and v_j (which are the dual variables) by setting $u_i = 0$ for the row having the largest number of allocations and solving the set of equations $c_{ij} = u_i + v_j$ for each (i,j) such that x_{ij} is a basic variable. The current solution is optimal if the $(c_{ij} - u_i - v_j)$ is greater than or equal to 0 for every nonbasic x_{ij}. If any of the $(c_{ij} - u_i - v_j)$ is negative, go to the iterative step.

Iterative step:

1. Select the nonbasic variable x_{ij} having the largest negative value of $(c_{ij} - u_i - v_j)$ as the entering basic variable.
2. Determine the leaving basic variable as the one having the smallest value by identifying a chain reaction required to retain feasibility. This will give us a set of donor cells and receiving cells where the allocation is decreased and increased, respectively.
3. Determine the new feasible solution by adding the value of the leaving variable to the allocation of the receiving cell and subtracting this value from the allocation of donor cell.
4. Test the new feasible solution for optimality.

References

[1] Akamai Technologies, Akamai Streaming—When Performance Matters, White Paper, 2004.
[2] RealNetworks, RealNetworks Production Guide for Release 8, September 2002.
[3] Alexander Ferreira, Optimizing Microsoft Windows Media Services 9 Series, Technical Brief, Microsoft New Media Platforms Division, February 2003.
[4] QuickTime Streaming: End-to-end Solutions for Live Broadcasting and On-Demand Streaming of Digital Media,

Technology Brief, Mac OS X Server: QuickTime Streaming, 2004.
[5] Akamai Technologies, Internet Bottlenecks: the Case for Edge Delivery Services, White Paper, 2000.
[6] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, F. Zane, Clustering and server selection using passive monitoring, in: Proc. IEEE INFOCOM, 2002.
[7] A. Myers, P. Dinda, H. Zhang, Performance characteristics of mirror servers on the Internet, in: Proc. of IEEE INFOCOM, 1999.
[8] M. Crovella, R. Carter, Dynamic server selection using bandwidth probing in wide-area networks, in: Proc. of IEEE INFOCOM, 1997.
[9] M. Conti, E. Gregori, F. Panzieri, Load Distribution among Replicated Web Servers: A QoS-based Approach, WISP, ACM Press, 1999.
[10] R.B. Bunt, D.L. Eager, Achieving load balance and effective caching in clustered web servers, in: Proc. of the 4th International Web Caching Workshop, San Diego, CA, 1999.
[11] M. Colajanni, P.S. Yu, D.M. Dias, Analysis of task assignment policies in scalable distributed web-server systems, IEEE Transactions on Parallel and Distributed Systems 9 (6) (1998) 585–600.
[12] P. Rodriguez, E.W. Biersack, Dynamic parallel access to replicated content in the Internet, IEEE Transactions on Networking 10 (4) (2002).
[13] L. Amini, H. Schulzrinne, On probe strategies for dynamic multimedia server selection, in: Proc. of ICME, August 2002.
[14] P. Francis, S. Jamin, C. Jin, V. Paxon, D. Raz, Y. Shavitt, L. Zhang, IDMaps: a global Internet host distance estimation service, in: Proc. IEEE INFOCOM, March 2000.
[15] K.P. Gummadi, S. Saroiu, S.D. Gribble, King: estimating latency between arbitrary Internet hosts, in: Proc. of the 2nd ACM SIGCOMM Workshop on Internet Measurement, 2002.
[16] M. Guo, Q. Zhang, W. Zhu, Selecting path-diversified servers in content distribution networks, in: Proc. of IEEE GLOBECOM, 2003.
[17] T. Nguyen, A. Zakhor, Multiple sender distributed video streaming, IEEE Transactions on Multimedia 6 (2) (2004).
[18] T. Nguyen, A. Zakhor, Distributed video streaming over the Internet, in: Proc. of MMCN, 2002.
[19] V.N. Padmanabhan, H. Wang, P. Chou, K. Sripanidkulchai, Distributing streaming media content using cooperative networking, in: Proc. NOSSDAV, 2002.
[20] M. Doar, A better model for generating test networks, in: Proc. of GLOBECOM, 1996.
[21] M. Mellia, R. Lo Cigno, F. Neri, Measuring IP, TCP behavior on the edge node, Planet IP and Nebula Joint Workshop, 2002.
[22] H.J. Burch, F. Ercal, Mapping the Internet, Computer 32 (4) (1999).
[23] A. Broido, K. Claffy, Internet topology: connectivity of IP graphs, CAIDA (July) (2001).
[24] S. McCreary, K. Claffy, Trends in Wide Area IP Traffic Patterns. Available from: <http://www.caida.org/outreach/papers/>.
[25] E. Zegura, K.L. Calvert, M.J. Donahoo, A quantitative comparison of graph-based models for Internet topology, IEEE/ACM Transactions on Networking 5 (6) (1997).
[26] K.L. Calvert, M. Doar, E. Zegura, Modeling Internet topology, IEEE Communications Magazine (1997).
[27] The Mercator Internet Mapping Project at <http://www.isi.edu/scan/mercator/maps.html>.
[28] T.D. Neame, M. Zukerman, Modeling broadband traffic streams, in: Proc. of GLOBECOM, 1999.
[29] F. Fitzek, M. Reisslein, MPEG-4 and H.263 video traces for network performance evaluation, IEEE Network Magazine 15 (6) (2001).
[30] P. Mundur, R. Simon, A. Sood, End-to-end analysis of distributed video on demand systems, IEEE Transactions on Multimedia (February) 6 (1) (2004).
[31] A. Matthur, P. Mundur, Dynamic load balancing across mirrored multimedia servers, in: Proc. of ICME, 2003.
[32] F. Hillier, G. Lieberman, Introduction to Operations Research, McGraw-Hill Publishers, 1986.
[33] P. Mundur, A. Sood, R. Simon, Class-based access policies for distributed video on demand systems, IEEE Transactions on Circuits and Systems for Video Technology 15 (7) (2005).
[34] NS—Network Simulator at <http://www.isi.edu/nsnam/ns/>.
[35] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, W. Willinger, Network topology generators: degree-based vs. structural, in: Proc. of SIGCOMM, 2002.

Padma Mundur received the M.E. degree in systems engineering from the University of Virginia, Charlottesville, VA, in 1990 and the Ph.D. degree in information technology from George Mason University, Fairfax, VA, in 2000. Her undergraduate degree is in industrial and production engineering.

She is currently an Assistant Professor in the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County, Baltimore, MD. Her research interests are in distributed systems, multimedia networking, performance evaluation and analytical resource allocation techniques. She serves on the editorial board of IEEE Communications Surveys and Tutorials Journal, and has served on program committees of ICDCS 2004, 2006 and ICME 2004, 2005 among others. She has been a reviewer for the National Science Foundation panels, journals and conferences.

Poorva Arankalle received the B.E. degree in computer science from Cummins College of Engineering in Pune, India in 2002, and the M.S. degree in computer science from the University of Maryland, Baltimore County, MD in 2004. During her Masters, she interned at RealNetworks in the summer of 2004. She is currently employed at Google, after working at Yahoo! for two years as a server side software engineer.