Decentralized Resource Management for a Distributed Continuous Media Server

Cyrus Shahabi, *Member*, *IEEE*, and Farnoush Banaei-Kashani, *Student Member*, *IEEE*

Abstract—Distributed continuous media server (DCMS) architectures are proposed to minimize the communication-storage cost for those continuous media applications that serve a large number of geographically distributed clients. Typically, a DCMS is designed as a pure hierarchy (tree) of centralized continuous media servers. In an earlier work, we proposed a redundant hierarchical topology for DCMS networks, termed *RedHi*, which can potentially result in higher utilization and better reliability over pure hierarchy. In this paper, we focus on the design of a resource management system for RedHi that can exploit the resources of its DCMS network to achieve these performance objectives. Our proposed resource management system is based on a fully decentralized approach to achieve optimal scalability and robustness. In general, the major drawback of a fully decentralized design is the increase in latency time and communication overhead to locate the requested object. However, as compared to the typically long duration and high resource/bandwidth requirements of continuous media objects, the extra latency and overhead of a decentralized resource management approach become negligible. Moreover, our resource management system collapses three management tasks, 1) object location, 2) path selection, and 3) resource reservation, into one fully decentralized object delivery mechanism, reducing the latency even further. In sum, decentralization of the resource management satisfies our scalability and robustness objectives, whereas collapsing the management tasks helps alleviate the latency and overhead constraints. To achieve a high resource utilization, the object delivery scheme uses our proposed cost function, as well as various object location and resource reservation policies to select and allocate the best streaming path to serve each request. The object delivery scheme is designed as an application layer resource management middleware for the DCMS architecture to be independent of the underlying telecommunication infrastructure. Our experiments show that our resource management system is successful in realization of the higher resource utilization for the DCMS networks with the RedHi topology.

Index Terms—Distributed continuous media servers, decentralized resource management, distributed multimedia systems, content delivery networks, distributed information systems, video-on-demand.

1 Introduction

The advent of broadband access network technology that supports high-bandwidth connectivity for diversified end-user devices (e.g., PCs, wireless hand-held devices, set-top boxes, etc.) on the one hand and introduction of effective encoding techniques, such as MPEG-4, that allow streaming of continuous media (e.g., video and audio) with reasonably low bandwidth requirements on the other hand enabled the emergence of an enormous number of multimedia applications such as video-on-demand (VOD) [1], [2], [3], teleconferencing [4], distance learning [5], and interactive TV [6]. Researchers in academia and industry have been struggling for more than a decade to design high-performance centralized continuous media servers (CCMSs) [7], [8], [9], [10] and provide facilities to support specific QOS requirements of continuous media communications on the network [11], [12], [13], [14], all motivated by a desire to realize wide-spread use of these applications.

Distributed continuous media applications (e.g. VOD) are expected to provide service to a large number of clients often geographically *dispersed* on a metropolitan, country-wide, or even global area. With a naive design, employing only one large CCMS to support these distributed clients results in inefficient resource allocations that renders the design virtually impractical. For instance, the overall bandwidth requirement to implement an interactive VOD system with such a design is estimated to be as high as 1.54 Pb/s for the continental United States [15]. The full economic potential of such applications will not be realized unless cost-effective solutions are achieved.

To address this problem, one group of researchers has focused on various techniques to either reduce bandwidth requirements of individual continuous media streams, with methods such as *smoothing* [16], *staging* [17], and *bandwidth renegotiation* [18], or reduce overall bandwidth requirements of multiple streams via aggregation (or *statistical multiplexing*), for example, with *batching* and *multicasting* [19], [20]. As an orthogonal solution, other researchers have proposed distribution of the service to manage dispersedness of clients, i.e., employing a number of CCMSs each to serve clients located in a certain locality and interconnecting them via a high-speed network infrastructure to be able to share/exchange the continuous media objects (it is important to note that, although serving clients locally will result in a dramatic reduction in the total *communication bandwidth* requirement of the system, unless servers are able to share the objects, the huge aggregated *storage capacity* requirement of the servers will render the no-sharing approach impractical, too). Systems designed based on this approach [21], [22], termed distributed continuous media servers (DCMSs), are shown to be able to provide the optimal solution, i.e., the minimum *commu-*

• *The authors are with the Integrated Media Systems Center, Computer Science Department, University of Southern California, Los Angeles, CA 90089. E-mail: {shahabi, banaeika}@usc.edu.*

Manuscript received 4 Jan. 2000; revised 5 Oct. 2001; accepted 9 Jan. 2002. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111194.

nication-storage cost for distributed continuous media streaming applications [23], [24].

In this paper, we focus on developing a novel object delivery scheme as one of the components of a DCMS architecture. In the remainder of this section, first we describe the components of a *typical* DCMS architecture. Second, we briefly review those components of *our proposed* DCMS architecture, introduced in our previous work [25], in order to provide the context. Finally, we highlight the contributions of this paper by enumerating the characteristics of our object delivery scheme, the component of focus in this paper.

1.1 A Typical DCMS Architecture

A distributed VOD system[1] can usually be considered a rich repository of continuous media objects with a large set of dispersed clients. Objects should be served/streamed to the clients on demand. Generally, a DCMS that realizes such a VOD system is designed as a network with hierarchical (or tree) topology, with individual CCMSs as the nodes, and network links as the edges of the hierarchy. Nodes are assumed to be able to store a limited number of continuous media objects and stream a finite number of continuous media objects. Meanwhile, network links are expected to guarantee the specific QOS requirements of continuous media communications. Currently, there are two alternative underlying telecommunication infrastructures that can provide such guaranteed services: 1) circuit-switching networks, such as SONET, which provide *dedicated* physical and/or logical links, and 2) packet-switching networks that support specific guaranteed services (such as the "Integrated Services" [26] or the "Differentiated Services" [27] for IP-based networks, e.g., the Internet, or CBR and VBR services for ATM networks [28]) to emulate characteristics of the dedicated links by enforcing statistical and/or deterministic guarantees.

The nodes at the leaves of the hierarchy, termed *head-ends*, are points of access to the system. In practice, clients are connected to the head-ends (usually located in local central offices or COs) via broadband local-access networks, e.g., xDSL or cable network. When a request for an object arrives at a head-end, if the object is available in its local storage, the head-end serves the client per se, else the request will be forwarded to the higher levels of the hierarchy and, eventually, some other node that has the object stored locally will serve the client by streaming the object through the hierarchy and, finally, via the head-end to the client. As this brief description of the system structure and functionality conveys, a DCMS network should also consist of a middleware component for resource management. The middleware is supposed to address two different orthogonal issues:

1. *Object Placement*. Static and/or dynamic mapping of objects onto the DCMS network nodes (the storage space) so that the overall communication-storage cost of the system is optimal. Many researchers have addressed this problem, also known as the Media Asset Mapping Problem (MAMP), by introducing analytical models that consider user access patterns [29] in addition to communication and storage constraints to obtain optimized object distribution and replication (or caching) policies [30], [31], [32], [33].
2. *Object Delivery*. On client demand and in real time, locating the replicas of the objects within the DCMS network and selecting the appropriate replica and allocating system *streaming* resources (i.e., node and link bandwidth) for object delivery so that high resource utilization is achieved. The focus of this paper is on object delivery; we discuss this problem in more detail in the subsequent sections.

1.2 DCMS Architecture—Previous Work

As far as a CCMS can support the required storage and streaming capabilities, choosing CCMS for the network nodes does not affect the DCMS design. Similarly, if the middleware is implemented at the application layer, as far as the links (either physical or logical, dedicated genuinely or by emulation) provide the required guaranteed service, the DCMS design will be independent of the underlying telecommunication infrastructure. Therefore, the *network topology* and the *resource management middleware* are the two main components that characterize a DCMS architecture. In [25], we studied the former by extending the pure hierarchy to a new topology termed *Redundant Hierarchy* or, briefly, *RedHi* (see Fig. 1b). RedHi relaxes the hard degree-1 parent connectivity restriction with pure hierarchy to be degree-2 or more.[2] Consequently, there is a higher potential for load-balancing among nodes and links of the DCMS network (both in the absence and presence of node and/or link failures), hence, higher streaming resource utilization and cost-efficiency. The redundancy in RedHi is not of bandwidth, but of number of links. In other words, the aggregated bandwidth of the links connecting a node to its parents can be the same as the bandwidth of the link connecting the node to its single parent in a pure hierarchy. Therefore, RedHi does not impose higher bandwidth requirements, but only requires higher connectivity. In fact, even if we are restricted to using dedicated physical links for DCMS network, RedHi structure is quite compatible with redundant hierarchical organization of the current telecommunication networks, such as the Internet, because COs (Central Offices), POPs (Points of Presence), and ISPs (Internet Service Providers), which are best potential locations for DCMS nodes, are usually redundantly connected to several larger parent nodes in the hierarchy [34], [35]. In [25], we also explored some object placement/replication schemes to address the MAMP problem.

In order to complete our proposed DCMS architecture and, particularly, its resource management middleware, in this paper, we introduce an object delivery scheme that exploits the characteristics of the RedHi topology to realize

1. In this paper, we consider VOD, as one of the most promising continuous media streaming applications, for discussion purposes. However, the discussion can be extended to cover other continuous media streaming applications as well.

2. We include the formal definition of the RedHi topology in the Appendix for ease of reference.

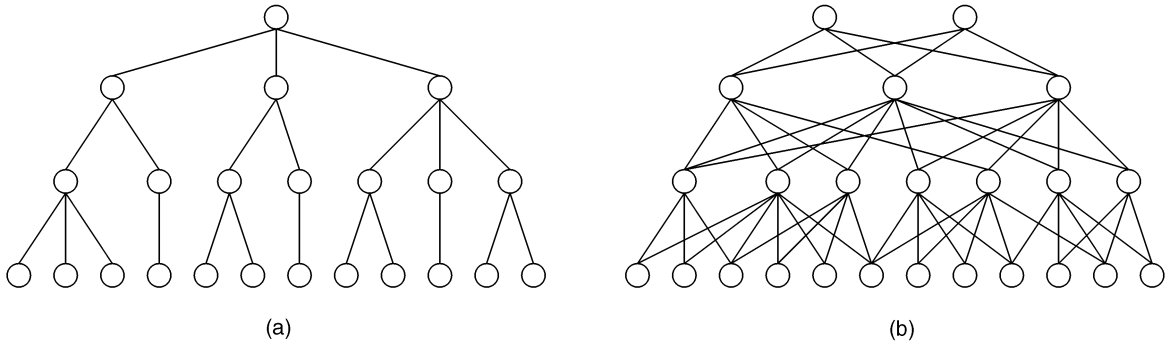

Fig. 1. (a) Pure hierarchy vs. (b) redundant hierarchy.

its potential advantage, i.e., higher utilization of streaming resources.

1.3 Contributions

Consider a set of shared continuous media objects distributed among nodes of a DCMS network based on an object placement scheme.[3] Several replicas of an object may exist in the network at a time, and replicas may *dynamically* be inserted and/or deleted according to some object replication (or caching) policy. As a client request arrives to *read* an object, an "object delivery scheme" is required to locate replica(s) of the object and select and allocate the *best* path within the DCMS network (sourcing in one of the object replicas) to deliver the object to the client. This resource management scheme should select the best path so that the overall utilization of the streaming resources (i.e., node and link bandwidth) of the DCMS network is high.

Many distributed object processing applications are not latency and/or overhead-tolerant. In such systems, full decentralization of the resource management is inefficacious and impractical. However, with continuous media applications, the additional start-up latency and resource management overhead (attributed to decentralization) is negligible as compared to the duration and resource requirements of continuous media objects, e.g., less than 2 seconds of start-up latency for a 1-hour movie or 10b/s of communication overhead for a 1Mb/s bandwidth movie (or, equivalently, less than 5KB of total communication overhead for a 0.5GB storage-size movie). On the other hand, one can achieve optimal robustness and scalability with decentralization of the management. In this paper, assuming the RedHi topology and a fixed object replication scheme, we introduce an object delivery scheme that *collapses* three mechanisms, namely, *object location*, *path selection*, and *resource reservation*, into one fully *decentralized* delivery mechanism. Decentralization of the resource management meets our scalability and robustness objectives optimally, whereas collapse of the mechanisms helps to satisfy the start-up latency and overhead constraints. To achieve high resource utilization, the object delivery scheme selects and allocates the best streaming path to serve each request based on our proposed cost function, which considers 1) the overall amount of the streaming resources engaged by selecting each streaming path and 2) the relative loads of the paths. The object delivery scheme also comprises various optional object location and resource reservation policies. Finally, this scheme is designed as an application layer resource management middleware for the DCMS architecture to be independent of the underlying telecommunication infrastructure.

We have extended the NS (Network Simulator) networking event simulator [36] to incorporate our object delivery scheme. We conducted several experiments via simulation to evaluate performances of the proposed optional policies and to verify that our object delivery scheme meets our objectives. Results of our experiments show that our delivery scheme with a RedHi DCMS network can achieve up to 50 percent improvement in resource utilization over a pure hierarchy.

The remainder of this paper is organized as follows: In Section 2, we discuss the design objectives of the object delivery scheme. Section 3 explains our design approach to meet the required objectives. In Section 4, we define the functionality of our object delivery scheme. Our simulation model and the results of our experiments are discussed in Section 5. We delay the survey of the related work until Section 6. Finally, Section 7 concludes the paper and discusses our future directions.

2 Design Objectives

Optimization of resource utilization is the main objective of an object delivery scheme. In addition to resource utilization optimization, the mechanisms used for object delivery should meet some other objectives, the most important of which are scalability, robustness, minimum start-up latency, and minimum resource management overhead. We believe that, unlike many other parallel/distributed object processing applications [37], such as distributed object programming models (e.g., CORBA), PRAMs (Parallel Random Access Machines) [38], and mobile object tracking systems [39], which must strive to minimize the latency and/or resource management overhead of access to the objects as their highest priority objectives, with object streaming applications, due to often large duration and high resource requirements of continuous media objects, these two objectives are generally less important. For instance, with a VOD system, the user reneging behavior due to intolerable start-up latency is often modeled by distributions with expected values estimated as high as

3. Here, we assume objects are not decomposable; hence, object is the grain size for distribution.

several minutes [40], [41], [42], e.g., in [42], the user reneging behavior is modeled by a normal distribution with expected value $\mu = 5min$ and standard deviation $\sigma = \mu/3$. Thus, any start-up latency in the order of seconds will result in a negligible user reneging rate. Besides, as compared to the high demands of continuous media objects for system resources (i.e., memory, CPU-time, and bandwidth), the amount of resources used for resource management purposes is relatively small. Therefore, since our object delivery scheme is designed for a DCMS network, we consider these two objectives, namely minimum start-up latency and minimum resource management overhead, as the least important ones. Below, we order the required objectives based on their importance and provide a more detailed definition for each objective:

1. *Resource Utilization Optimization*. The object delivery scheme should realize the higher streaming resource utilization advantage of the RedHi topology, which is considerable both in the absence and presence of node and/or link failures. We evaluate the utilization by the conventional *client blocking ratio* (or simply *blocking*) measure; the less the blocking, the higher the resource utilization. Of course, with low blocking and high resource utilization, the media server system is more cost-effective/profitable.
2. *Scalability*. The object delivery scheme should remain practical for large-scale applications; particularly, resource management performance bottlenecks should be avoided to allow extending the DCMS network by adding nodes and links as required.
3. *Robustness*. The object delivery scheme should be tolerant toward the failures of the resource management system components. Particularly, if the resource management scheme is designed to run on top of the same DCMS network used to deliver the actual continuous media objects, failures of the network components, i.e., nodes and links, should have the least possible impact on the performance of the resource management system.
4. *Start-Up Latency Alleviation*. The object delivery scheme should decrease the object streaming start-up latency to the point that the client reneging rate due to latency is negligible.
5. *Resource Management Overhead Alleviation*. Memory, CPU-time, and bandwidth requirements of the resource management imposed as overhead to the DCMS system should be negligible as compared to requirements of the actual content of the system, i.e., continuous media objects.

As we discuss in Section 3, this ordering of the objectives has an important impact on the main decision with the design of the object delivery scheme, i.e., positioning strategy (centralized or decentralized) for the control data that are used for resource management purposes.

3 Design Approach

For resource management tasks, such as object location, path cost evaluation, and resource allocation/reservation, the object delivery scheme should maintain a set of static and/or dynamic state data, termed *metadata*, about system resources (i.e., objects, nodes, and links). The metadata can be either maintained at a central location in the DCMS network or distributed among the network nodes. At one extreme case, the metadata are fully distributed (or *decentralized*) so that each node only stores the metadata relevant to the local resources (i.e., the node itself, the adjacent links, and the objects stored at that node). At the other extreme, the central management point becomes both a performance bottleneck and a single point of failure. However, full decentralization of the metadata results in optimal scalability and robustness for the object delivery scheme. This is because the management load is shared to the extreme and the impact of each failure is reduced to the minimum (only metadata relevant to the locality of a certain failed node or link may be unavailable). Meanwhile, with decentralization, management of the *metadata*, on its own, may add to the response-time and resource requirements of the resource management system; hence, metadata decentralization possibly results in higher start-up latency and resource management overhead (particularly, CPU-time and bandwidth overhead, but not memory overhead).

However, the increase in start-up latency and resource management overhead attributed to the decentralization of metadata is negligible as compared to the duration and resource requirements of continuous media objects (see Section 2). Therefore, we design our delivery scheme assuming fully decentralized metadata maintenance to achieve optimal scalability and robustness. In this case, each node executes a set of three *distributed* algorithms to perform object location, path selection, and resource reservation tasks. To meet the other two (less important) objectives, namely start-up latency and resource management overhead, many distributed resource management systems allow *replication* of the metadata besides decentralization, i.e., there might be several replicas of the same metadata located in various positions in the system. Replication potentially facilitates access to the metadata; hence, it decreases overall response-time and resource requirements of the resource management system. However, since metadata may be dynamically updated, replication introduces the consistency issue, which either decreases the accuracy of the resource management system or, if addressed, results in additional overhead. With our design, we use task *collapsion* instead of metadata replication. Our object delivery scheme collapses the three tasks required for object delivery into a single delivery mechanism that performs the tasks integrated. As compared to serial execution of the three distributed algorithms, integrated execution helps minimize the start-up latency. Moreover, the collapsion mechanism uses multipurpose communication (e.g., single packet sent for object location and path cost evaluation at the same time) to reduce the resource management overhead (particularly CPU-time and bandwidth overhead due to decentralization). In brief, decentralization of the mechanism meets the higher priority scalability and robustness objectives optimally, whereas collapsion of the tasks helps satisfy the start-up latency and resource management overhead constraints. In Section 4, we

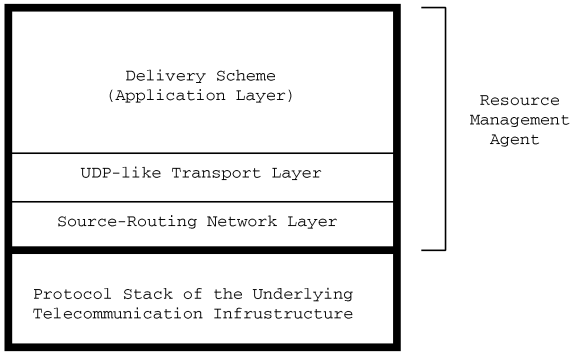

Fig. 2. Protocol stack at a DCMS node.

focus on describing the *functionality* of the object delivery mechanism developed based on this approach and explain how it achieves its main objective, i.e., high resource utilization.

4 Mechanism

Our object delivery scheme runs as an application layer middleware on a DCMS network. The comprehensive set of assumptions that we make with respect to the DCMS network is as follows:

- The DCMS network has a RedHi topology.
- Nodes of the DCMS network are CCMSs (possibly heterogeneous), each with limited streaming bandwidth and storage space (to cache some continuous media objects locally) known to the middleware. Nodes execute both the resource management middleware and the stream server.
- Links of the DCMS network (logical, physical, or both) are provided by some underlying telecommunication infrastructure that guarantees availability of a fixed limited bandwidth known to the middleware. Resource management control traffic and continuous media object streams *share* the links. Streaming resources are reserved for media streams, but control traffic is served based on the best-effort service model.[4] The middleware considers the links as point-to-point connections for duplex communication between adjacent pairs of nodes.
- We assume a simple object distribution and caching policy: The entire set of objects are located at the root(s) of the network when DCMS is initiated; as an object is served through a path from an object server node to a head-end, the object is cached at all intermediate nodes on the path based on the LRU (Least Recently Used) caching policy [43].

Our middleware consists of a set of peer resource management agents mapped one-to-one onto the DCMS network nodes. Each agent is comprised of a protocol stack on top of the network links (see Fig. 2), which makes the middleware independent of the nature of the links: the network layer supports source routing, the transport layer executes a UDP-like protocol with minimum required functionality, and the application layer implements the actual delivery mechanism. Here, we focus on the functionality of the application layer, which is the main contribution of this paper. Implementation of other layers is trivial and based on the standard approaches.[5]

The agent associated with each node exclusively maintains the local metadata relevant to the node resources, i.e., objects stored in local storage, available bandwidth of the node, and available bandwidth of the links that connect the node to its child nodes. As the agent running on a head-end receives a request from a client for streaming an object, the agents running on the network nodes collectively execute a *distributed* mechanism to 1) locate all nodes of the DCMS network that have the object stored locally, 2) select one of those nodes as the object server and select the "best" path through the DCMS network for streaming the object from the object server to the head-end so that overall utilization of the streaming resources of the network (i.e., nodes and links bandwidth) is optimized by load balancing, and 3) reserve required streaming resources along the selected path so that the server can start streaming the object to the client. Below, first we provide an overview on the life cycle of a request as served by this object delivery mechanism. Subsequently, we focus on the particular characteristics of each of the three tasks collapsed into the mechanism.

4.1 Request Life Cycle

As a head-end receives a *request* from a client to read an object (e.g., as an RTSP request [44]), its agent generates a *query* packet and propagates/floods it to the network. Propagation of the query packet is performed by *selective* flooding based on a propagation policy that determines the coverage of the propagation (see Section 4.2 for various propagation policies). Instances of the flooded query packet traverse different paths and incrementally evaluate the cost of the paths as they are forwarded from agent to agent (see Section 4.3 for the definition of the path cost). Some query packets reach the nodes that store a replica of the requested object locally. Those nodes send *response* packets back along the same path traversed by the corresponding query packets to report availability of the object replica as well as the cost of the path to reach the replica. The agents along the path compare the responses to select and reserve the best path and to filter out the rest of the responses so that, finally, the head-end receives the response that indicates the best path. The path is already reserved and the selected object server starts streaming the requested object as soon as it receives a *start* packet from the head-end.

Agents use four types of packets to communicate and to execute the object delivery algorithm: query packet, response packet, release packet, and start packet. Data fields included in these packets are illustrated in Table 1. As indicated in the table, the combination of the two fields

4. Since our object delivery mechanism uses *soft-state* (for the control state of the mechanism and not the metadata state of the resources) it does not necessarily assume guaranteed service without lost/delayed packets on the communication links. Therefore, optionally, the resource management mechanism can use best-effort service for communication between nodes to eliminate resource management communication overhead on the DCMS network.

5. Our source code that implements these protocols is available at ftp://zahedaan.usc.edu/GMeN/simulation.

TABLE 1
Data Fields of the Packets

Data Field	Definition	Packet Type
type	packet type	all
headend_ID	ID of the head-end which receives the request	all
sequence_no	unique sequence number assigned to the request by the head-end	all
path	path	all
object_ID	ID of the requested object	all
response	result of the object location query (REJECT, FOUND, or NOT-FOUND)	response
local_seqno	local sequence number assigned to the query packets replicated at a node	query, response
path_length	path length (number of nodes traversed in the path)	query, response
path_freeBW	amount of free streaming resources of the path	query, response

```
while (lifetime > 0) {
    ReceiveQuery(new_query);

    if (request is already RESPONDED)
        SendResponse(new_query, REJECT);
    else if (path_length > 2× network diameter)
        SendResponse(new_query, REJECT);
    else if (the requested object is available in local storage)
        if (do not have enough local streaming resources to stream the object)
            SendResponse(new_query, REJECT);
        else {
            SendResponse(new_query, FOUND);
            if ((reserve_count == 0) {
                ReserveLocalResources(object_ID);
                reserve_status = RESERVED;
            }
            reserve_count++;
        }
    else // the requested object is not available in local storage
        if ((cannot forward the query to any other agent) OR
           (have already received a query packet from a path that costs less))
            SendResponse(new_query, REJECT);
        else {
            if (not the first privileged query)
                SendResponse(old_query, REJECT);
            if ((reserve_count == 0) AND
               (RESERVATION_MODE == ERP)) {
                ReserveLocalResources(object_ID);
                reserve_status = RESERVED;
                reserve_count++;
            }
            UpdateQueryState(new_query);
            ForwardQuery(new_query);
            ResetResponseTimeout();
        }
}

PurgeQueryState();
```

Fig. 3. Processing of a query packet at an agent.

sequence_no and headend_ID, existing at all types of the packets, uniquely identifies the packets generated in response to the same object request.[6] Below, we summarize functionality of the delivery algorithm by discussing how packets are processed at an agent.

6. We use a *lollipop* sequence number space [45]. The space is large enough so that the sequence number does not wrap around during the lifetime of a request.

Fig. 3 shows how an agent processes a *query packet* received from a neighbor. The agent creates and maintains a *query state* for each request as it receives the first query packet (see Table 2 for the contents of the query state). The agent may receive several query packets about the same request from different paths at different times. It only updates the query state if the received query has traversed the path with the minimum cost up to the current time; such

TABLE 2
Data Fields of the Query State

Data Field	Definition
headend_ID	ID of the head-end which initiated the query
sequence_no	unique sequence number assigned to the request by the head-end
querier_ID	ID of the neighbor agent that has forwarded the privileged query
querier_localseqno	local sequence number assigned to the query packet by the querier (the neighbor)
path_length	path length (number of nodes traversed by the query packet)
path_freeBW	amount of free streaming resources of the path traversed by the query packet)
reserve_count	number of paths for which the node resources are reserved (reserved once only)
reserve_status	if local resources are reserved for this request (RESERVED or NOT-RESERVED)
status	state of the request (RESPONDED or NOT-RESPONDED)
lifetime	time-out to purge the query state

a query is termed the current *privileged* query. Other queries are immediately rejected. A query is interpreted as: "What is the best path to reach a replica of the requested object through you (the agent)?" As a new privileged query is received, if a replica of the requested object is available locally and local resources are also enough to stream the object, the agent sends a response packet in reply to the query to report 1) the path between the head-end and the local node, and 2) the total cost of the path. Otherwise, it replicates the privileged query, updates the path parameters in the replicated packet (i.e., `path`, `path_length`, and `path_freeBW`), and selectively floods it to other neighbors, querying them for the same object. The forwarded query packets are marked with a local sequence number, `local_seqno`, which increases each time the query state is updated with a new privileged packet. Therefore, in case multiple queries have already been forwarded, the response packets received are identifiable. Eventually, when a privileged query is replied to by sending a response packet to the querier, `status` of the request is set to RESPONDED (see Fig. 4). Those query packets received while the query state is in RESPONDED mode are rejected. Also note that, to damp possible lost packets, if the length of the path traversed by the received query packet is more than two times the network diameter, the query is rejected. The query state will be maintained for a duration indicated with `lifetime`, which is long enough for the delivery algorithm to end (with our experiments, we set `lifetime` to the streaming duration of the requested object).

Fig. 4 illustrates how an agent processes the *response packets* received from neighbors. As an agent receives the first query packet and forwards it to its neighbors, it generates a *response state* to keep track of the responses from neighbors (see Table 3 for the contents of the response state). The agent waits to receive the response packets in reply to its query, at most for an amount of time equal to `time_out`. Hence, in a way, the response state is a soft state and failure to receive responses from some neighbors does not confuse the algorithm. The `time_out` value is set to two times the maximum RTT between the agent and a root node. If the received response packets report finding replicas of the requested object, the agent generates a response packet containing the path to reach a replica with the minimum cost, sets the `local_seqno` filed of the packet to `querier_localseqno`, and sends it back to the querier of the current privileged query. If a new privileged query is received and forwarded before the response is sent to the last privileged query, the response state will be reset and the agent will wait to receive responses for the new current privileged query. In such a case, responses to the old forwarded queries are ignored because those queries are rejected as the new privileged query arrives (see Fig. 3). As mentioned before, when the current privileged query is responded, the query state of the request is set to RESPONDED mode and the response state of the request is purged. Agents ignore the responses relevant to a request for which they do not have a response state.

With recursive execution of the processes that we have just described, agents can cumulatively locate the replicas of the requested object and select the best path to reach a replica. Using *release packets*, reservation of the best path is also performed integrated with the other two tasks. Since there are two options for the implementation of reservation, we delay discussion of this part of the mechanism until Section 4.4. As the head-end agent receives responses from its neighbors, similarly to regular agents it processes the response packets to select the best path (see Fig. 4). Successively, it sends a source-routed *start packet* to the selected object server and requests for streaming the object through the selected path. Finally, when streaming of the object is terminated, head-end sends a release packet to the object sever through the same path. All agents along the path release the streaming resources reserved for the object as they receive and forward the release packet.

4.2 Object Location

As a routing mechanism, *flooding* is well known for 1) tremendous robustness, by taking every possible path in parallel, 2) ability for concurrent communication with all nodes of a network, which is particularly useful in distributed applications, and 3) minimum delay, by taking the path with minimum delay among other paths. All these characteristics are highly desirable for object location. Of course, the overhead imposed by flooding is a prohibitive disadvantage that restricts its use with most distributed object processing applications. However, as we explained is Section 1.3, continuous media applications are overhead-tolerant.

Since, with our application, object location is integrated with other tasks, we need to modify the typical flooding

```
while (time_out > 0) AND
      (response to the current privileged query is not received from all addressees)) {
    ReceiveResponse(new_respnse);

    if ((local sequence number of response packet is equal to local_seqno in response state) AND
        // ignore the responses to the old privileged queries; they have already been rejected
       (response != REJECT) AND
        // ignore REJECT responses
       (response != NOT - FOUND) AND
        // ignore NOT - FOUND responses
       (cost of the path advertised in the response packet is less than minimum cost)
        // ignore non-optimal paths
      )
        UpdateResponseState(new_response);

    if ((local sequence number of response packet is not equal to local_seqno in response state) OR
       (cost of the path advertised in the response packet is more than minimum cost)
        SendRelease(new_response);
}

// sending the aggregated response to the current privileged query packet
if (have not received any FOUND response for the current privileged query) {
    SendResponse(NOT - FOUND);
    if (RESERVATION_MODE == ERP) {
        ReleaseLocalResources(object_ID);
        reserve_status = NOT - RESERVED;
        reserver_count = 0;
    }
} else {
    if (RESERVATION_MODE == ERO)
        if (do not have enough local streaming resources to stream the object) {
            SendRelease(new_response);
            SendResponse(REJECT);
        }
        else {
            ReserveLocalResources(object_ID);
            reserve_status = RESERVED;
            reserve_count++;
        }

    else // RESERVATION_MODE == ERP
        SendResponse(FOUND);
}

status = RESPONDED;

PurgeResponseState();
```

Fig. 4. Processing of a response packet at an agent.

algorithm to adapt it for our delivery mechanism. Typically, flooding is used to inform/update nodes of a network with some piece of information. Therefore, as soon as one replica of the flooded packet is received by a node, the node can ignore other replicas. Since, with our delivery mechanism, replicas of the flooded query packet evaluate the cost of the paths that they traverse in addition to querying the nodes for the requested object, the nodes must differentiate the replicas based on their paths. As we described in Section 4.1 (see Fig. 3), when a node receives new privileged queries, it resets its query state and repeats the query process by forwarding the new query to its neighbors, even though it might have already received and forwarded query packets for the same request but from more costly paths. Recursively, this approach allows other nodes to reevaluate their queries based on the cost of the path introduced by the new query packet so that, progressively, the cost of the selected path is optimized.

Moreover, we introduce several propagation policies for *selective* flooding to be able to control the coverage of the object location and study its effect on the resulting utilization. The wider the coverage, the nearer the resource utilization to the optimal case because more paths and object servers are investigated. On the other hand, widespread object location results in higher start-up latency and resource management overhead. The three choices of propagation policy are as follow:

TABLE 3
Data Fields of the Response State

Data Field	Definition
headend_ID	ID of the head-end which initiated the query
sequence_no	unique sequence number assigned to the request by the head-end
addressee_IDs	IDs of the neighbor nodes that receive the forwarded query
local_seqno	local sequence number assigned to the current privileged query packet
path	path to reach the replica advertised in the response packet
path_length	length of the path to reach the replica advertised in the response packet
path_freeBW	amount of free streaming resources of the path to reach the replica advertised in response
time_out	time-out for receiving response packets (soft-state to consider possible failures)

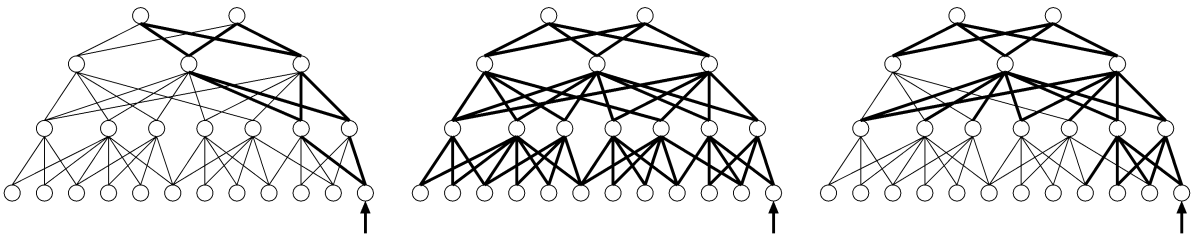

Fig. 5. Coverage of the propagation policies. (a) Parent-only. (b) Inclusive. (c) Sibling.

1. *Parent-Only*. With this propagation policy, nodes only forward the queries to their parents. Therefore, the coverage of the object location is limited to the the ancestors of the head-end that receives the request for the object (see Fig. 5a).
2. *Inclusive*. This propagation policy is equivalent to nonselective flooding, i.e., nodes forward the query packets to all their neighbors except the one from which the query is received. Therefore, all replicas of the requested object are located by covering the entire network (see Fig. 5b).
3. *Sibling*. This propagation policy is an extension of the parent-only policy. With this policy, a node that receives a query from one of its children forwards it to 1) the siblings of the child from which the query is received and 2) its own parents. However, if a node receives the query from one of its parents, it does not forward the query at all (see Fig. 5c). Sibling policy is simply one logical intermediate case, with wider coverage for object location as compared to the parent-only policy, but not as wide as the full coverage of the inclusive policy.

Our experiments show that the sibling policy (as compared to the inclusive policy) results in near-optimal utilization while it imposes much less overhead and latency with its limited coverage (see Section 5.2.2 and Section 5.2.3).

4.3 Path Selection

As discussed in Section 4.1, the query packets incrementally evaluate the cost of the path that they traverse as they are forwarded from agent to agent. Evaluation of the path cost during propagation of the query packets allows the nodes to select the best path from the head-end to the node and damp the query packets that have not traversed the best path (see Fig. 3 for query processing). On the other hand, the response packets sent in reply to the queries also contain path cost values that enable the node to select the best path from the node to a replica (see Fig. 4 for response processing). Therefore, in this way, cumulatively, nodes converge on selecting the best path from the head-end to a replica.

Our cost function evaluates a path based on two parameters: 1) length of the path (path_length) and 2) amount of the free streaming resources available along the path (path_freeBW). To contrast the costs of two paths, first they are compared on the total amount of streaming resources required to stream an object through the paths. This value is proportional to the length of the path[7]; hence, we use path_length as the measure for this comparison. Second, if the paths have the same length, the cost function breaks the tie condition based on the amount of free streaming resources of the paths; the more free resources are available along a path, the less cost is associated to the path so that we can avoid blocking fairly loaded paths and balance the load among the parallel paths. We propose two measures to assess the amount of free streaming resources in a path: 1) *bottleneck*, i.e., minimum amount of free bandwidth available among the components (nodes and links) of a path, and 2) *average*, i.e., average amount of free bandwidth available at those components. The two parameters representing the cost of a path, P, are formally defined as follows:

Definition 1 Cost Function Parameters. *Assuming* $P = N \cup L$, *where*[8]

$$N = \{n \mid n \text{ is a node of the DCMS network}, n \in P\}$$
$$L = \{l \mid l \text{ is a link of the DCMS network}, l \in P\}$$

7. Particularly, it is equal to $B \times (2n-1)$, where B is the bandwidth of the object and $n =$ path_length.

8. Similar to the bandwidth of a link, the bandwidth of a node is defined as the total number of bits of information that a node can serve per time unit.

then

$$\texttt{path\_length} = \|N\|$$

$$\texttt{path\_freeBW} = \begin{cases} \texttt{path\_freeBW}_{\text{bottleneck}} \\ or \\ \texttt{path\_freeBW}_{\text{average,}} \end{cases}$$

where

$$\texttt{path\_freeBW}_{\text{bottleneck}} = min(\{FreeBandwidth(i) \mid i \in P\})$$

$$\texttt{path\_freeBW}_{\text{average}} = \frac{\sum_{i \in P} FreeBandwidth(i)}{\|P\|}.$$

With this cost function, we never select longer paths over available shorter paths. Since, in DCMS networks with leveled topologies (such as RedHi or pure hierarchy), head-ends are the leaves at the lowest level, any path sourcing in higher levels should go through the lower levels to reach a head-end; therefore, sharing loads of the short paths with long paths does not necessarily result in better overall utilization. We try load balancing by considering the path traffic only when the possible path options have the same length. Our cost function can be used to evaluate the cost of the paths incrementally. As the query packet is forwarded from agent to agent, each agent updates the cost parameters of the path by considering the effect of appending the local link and node to the path. Finally, it is important to note that our resource management system, as an application layer middleware, applies the cost function to optimize utilization of the streaming resources. Therefore, the cost function does not need to consider the network level costs, such as link delay, or the actual financial cost of the resources along the path.

Since 1) with flooding we evaluate *all* possible paths restricted to the DCMS nodes explored during object location (i.e., all paths within the subnet of the network that is covered by the particular propagation policy applied) and 2) with our cost function we are able to minimize the resources allocated to respond to each request (by selecting the shorter paths as well as balancing the load among the paths with the same length), we expect our path selection approach can realize the higher utilization of the RedHi topology. Our experiments verify our expectations. We have also performed experiments to compare performance of $\texttt{path\_freeBW}_{bottleneck}$ with $\texttt{path\_freeBW}_{average}$ (see Section 5.2.1).

4.4 Resource Reservation

Reservation of the streaming resources by the application layer middleware is a bookkeeping task required to ensure load balancing as well as uninterrupted streaming. Reservation and release of the resources are performed by updating the metadata maintained locally. Generally, there are two alternative approaches for resource reservation: 1) *pessimistic*: to prereserve all resources that might possibly be required for the selected path during the path selection process and to release the extra resources only after the path is actually selected and 2) *optimistic*: to start reserving the required resources after path selection is finalized, hoping the resources are still available. With the pessimistic approach, prereservation guarantees availability of the required resources when the path is selected. However, with prereservation, since the selected path is not known yet, resources are over-reserved. Therefore, concurrent requests underestimate the available streaming resources, which might result in selecting inappropriate paths. On the other hand, with the optimistic approach resources are not over-reserved, but the resources required for the selected path might have been preallocated to other concurrent requests.

The extreme pessimistic and optimistic approaches are not appropriate for our mechanism. The extreme pessimistic approach results in high blocking ratio. Meanwhile, with the extreme optimistic approach, reservation cannot collapse into the mechanism, but it should be performed in serial with other delivery tasks. We propose two moderated policies for resource reservation: Early Release Pessimistic (ERP) and Early Reserve Optimistic (ERO). With ERP, resources are still prereserved during query propagation, but, unlike the extreme pessimistic approach, the extra resources are released early *during* the path selection process (and not after path is selected), as soon as they are disqualified from being in the best path. With ERO, resources are reserved early during response processing, before path selection is finalized. Both of these policies suggest prereservation, but ERP is still more pessimistic as compared to ERO because, with ERP, reservation is performed during query propagation while ERO reserves resources during response processing.

Agents decide on prereserving the local resources as they receive query packets (with ERP policy) or response packets (with ERO policy). Release packets are used to release the extra resources afterward. Fig. 6 shows how an agent processes a *release packet*. A release packet is source-routed from a node toward an agent that has found a replica of the requested object stored locally and has sent a FOUND response in reply to an earlier query. The release packet releases all resources reserved for the corresponding request along the path. The agents that are included in several reserved paths reserve the resources only once and release the resources only if they receive release packets for all the paths. The reservation process is performed based on either ERP or ERO policy as follows (see Figs. 3 and 4):

1. *Early Release Pessimistic (ERP)*. With ERP, an agent reserves the required local sources as it receives the first query packet about the request. As response packets in reply to the forwarded queries are received, the node sends release packets back along all paths from which FOUND responses are received except the best path selected at the node. If the agent does not receive any FOUND response, it releases the local resources reserved for the request and rejects the query.
2. *Early Reserve Optimistic (ERO)*. With ERO, an agent does not reserve local resources until it receives FOUND responses in reply to its queries. Thus, the resources are only reserved when the agent actually finds a path to reach a replica of the requested object, although the path might not be selected as the best path at last. Releasing the extra resources is

```
ReceiveRelease(new_release);

if (reserve_status == RESERVED) {
    reserve_count−−;

    if (reserve_count == 0) {
        ReleaseLocalResources(object_ID);
        reserve_status = NOT − RESERVED;
    }
}

if (not the last hop in the path included in the release packet)
    ForwardRelease(new_release);
```

Fig. 6. Processing of a release packet at an agent.

TABLE 4
Pure Hierarchy Configuration

Level(l)	CCMSs			Links to higher level	
	Nodes ($\mathcal{V}_{level}$)	Bandwidth(Mb/s)	Storage(GB)	Edges	Bandwidth(Mb/s)
1	{0}	400	250	$\emptyset$	-
2	{1,2,3}	300	160	$\{(i,j) \mid i \in \mathcal{V}_2, j \in \mathcal{V}_1\}$	300
3	{4,..,12}	200	40	$\{(i,j) \mid i \in \mathcal{V}_3, j \in \mathcal{V}_2, j = (i-1)/3\}$	200
4	{13,..,39}	100	10	$\{(i,j) \mid i \in \mathcal{V}_4, j \in \mathcal{V}_3, j = (i-1)/3\}$	100

TABLE 5
RedHi Configuration

Level(l)	CCMSs			Links to higher level	
	Nodes ($\mathcal{V}_{level}$)	Bandwidth(Mb/s)	Storage(GB)	Edges	Bandwidth(Mb/s)
1	{0,40}	200	250	$\emptyset$	-
2	{1,2,3}	300	160	$\{(i,j) \mid i \in \mathcal{V}_2, j \in \mathcal{V}_1\}$	150
3	{4,..,12}	200	40	$\{(i,j) \mid i \in \mathcal{V}_3, j \in \mathcal{V}_2, j = (i-1)/3$ or $j = (i-1)/3+1$, if $j > 3$ then $j = 1\}$	100
4	{13,..,39}	100	10	$\{(i,j) \mid i \in \mathcal{V}_4, j \in \mathcal{V}_3, j = (i-1)/3$ or $j = (i-1)/3+1$, if $j > 12$ then $j = 4\}$	50

performed similarly to ERP by sending release packets in reply to all FOUND responses but the one that advertises the path with minimum cost. If the agent does not find enough local resources to reserve for the requested object, it releases the resources on the selected path and rejects the query.

Our experiments show that ERO results in lower blocking ratio, hence, higher utilization (see Section 5.2.1).

5 Performance Evaluation

We have extended the NS (Network Simulator) networking event simulator to incorporate our object delivery scheme.[9] We conducted several experiments via simulation to 1) evaluate performances of our proposed optional object location policies, cost functions, and resource reservation policies and 2) verify that our object delivery scheme meets our objectives.

5.1 Simulation Model

With our experiments, we simulated a 4-level 3-ary pure hierarchy and a 4-level redundant hierarchy (RedHi) with $d = 2$ as its degree of redundancy (i.e., the number of parents for each node). The detailed setup and configuration parameters of these hierarchies are shown in Table 4 and Table 5, respectively. The configuration parameters are selected to estimate the capabilities of the products available in the market. Particularly, head-ends are assumed to have storage capacity and streaming bandwidth of 10 GB and 100 Mb/s, respectively, both of which are compatible with off-the-shelf PCs used as CCMS [46]. Since head-ends are numerous, to reduce the cost of the DCMS, in practice, low-cost nodes are used as head-ends. At higher levels (i.e., levels with less l), which have fewer number of nodes, available resources per node are scaled up. The links of the hierarchies have bandwidth x-multiple of 50 Mb/s to estimate SONET OC-x carriers. Fan-in of each node (total bandwidth of the links to parents) is equal to bandwidth of the node itself. Total link/node bandwidth available at each level decreases in higher levels of the hierarchies because we presume more resources are required to stream the objects cached at lower levels of the hierarchies. The total amounts of streaming resources assigned to the two hierarchies are identical; RedHi redundancy is in connectivity rather than streaming resources. Our simulator assumes delay with normal distribution[10] for links of the hierarchies. Considering the propagation delay of typical

9. The source code for our extension of NS is available at ftp://zahedaan.usc.edu/GMeN/simulation.

10. $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$.

TABLE 6
Simulation Parameters

Parameter	Definition	Value
T_M	Duration of an object	1 hours
S_M	Storage size of an object	0.5 GB (MPEG-4 compressed)
B_M	Bandwidth requirement of an object	1 Mb/s
N_H	Number of head-ends	27
B_H	Bandwidth of a head-end	100 Mb/s
T_S	Duration of each simulation execution (virtual)	100 hours

terrestrial long-haul links, the normal distribution is defined with expected value $\mu = 70ms$ and standard deviation $\sigma = \mu/5$.

A static population of 500 objects (e.g., movies) is assumed. All objects are initially stored at the root node(s) of the hierarchies, from which they are distributed through the DCMS network and replicated at other nodes. We do not exclude statistics collected during the warm-up period of object distribution from experiment results; we presume the possible impact of warm-up is the same for all simulation executions. To capture the popularity profile of the objects, we assume Zipf distribution[11] with skew factor $\theta = 0.271$. It is shown that this distribution accurately models the popularity of rental movies [47]. All the objects are assumed to have identical specifications (see Table 6).

In an ideal case, optimistically, a hierarchy can serve up to L_{max} requests during a simulation execution:

$$L_{max} = \frac{B_H}{B_M} \times \frac{T_S}{T_M} \times N_H.$$

To emulate α percent load (i.e., $\frac{\alpha}{100} \times L_{max}$ requests) for a hierarchy, the ith head-end generates requests by discrete simulation of Poisson distribution[12] with λ_i as the expected number of requests generated during the simulation execution. λ_i parameters are randomly selected so that:

$$\Sigma_{i=1}^{N_H} \lambda_i = \frac{\alpha}{100} \times L_{max}.$$

For each execution of the simulation, we evaluate resource utilization in the hierarchies based on the blocking ratio of the requests, B:

$$B = \frac{\text{Number of rejected requests during } T_s}{\text{Total number of requests generated during } T_x} \times 100.$$

A request is rejected if DCMS cannot find any path with enough streaming resources to serve the request. Also, we define I_B as the improvement of the blocking ratio in RedHi as compared to the pure hierarchy:

$$I_B = B_{Pure} - B_{RedHi}.$$

With our experiments reported in Section 5.2, we used *inclusive* propagation with RedHi and *parent-only* propagation with the pure hierarchy; also, we used $\texttt{path\_freeBW}_{average}$ as the cost function and *ERO* as the reservation policy, unless stated otherwise. Besides, since the results of our experiments are often quite consistent (standard deviation is always less than 5 percent of the average value), we only report on the average values of the resulting quantities and exclude higher moments.

11. With a Zipf distribution, if the objects are sorted according to the access frequency, then the access frequency for the ith movie is given by $f_i = \frac{C}{i^{1-\theta}}$, where θ is the skew factor of the distribution and C is the normalization constant.

12. $\Pr[k \text{ requests arrive in time interval } T_S] = \frac{\lambda_i^k}{k!} e^{-\lambda_i}$, where $\lambda_i = \mu T_S$ and μ is the mean arrival rate in request per time unit.

5.2 Experimental Results

5.2.1 Performance of Optional Policies and Cost Functions

In Section 4, we proposed three propagation policies for object location, two variations of a cost function to evaluate the load of the streaming paths, and two policies for resource reservation. We conducted some comparative experiments to assess relative performance of these techniques under various loads. We delay discussing the propagation policies until Section 5.2.2, where we study their impact on resource utilization.

$\texttt{path\_freeBW}_{bottleneck}$ and $\texttt{path\_freeBW}_{average}$, the two variations of our cost function, are used for load balancing between paths with identical lengths (see Section 4.3). $\texttt{path\_freeBW}_{bottleneck}$ only considers the load on the bottleneck of a path, whereas $\texttt{path\_freeBW}_{average}$ evaluates the load of the entire path. Fig. 7a shows that $\texttt{path\_freeBW}_{average}$ marginally outperforms $\texttt{path\_freeBW}_{bottleneck}$ by capturing more data about the available resources along the path.

In Section 4.4, we discussed two pessimistic (ERP) and optimistic (ERO) resource reservation policies. Fig. 7b depicts the relative performance of these policies. Since, with ERP, resources are vastly over-reserved at all nodes that receive at least one privileged query packet, total available resources of the DCMS network are underestimated by the object delivery mechanism. As a result of this underestimation of resources, more requests are rejected when ERP is used for reservation, hence, higher blocking ratio. Although ERO outperforms ERP under the entire range of the load sinc, with higher loads, DCMS resources exhaust, as load increases, relative advantage of ERO over ERP decreases.

5.2.2 Resource Utilization: RedHi vs. Pure Hierarchy

In Section 5.2.2a and Section 5.2.2b, we report on the relative resource utilization of the hierarchies assuming 1) no link failures or 2) possible link failures, respectively.

5.2.2a Without Link Failures. Fig. 8a depicts the blocking ratio of several hierarchy-setups under various loads: the pure hierarchy with parent-only propagation policy as compared to RedHi with parent-only, sibling, and inclusive propagation policies. As expected, the blocking ratio is an increasing function of the load in all cases. The number of rejected requests depends on both the load (proportionally)

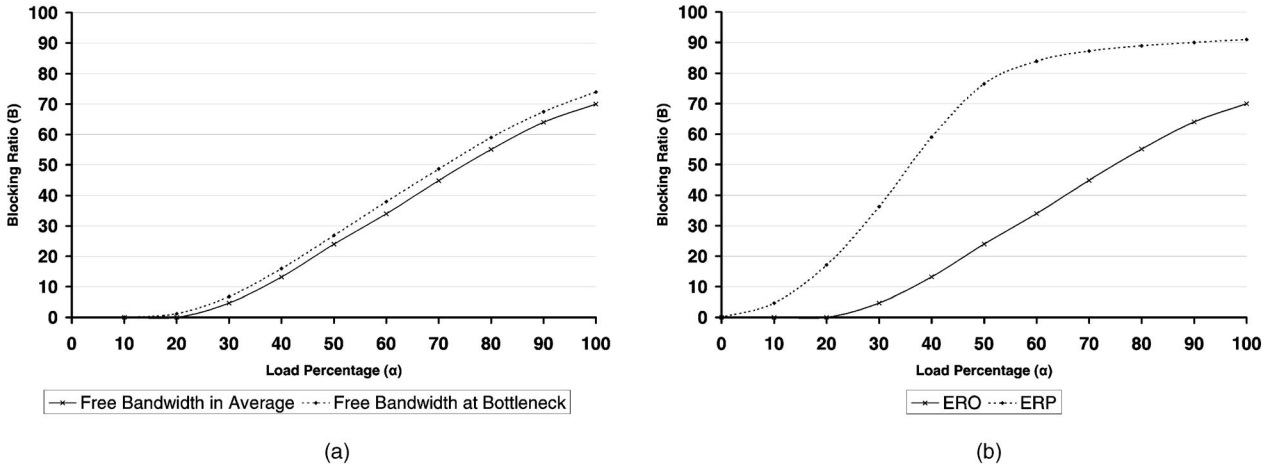

Fig. 7. Comparative evaluation of the optional techniques. (a) Cost function. (b) Reservation policies.

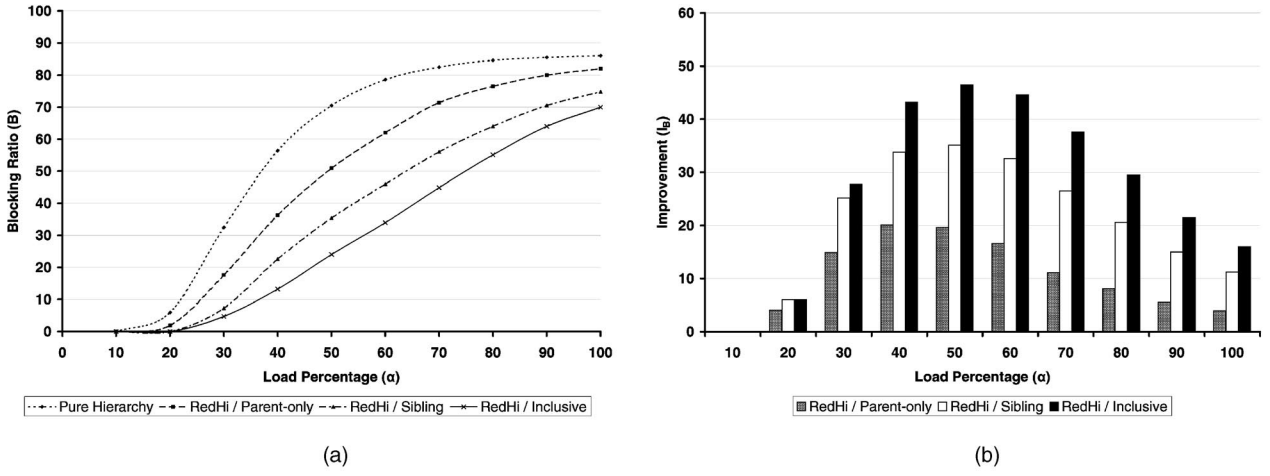

Fig. 8. Resource utilization (without link failures). (a) Various hierarchy-setups. (b) RedHi utilization improvement over the pure hierarchy.

and the available resources of the system (reciprocally). However, as the load increases, available system resources decrease. Therefore, the number of rejected requests grows faster than the load, hence, increasing blocking ratio (see the definition of B in Section 5.1).

The blocking ratio of the pure hierarchy has the highest growth rate. As the capability of resource sharing increases by 1) employing a highly-connected topology (RedHi) and 2) applying propagation policies with wider coverage, due to statistical multiplexing of the resources, the growth rate of the blocking ratio decreases, hence, a large improvement on B with the average loads. On the other hand, with higher loads, since system resources are close to exhaustion, resource sharing is less beneficial. Therefore, while the pure hierarchy is already saturated, the blocking ratio with other setups grows rapidly. As a result, the offset between setups decreases such that, at L_{max}, the offset is marginal. If we increase the load further (of course, any additional load is certainly rejected), all systems will asymptotically converge to $B = 100\%$.

Fig. 8b depicts the relative improvement of the blocking ratio at RedHi with various propagation policies as compared to the pure hierarchy (with the parent-only propagation policy). On the one hand, this figure shows the benefit of using the RedHi topology to achieve higher resource utilization (RedHi/parent-only setup vs. the pure hierarchy/parent-only setup). On the other hand, the advantage of applying the propagation policies with wider coverage is illustrated by observing the relative increase in improvement of B with RedHi/parent-only, RedHi/sibling, and RedHi/inclusive setups. Particularly, it is important to note that, although the sibling policy has much less coverage as compared to the inclusive policy, applying this propagation policy results in fairly high improvement in B.

5.2.2b With Link Failures. Since the RedHi topology has a higher connectivity as compared to the pure hierarchy, we expect even higher improvement in B when some of the network links fail. Assuming links of the DCMS network are physical links (i.e., logical and physical topologies of the DCMS network are the same[13]), we simulate the link failures by disabling the links selected randomly with a Poisson arrival time distribution with $\lambda = 10$ as the expected number of link failures diring T_S (i.e., $MTBF = \frac{T_S}{\lambda} = 10$ hours) Therefore, on average, nearly 25 percent of the links in the pure hierarchy and 12.5 percent of the links

13. In the case of the DCMS networks with logical links, a single physical link failure may result in multiple logical link failures.

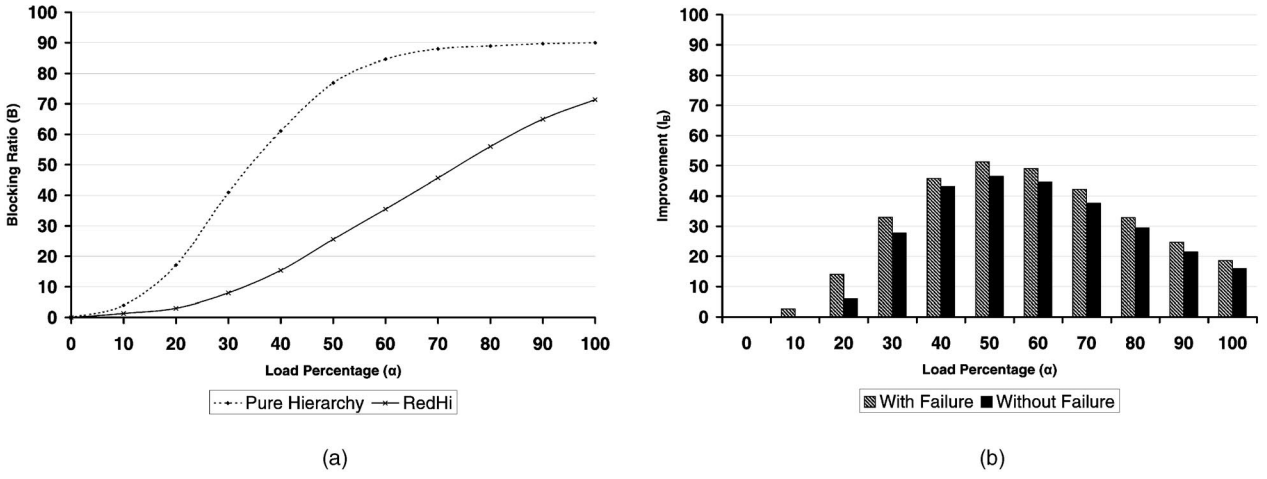

Fig. 9. Resource utilization (with link failures). (a) The pure hierarchy vs. RedHi. (b) RedHi utilization improvement: with failure vs. without failure.

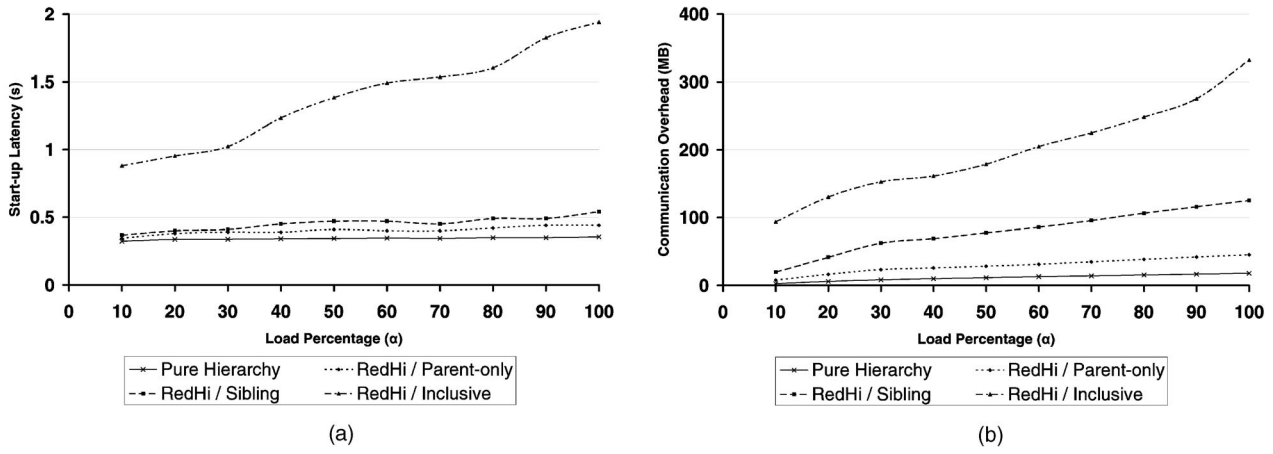

Fig. 10. Start-up latency and communication overhead with the proposed delivery mechanism. (a) Average start-up latency. (b) Average communication overhead.

in RedHi fail during the simulation. The $MTTR$ for each link is set to two hours.

Fig. 9a depicts the blocking ratio of the RedHi/inclusive setup as compared to the pure hierarchy/parent-only setup, both under the failure condition. The behavior of the two setups is similar to the no-failure condition (see Fig. 8a) with an expected relative increase in B for both setups. However, as illustrated in Fig. 9b, the improvement in B due to using RedHi/inclusive (instead of the pure hierarchy/parent-only) is more under the failure condition as compared to the no-failure condition. This growth in improvement of B can be attributed to the higher connectivity of the RedHi that results in less probability for node isolation.

5.2.3 Latency and Communication Overhead

We conducted some experiments to verify that the start-up latency and the communication overhead with our proposed delivery mechanism is acceptable.

Fig. 10a depicts the average start-up latency for each request in various hierarchy setups. The start-up latency for a request is the duration of the time period between the arrival time of the request and the starting time of the streaming. The average latency is calculated over all requests arrived during T_S and served successfully. As illustrated in the figure, the start-up latency reasonably grows as connectivity of the topology (from the pure hierarchy to RedHi) and the coverage of propagation policy increase. However, even for the RedHi/inclusive setup, the latency is still less than two seconds, which as compared to the five-minute expected user reneging time is negligible (see Section 2).

Fig. 10b shows the total amount of the control data exchanged between the resource management agents in various hierarchy setups during the entire simulation (T_S). Similarly, the greater the network connectivity and the propagation coverage, the greater the communication overhead. Obviously, the RedHi/inclusive setup involves more communication between the network agents and results in the largest amount of overhead. With the RedHi/inclusive setup, the communication overhead per each successfully served request amounts to:

$$\frac{350 \times 10^6 \times 8}{L_{max} \times (1 - \frac{B_{L_{max}}}{100})} = \frac{350 \times 10^6 \times 8}{27 \times 10^4 \times (1 - 0.7)} \approx 35Kb.$$

Considering a one-hour duration of the objects of our simulation model, this overhead is equivalent to $\frac{35Kb}{3600s} \approx 10b/s$

per request, which is certainly negligible for a 1Mb/s bandwidth object. Even if we need to consider the bursty nature of the overhead, the worst-case 35Kb communication overhead per request is negligible as compared to the size of the typical continuous media objects. If the size of the objects were of the same order of the size of this overhead, we would prefer to use File Transfer (e.g., ftp) to serve the objects rather than serving them continuously.

6 Related Work

In this section, we review some related studies that investigate one or more of the three tasks we support with our object delivery scheme. The *object location* problem has been studied in various but closely related contexts such as tracking mobile users [39], locating objects for distributed object programming [48], locating nearby copies of replicated servers [49], and resource discovery [50]. *Directory service* is a common mechanism often used for object location and many resource management systems include centralized directory servers [51], [52]. These servers usually comprise object state databases and query servers for object location. All client requests for objects are redirected to the directory server, where the replicas of the requested object are located, via object state database and, usually, a particular cost function is used to select the most appropriate object server to respond the request. To enhance both scalability and robustness of the object location mechanisms (particularly for those used with large, distributed, and usually interconnected systems that carry many objects), many researchers have introduced various distributed (and possibly replicated) directory structures that maintain states of the objects in a distributed fashion among multiple directory servers. As discussed in Section 3, decentralization of the service can cause an increase in the response-time and resource management overhead of such systems. To address this problem, besides replication, these systems usually exploit the locality characteristic of user access patterns and map a hierarchy of regional directories to the network (i.e., the system carrying the objects) so that most read and update accesses to the object states within a region are responded to by the regional directory server [53]. Consequently, response-time and overhead of resource management are drastically reduced. For instance, Awerbuch and Peleg [39] introduce a distributed directory server that is able to locate migrating objects with a communication overhead within a polylogarithmic factor of the lower bound; however, their directory service does not support replicated objects. Bartal et al. [54] also propose a distributed directory service and a "competitive" access algorithm with optimal communication cost for mesh and pure tree network topologies. The hierarchy of directory servers developed by Steen et al. [48] considers the "stability" of objects in each region and dynamically adapts to object migrations, but, as far as resource utilization optimization is concerned, it cannot distinguish the most appropriate replica of an object to be served. Plaxton et al. [49] use a virtual tree of object location agents that maps one-to-one into the network to locate replicas of an object. For each object, there is a separate tree and each node of the tree keeps track of the object replicas located at the entire subtree rooted at the node. On a request for an object, if a node does not find any replica of the object at its subtree, it forwards the request to its parent and this process repeats until a replica of the object is located. Due to the limited "view" of each agent, this approach does not guarantee locating the most appropriate object replica (i.e., the one that results in optimal resource utilization). This drawback might not be important in a system with small objects that are served with a low amount of resources and a good response-time requirement, however, for streaming applications with continuous media objects, which have intensive resource requirements and longer tolerable start-up latency, it will result in low resource utilization.

Our object delivery scheme can be considered as an extreme case distributed directory service to achieve optimal scalability and robustness. As compared to a normal distributed directory service, as far as response-time and resource management overhead are concerned this approach minimizes the overhead of the update to the states (updates local metadata only), while it results in increase in the read access overhead. As discussed in Section 3, we alleviate the overhead of the read access to negligible level by collapsion of the object delivery tasks. It is important to note that almost all distributed directory structures assume a pure tree or a mesh topology for the network that contain the objects, whereas our scheme assumes a RedHi topology for the DCMS network. Besides, some systems rely on specific capabilities of the underlying telecommunication infrastructures, such as anycasting and multicasting, as the main means for object location [55]. This approach makes these systems network dependent. Our application layer scheme does not assume any of these specific capabilities for the underlying infrastructure of the DCMS network rather than point-to-point communication between adjacent nodes. A hybrid system in which both normal unicast and multicast communications can be used for object location was recently introduced by Cranor et al. [56].

The *path selection* problem is well studied with dynamic routing algorithms such as distance vector (RIP) [57] and link state (OSPF) [58]. These algorithms enable each node of a network to find the "best" path between itself and other nodes of the network. A cost function (e.g., hop count) is usually defined to evaluate the costs of all paths between two nodes and to select the best path (i.e., the path with minimum cost) among them. Our path selection algorithm is supposed to find the best path among the paths between a single node, a head-end, and *multiple* potential object server nodes of the DCMS network, which are located by the object location algorithm. With multiple object servers, each probably having multiple paths to the head-end in the RedHi redundant topology, a simple cost function such as hop count cannot resolve the highly probable tie conditions in the costs of the paths efficiently so that it results in low resource utilization. To achieve high resource utilization, our cost function considers load balancing by breaking the ties based on the current traffic load of the paths. Moreover, our path selection algorithm runs at the application layer on the DCMS network, whereas normal routing algorithms run at the network layer within the underlying

telecommunication infrastructures. Therefore, our cost measures relate to streaming resources rather than networking resources, e.g., we consider CCMS available bandwidth rather than link propagation delay as a cost measure.

Finally, the *resource reservation* problem is also studied under protocols such as RSVP [59] and ST-2 [60], which intend to provide communication links with guaranteed service by reserving resources across the network. These protocols assume the communicating peers are already known and describe the procedure to reserve the required resources across a path between the peers, whereas, since, with our approach, reservation is integrated into object location and path selection, the peers (particularly the selected object server) and the selected path between peers are not known in advance. Our resource reservation algorithm runs at the application layer to reserve resources in the DCMS network, whereas the reservation protocols mentioned above run at the network layer within the underlying telecommunication infrastructures. Cao et al. [61] and Huang et al. [62] propose two typical application layer centralized and semidistributed resource reservation schemes, respectively. Our object delivery scheme supports fully distributed resource reservation.

7 Conclusion and Future Work

In this study, we developed an object delivery scheme for a DCMS network assuming the RedHi topology. The unique characteristics of this scheme are as follows:

- Given the unique features of continuous media objects, our object delivery scheme is designed as a fully *decentralized* resource management system that guarantees scalability and robustness as well as negligible start-up latency and resource management overhead.
- Our object delivery scheme *collapses* three different resource management tasks into one *distributed* mechanism that executes all tasks integrated.
- Our object delivery scheme is designed as an *application layer* resource management middleware that is independent of the underlying telecommunication infrastructure of the DCMS network. It does not assume any specific capabilities (e.g., multicasting and anycasting) for the telecommunication infrastructure except for providing point-to-point communication between adjacent nodes of the DCMS network.

With our object delivery scheme, we introduced a cost function as well as various object location and path reservation policies. We performed extensive experiments via simulation to evaluate performances of these policies as well as overall performance of our object delivery scheme. Results of our experiments show that:

- *Inclusive*, $\texttt{path\_freeBW}_{average}$, and *ERO* outperform other propagation, cost evaluation, and reservation techniques, respectively.
- Our delivery scheme with a RedHi DCMS network can achieve up to 50 percent improvement in resource utilization over a pure hierarchy.
- The start-up latency and the communication overhead per each request can be as low as two seconds and 10b/s, respectively.

We intend to extend this study in several ways. First, we would like to extend the object delivery mechanism to support VCR functions for interactive object delivery. Second, we also intend to merge the bandwidth renegotiation idea into our object delivery scheme to provide a hybrid solution for distributed continuous media applications. Finally, we are investigating techniques to incorporate self-configuration/warm-expansion capabilities into resource management.

Appendix

Definition 2 (Leveled Graph). *"Leveled Graph" $\mathcal{G} = (\mathcal{V}, \mathcal{E}, n)$ is a graph (with $\mathcal{V}$ as the set of nodes and $\mathcal{E}$ as the set of edges) that satisfies the following two conditions:*

1. *There exists a partition $\{\mathcal{V}_i \mid i \in 1..n,\ n > 1\}$ of $\mathcal{V}$ such that $\forall e \in \mathcal{E}$, if $e = (u, w)$, $\exists j \in 2..n$ such that $u \in \mathcal{V}_{j-1}$ and $w \in \mathcal{V}_j$. In such a case, u is a* parent *node of w, and w is a* child *node of u.*
2. *$\forall v \in \mathcal{V}_1, \exists e \in \mathcal{E}$ such that $e = (v, w)$ and v is a parent node of w. Moreover, $\forall v \in \mathcal{V}_{i \in 2..n}$, $\exists e \in \mathcal{E}$ such that $e = (u, v)$ and v is a child node of u.*

The number n is the number of levels *(i.e., partition members) in the leveled graph $\mathcal{G}$. If $v \in \mathcal{V}_1$, v is a* root *node of $\mathcal{G}$. Nodes that have no children are termed* head-ends *or* leaves *of $\mathcal{G}$.*

Definition 3 (Pure Hierarchy). *A "Pure Hierarchy" is a leveled graph in which:*

1. *There is one and only one root and*
2. *Every node, except the root, has one and only one parent node.*

Definition 4 (Redundant Hierarchy (RedHi)). *A "Redundant Hierarchy" is a leveled graph in which:*

1. *There are at least two roots and*
2. *Every node, except the roots, have at least two parent nodes.*

Acknowledgments

The authors would like to thank Mohammad H. Alshayeji for his contributions to the first version of this paper. They are also grateful for the assistance given to them by Vasan R. Sundar in conducting the experiments. This research has been funded in part by US National Science Foundation grants EEC-9529152 (IMSC ERC) and IIS-0082826, National Insistutes of US Health-National Library of Medicine (NIH-NLM) grant no. R01-LM07061, Defense Advanced Research Projects Agency (DARPA) and US Air Force under agreement no. F30602-99-1-0524, and unrestricted cash/equipment gifts from the Okawa Foundation, Microsoft, NCR and SUN.

REFERENCES

[1] C.N. Judice, E.J. Addeo, M.I. Eiger, and H.L. Lemberg, "Video-on-Demand: A Wideband Service or Myth?" *Proc. Int'l Conf. Comm. (ICC '86),* June 1986.

[2] W.D. Sincoskie, "Video-on-Demand: Is It Feasible?" *Proc. GLOBECOM '90,* vol. 1, pp. 201-205, 1990.

[3] T. Little and D. Venkatesh, "Prospects for Interactive Video-on-Demand," *IEEE Multimedia,* col. 1, no. 3, pp. 14-24, Fall 1994.

[4] J. Crowcroft, P.T. Kirstein, and D. Timm, "Multimedia Teleconferencing over International Packet Switched Networks," *Proc. TRICOMM '91, IEEE Conf. Comm. Distributed Applications and Systems,* pp. 23-33, 1991.

[5] F.A. Tobagi, "Distance Learning with Digital Video," *IEEE Multimedia,* vol. 2, no. 1, pp. 90-94, Spring 1995.

[6] B. Furht, D. Kalra, F.L. Kitson, A.A. Rodriguez, and W.E. Wall, "Design Issues for Interactive Television Systems," *Computer,* vol. 28, no. 5, pp. 25-38, May 1995.

[7] D. Anderson and G. Homesy, "A Continuous Media I/O Server and Its Synchronization," *Computer,* vol. 24, no. 10, Oct. 1991.

[8] B. Ozden, R. Rastogi, and A. Silberschatz, "A Framework for the Storage and Retrieval of Continuous Media Data," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* May 1995.

[9] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.W. Li, "Mitra: A Scalable Continuous Media Server," *Multimedia Tools and Applications J.,* vol. 5, no. 1, pp. 79-108, Apr. 1997.

[10] C. Chou, L. Golubchik, and J.C.S. Lui, "A Performance Study of Dynamic Replication Techniques in Continuous Media Servers" *Proc. Eighth Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems,* pp. 256-263, Aug. 2000.

[11] R. Nagarajan and J.F. Kurose, "On Defining, Computing and Guaranteeing Quality-of-Service in High-Speed Networks," *Proc. INFOCOM '92,* vol. 3, pp. 2016-2025, 1992.

[12] J.F. Kurose, "Open Issue and Challenges in Proving Quality-of-Service Guarantees in High-Speed Networks," *ACM Computer Comm. Rev.,* vol. 23, no. 1, pp. 6-15, Jan. 1993.

[13] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for Next-Generation High-Speed Networks: Problems and Solutions," *IEEE Network,* vol. 12, no. 6, pp. 64-79, Nov. 1998.

[14] D. Ghosh, V. Sarangan, and R. Acharya, "Quality-of-Service Routing in IP Networks," *IEEE Trans. Multimedia,* vol. 3, no. 2, pp. 200-208, June 2001.

[15] J. Nussbaumer, B. Patel, F. Schaffa, and J. Sterbenz, "Network Requirements for Interactive Video-on-Demand," *IEEE J. Selected Areas in Comm.,* vol. 13, no. 5, pp. 779-787, June 1995.

[16] J.D. Salehi, Z.L. Zhang, J.F. Kurose, and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 222-231, May 1996.

[17] D. Aksoy, S. Zdonik, and M.J. Franklin, "Data Staging for On-Demand Broadcast," *Proc. VLDB 2001, Int'l Conf. Very Large Databases,* Sept. 2001.

[18] T.Y. Kim, B.H. Roh, and J.K. Kim, "Bandwidth Renegotiation with Traffic Smoothing and Joint Rate Control for VBR MPEG Video over ATM," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 10, no. 5, pp. 693-703, Aug. 2000.

[19] S.G. Chan and F. Tobagi, "Providing On-Demand Video Services Using Request Batching," *Proc. IEEE Int'l Conf. Comm. (ICC '98),* vol. 3, pp. 1716-1722, 1998.

[20] K.C. Almeroth and M.H. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 6, pp. 1110-1122, Aug. 1996.

[21] A.D. Gelman, H. Kobrinski, L.S. Smoot, S.B. Weinstein, M. Fortier, and D. Lemay, "A Store-and-Forward Architecture for Video-on-Demand Service," *Proc. IEEE Int'l Conf. Comm. (ICC '91),* vol. 2, pp. 842-846, 1991.

[22] L. De Giovanni, A.M. Langellotti, L.M. Patitucci, and L. Petrini, "Dimensioning of Hierarchical Storage for Video-on-Demand Services," *Proc. IEEE Int'l Conf. Comm. (ICC '94),* vol. 3, pp. 1739-1743, 1994.

[23] F. Schaffa and J.P. Nussbaumer, "On Bandwidth and Storage Tradeoffs in Multimedia Distribution Networks," *INFOCOM '95 Proc. 14th Ann. Joint Conf. IEEE Computer and Comm Soc.,* vol. 3, pp. 1020-1026, 1995.

[24] S.A. Barnett and G.J. Anido, "A Cost Comparison of Distributed and Centralized Approaches to Video-on-Demand," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 6, pp. 1173-1183, Aug. 1996.

[25] C. Shahabi, M. Alshayeji, and S. Wang, "A Redundant Hierarchical Structure for a Distributed Continuous Media Server," *Proc. Fourth European Workshop Interactive Distributed Multimedia Systems and Telecomm. Services (IDMS '97),* Sept. 1997.

[26] D.D. Clark and S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. Conf. Comm. Architectures and Protocols,* pp. 14-26, 1992.

[27] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *RFC Services,* Dec. 1998.

[28] J.M. McManus and K.W. Ross, "Video-on-Demand over ATM: Constant-Rate Transmission and Transport," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 6, pp. 1087-1098, Aug. 1996.

[29] C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan, "Information Caching for Delivery of Personalized Video Programs on Home Entertainment Channels," *Proc. Int'l Conf. Multimedia Computing and Systems,* pp. 214-223, 1994.

[30] R. Ramarao and V. Ramamoorthy, "Architectural Design of On-Demand Video Delivery Systems: The Spatio-Temporal Storage Allocation Problem," *Proc. IEEE Int'l Conf. Comm. (ICC '91),* vol. 1, pp. 506-510, 1991.

[31] J.D. Ryoo and S.S. Panwar, "Algorithms for Determining File Distribution in Networks with Multimedia Servers," *Proc. IEEE Int'l Conf. Comm. (ICC '99),* vol. 2, pp. 875-879, 1999.

[32] R. Lüling, "Static and Dynamic Mapping of Media Assets on a Network of Distributed Multimedia Information Servers," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems,* pp. 253-260, 1999.

[33] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Optimal Placement of Replicas in Trees with Read, Write, and Storage Costs," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 6, pp. 628-637, June 2001.

[34] H.J. Burch and F. Ercal, "Mapping the Internet," *Computer,* vol. 32, no. 4, pp. 97-102, Apr. 1999.

[35] A. Broido and K. Claffy, "Internet Topology: Connectivity of IP Graphs," *Cooperative Assoc. Internet Data Analysis (CAIDA),* July 2001. Available online at http://www.cadia.org.

[36] "NS—The Network Simulator," Information about NS is availabale at http://www.isi.edu/nsnam/ns/, Year???

[37] C. Leopold, *Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches,* first ed. John Wiley, 2001.

[38] S. Fortune and J. Willie, "Parallelism in Random Access Machines," *Proc. 10th ACM Symp. Theory of Computing,* pp. 114-118, 1978.

[39] B. Awerbuch and D. Peleg, "Concurrent Online Tracking of Mobile Users," *Proc. Conf. Comm., Architectures, and Protocols (ACM SIGCOMM '91),* pp. 221-233, Sept. 1991.

[40] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. Second ACM Int'l Conf. Multimedia (MM '94),* pp. 15-23, Oct. 1994.

[41] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Servers," *Proc. Third IEEE Int'l Conf. Multimedia Computing and Systems,* pp. 253-258, 1996.

[42] K.A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," *Proc. Sixth ACM Int'l Conf. Multimedia (MM '98),* pp. 191-200, Sept. 1998.

[43] A.J. Smith, "Bibliography on Paging and Related Topics," *Operating Systems Rev.,* vol. 12, pp. 39-56, 1978.

[44] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2326, Apr. 1998.

[45] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," *Computer Networks,* vol. 7, pp. 395-405, Dec. 1983.

[46] R. Zimmermann, K. Fu, C. Shahabi, D. Yao, and H. Zhu, "Yima: Design and Evaluation of a Streaming Media System for Residential Broadband Services," *Proc. VLDB 2001 Workshop Databases in Telecomm. (DBTel 2001),* Sept. 2001.

[47] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. ACM Multimedia Conf.,* pp. 15-24, 1994.

[48] M.V. Steen, F.J. Hauck, P. Homburg, and A.S. Tanenbaum, "Locating Objects in Wide-Area Systems," *IEEE Comm.,* vo. 36, no. 1, pp. 104-109, Jan. 1998.

[49] C.G. Plaxton, R. Rajaraman, and A.W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Theory of Computing Systems,* vol. 32, pp. 241-280, 1999.
[50] K. Obraczka, P.B. Danzig, and S.H. Li, "Internet Resource Discovery Services," *Computer,* vol. 26, no. 9, pp. 8-22, Sept. 1993.
[51] D.W. Brubeck and L.A. Rowe, "Hierarchical Storage Management in a Distributed VoD System," *IEEE Multimedia,* vol. 3, no. 3, pp. 37-47, Fall 1996.
[52] H. Yu and C.P. Low, and Y. Atif, "Design Issues on Video-on-Demand Resource Management," *Proc. IEEE Int'l Conf. Networks (ICON 2000),* pp. 199-203, 2000.
[53] B. Awerbuch and D. Peleg, "Sparse Partitionsm" *Proc. 31st IEEE Symp. Foundations of Computer Science,* pp. 503-513, Oct. 1990.
[54] Y. Bartal, A. Fiat, and Y. Rabani, "Competitive Algorithms for Distributed Data Management," *Proc. 24th Ann. ACM Symp. Theory of Computing,* pp. 39-50, May 1992.
[55] S. Stavros, K. Nectarios, and P. George, "A Distributed Media Server Management Scheme," *Proc. 10th MELECON 2000,* pp. 6-10, 2000.
[56] C.D. Cranor, M. Green, C. Kalmanek, D. Shur, S. Sibal, J.E. Van der Merwe, and C.J. Sreenan, "Enhanced Streaming Services in a Content Distribution Network," *IEEE Internet Computing,* vol. 5, no. 4, pp. 66-75, Aug. 2001.
[57] G. Malkin, "RIP Version 2," RFC 2453, Nov. 1998.
[58] J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998.
[59] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network,* vol. 7, no. 5, pp. 8-18, Sept. 1993.
[60] C. Topolcic, "Experimental Internet Stream Protocol, Version 2 (ST-II)," RFC 1190, Oct. 1990.
[61] F. Cao, J. Smith, and K. Takahashi, "An Architecture of Distributed Media Servers for Supporting Guaranteed QoS and Media Indexing," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems '99,* vol. 2, pp. 1-5, 1999.
[62] J. Huang, Y. Wang, N.R. Vaidyanathan, and F. Cao, "Grms: A Global Resource Management System for Distributed QoS and Criticality Support" *Proc. IEEE Int'l Conf. Multimedia Computing and Systems '97*, pp. 424-432, 1997.

Cyrus Shahabi received the BS degree in computer engineering from Sharif University of Technology at Tehran and the MS and PhD degrees in computer science from the University of Southern California (USC) in August 1993 and 1996, respectively. He is currently an assistant professor and the director of the Information Laboratory at the Computer Science Department at USC. He is also the director of the Information Management Research Area at the Integrated Media Systems Center (IMSC) and a US National Science Foundation Engineering Research Center at USC. He has published more than 50 articles, book chapters, and conference papers in the areas of databases and multimedia. His current research interests include multidimensional databases, multimedia servers, and data mining. He is a member of the IEEE and the IEEE Computer Society.

Farnoush Banaei-Kashani received the BS degree in computer engineering from Sharif University of Technology at Tehran in 1995 and the MS degree in electrical engineering (computer networks) from the University of Southern California, Los Angeles, in 2001. He is currently working towards the PhD degree in computer science at USC. He did research at the Electrical Research Center (ERC) prior to coming to the United States and is currently a research assistant working on distributed continuous media systems at the Integrated Media Systems Center (IMSC)—Information Laboratory at the Computer Science Department at USC. He is a student member of the IEEE and the IEEE Computer Society.