Caching multicast protocol for on-demand video delivery *

Kien A. Hua, Duc A. Tran, Roy Villafane

School of Computer Science
University of Central Florida
Orlando, FL 32816, U. S. A.

ABSTRACT

Despite advances in networking technology, the limitation of the server bandwidth prevents multimedia applications from taking full advantage of next-generation networks. This constraint sets a hard limit on the number of users the server is able to support simultaneously. To address this bottleneck, we propose a *Caching Multicast Protocol* (CMP) to leverage the in-network bandwidth. Our solution caches video streams in the routers to facilitate services in the immediate future. In other words, the network storage is managed as a huge "video server" to allow the application to scale far beyond the physical limitation of its video server. The tremendous increase in the service bandwidth also enables the system to provide true on-demand services. To assess the effectiveness of this technique, we develop a detailed simulator to compare its performance with that of our earlier scheme called *Chaining*. The simulation results indicate that CMP is substantially better with many desirable properties as follows: (1) it is optimized to reduce traffic congestion; (2) it uses much less caching space; (3) client workstations are not involved in the caching protocol; (4) it can work on the network layer to leverage modern routers.

Keywords: Video on demand, multimedia communications, multicast protocol, router, network caching

1. INTRODUCTION

The desire to deploy multimedia applications over broadband networks has introduced new challenges for storage and communication of multimedia objects. In this paper, we focus on video and audio data that must be presented to the users in a continuous manner. Without loss of generality, we will only refer to video data in this paper though the techniques can also be used for audio.

In a multimedia information system, a large media server is typically used to store video files. Upon a service request, the desired video is transmitted to the remote user over a high-speed network. Although the aggregate bandwidth of communication networks has improved tremendously, the performance of such a system is severely constrained by the bandwidth limitation of the media server. It is desirable to overcome this hurdle in order to take advantage of the in-network bandwidth, i.e., the aggregate bandwidth of the entire network. Two approaches can be used to address this bottleneck: (1) placing multiple media servers at strategic locations to exploit the aggregate bandwidth of the servers;(2) using multicast to facilitate data sharing. The first approach is simple, however, it is expensive to install and maintain multiple data centers. A good design should also exploit the latter approach to keep the degree of distribution to a minimum. Technology such as IP Multicast[1,2] is available today for such an implementation. In this paper, we focus on improving the effectiveness of the multicast mechanism. There are a number of multicast schemes that have been proposed for video data:

- Periodic Broadcast[3–6]: In this approach, each video is partitioned into a number of segments, each repeatedly broadcast on its own communication channel (e.g., multicast group). To receive a service, a client tunes to the appropriate channels to download the desired video. This strategy guarantees the service delay to be no more than the broadcast period of the first segment. To ensure acceptable delays, this segment can be made small. A major advantage of this approach is that the required server bandwidth is independent of the number of users the system is designed to support. Each video, however, requires substantial bandwidth. This requirement renders this approach suitable only for very popular videos.

* This research is partially supported by the National Science Foundation grant ANI-9714591.
Author information: K.A.H.; Email: kienhua@cs.ucf.edu; D.A.T.; Email: dtran@cs.ucf.edu; R.V.; Email: villafan@cs.ucf.edu.

- Batching[7,8]: This approach delays service requests for a particular video hoping that more will arrive within the current batching interval. All these requests can be served together using a single multicast. Compared to periodic broadcast, batching is effective for a wider range of videos. However, it is less effective for very popular videos. Studies have shown that a hybrid of the two techniques provides the best performance.[9]

- Chaining[10]: In the two approaches above, since each request must wait for the next multicast, they cannot offer true on-demand services. Another drawback is that the number of concurrent multicasts is still constrained by the bandwidth limitation of the server. To address these issues, we proposed an on-demand multicast technique, called *Chaining*.[10] This approach manages the disk buffers in the client machines as a huge network cache. Each client is capable of caching the requested video and pipelining it to other clients in the downstream at a later time. In other words, a multicast tree can grow dynamically at the application level to accommodate future requests for the same video. This strategy allows the application to scale far beyond the physical limitation of the video server. Furthermore, since the multicasts are performed on demand, true on-demand services can be achieved.

- Patching[11–13]: Patching is another on-demand multicast technique. When a client requests a video, it joins an on-going multicast of the video. The multicast data is temporarily cached in the local disk as the client plays back the leading portion of the video arriving on a separate channel called the *patching channel*. When the playback of the patching data is completed, the client switches to play the multicast data cached in the local buffer. This approach also can offer true on-demand services. It is simpler than Chaining because clients do not have to serve other clients. However, since the server is the only source for data, this approach cannot utilize the in-network bandwidth as effectively as Chaining can.

In general, multicast techniques try to reduce the demand on the server bandwidth by exploiting in-network bandwidth. It is obvious that Chaining is the best among the above techniques in achieving this goal. In this scheme, every client who uses the service must contribute its own resources (i.e., disk space and bandwidth) to the environment. As a result, each service request can be seen as a contributor, rather than just a burden to the server. This unique characteristic makes Chaining much more scalable than the other methods.[10]

Although Chaining offers outstanding performance, it still has a few drawbacks. While using client resources for later multicasts seems to be equitable, the matter of reliability should not be ignored. Handling sudden terminations of the services in a chain can be quite complex. Another drawback is due to the fact that forwarded data must travel from one edge to another edge of the network. This approach results in very expensive network costs. To address these practical problems, we propose in this paper a new technique called *Caching Multicast Protocol* (CMP). This scheme caches video streams in the routers to provide future services to the local requests. This idea is consistent with the current trend of designing routers to support more complex forwarding logic. Our simulation results indicate that this protocol outperforms Chaining by a significant margin. Furthermore, CMP offers many desirable properties as follows: (1) it is optimized to reduce network costs; (2) it uses much less caching space; (3) client workstations are not involved in the caching protocol; (4) it can work on the network layer to leverage modern routers.

CMP is motivated by the observation that it is essential to get content, especially rich content (such as video), closer to the edge of the network and to the consumer in order to leverage new high-speed, "last-mile" technologies, such as xDSL and cable modem. We believe network caching is an effective way to achieve this goal.

Compared to the multicast techniques discussed previously, CMP is very different. In fact, they are really scheduling policies designed to use existing multicast techniques more efficiently. In this paper, we look deeper into the network layer to investigate routing and caching protocols for more efficient video communications. CMP is also different from many existing protocols which were designed to support general-purpose applications.[14–17] In contrast, CMP is customized for audio and video delivery.

The remainder of this paper is organized as follows. To make the paper self contained, we describe Chaining in more detail in Section 2. The proposed technique is presented in Section 3. Our simulation studies are discussed in Section 4. Finally, we give our concluding remarks in Section 5.

2. CHAINING

Two variations of Chaining have been studied.[10] They are described in this section.

2.1. Standard Chaining

The basic idea in Chaining is that each client station reserves a small amount of disk space for caching purposes. When a communication channel becomes available, it is scheduled to multicast a video to all the pending requests currently waiting for this video. As these clients play back the video, they cache the data in their disk buffers using a FIFO replacement policy. Thus, the first block of the video data remains in a buffer until the cache is full. Before that happens, any new requests can be serviced from these clients creating the "second generation" of a dynamic multicast tree. That is, the second-generation clients are chained to the first in order to pipeline the video data. Similarly, the third-generation can receive data from the second, and so forth. This process can continue until the first data block of the video is dropped out of the tree due to the cache replacement policy. When that happens, the server needs to initiate another dynamic multicast tree to serve future requests.

We observe that a dynamic multicast tree can grow to be very big, yet uses only one server channel. This explains the effectiveness of Chaining in using in-network bandwidth to reduce the demand on the server bandwidth. Another observation is that a tree can grow dynamically to accommodate new requests as soon as they arrive. True on-demand services, therefore, can be achieved.

2.2. Extended chaining

It is desirable to prolong the "life" of a dynamic multicast tree to save server bandwidth. This can be achieved by strategically deferring the services for some of the requests as follows. When we schedule a group of pending requests to join a new generation of a dynamic multicast tree, we can delay the youngest request to the next generation as long as it has not waited more than a predetermined threshold. The purpose is to make sure that the tree can grow at least another generation, and thereby buy some time for the tree to admit more requests.

The advantage of the above strategy is best illustrated using a state diagram as illustrated in Fig. 1. There is a state diagram for each video file in the system. The state diagram for conventional batching is shown in Fig. 1(a). It shows that a video file can be in one of two possible states:

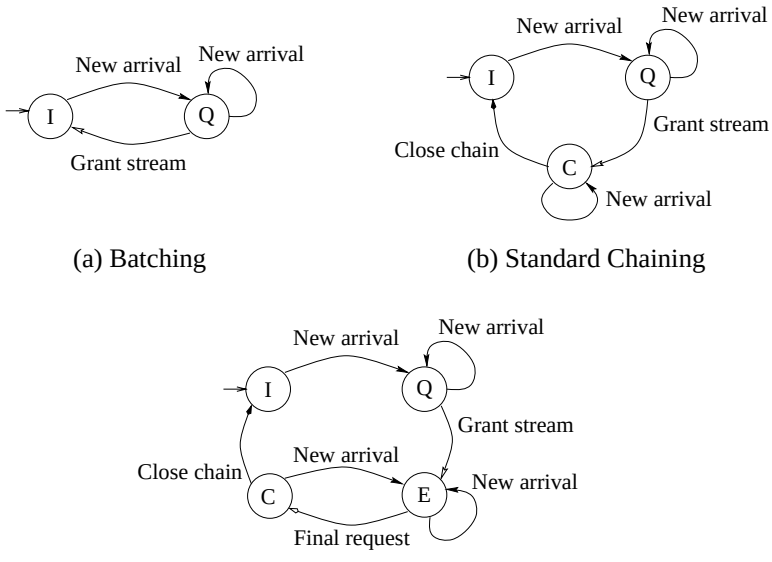

Figure 1. State transition diagrams of various schemes for multicast scheduling.

- *I* **State**: The video file is in I (or initial) state if it currently has no pending requests.
- *Q* **State**: The arrival of a new request causes the video to transit from the I state to the Q (or queuing) state, and a new batch to be created to hold the new arrival. Subsequent requests for this video are accumulated in this batch while the video remains in the Q state. Eventually, a video stream is allocated to serve this batch according to some queuing policy; and the video file returns to the I state.

The state diagram for Standard Chaining is shown in Fig. 1(b). It is similar to that of conventional batching except for one additional state.

- C **State**: Unlike regular batching, when a video stream is allocated to serve the current batch, the video file does not return to the I state. Instead, it goes into the C (or chaining) state signifying the existence of a dynamic multicast tree for this video. As long as new requests continue to arrive to keep this tree open, the video file remains in the C state. If the arrival of requests discontinues for an extended period of time, the tree is closed down and the video file returns to the I state.

The state diagram for Extended Chaining is shown in Fig. 1(c). This scheme has one additional state, E (or extending) state. Unlike Standard Chaining, the allocation of a video stream does not immediately move the video file into the C state. Instead, it transits into the E state.

- E **State**: In this state, the scheduler delays the youngest request to the next generation. As long as new requests continue to arrive, the video remains in the E state. If no new request arrives for an extended period of time, the video transits into the C state after the service is initiated for the last delayed request. We note that if new requests begin to arrive again at this time, the video file can return to the E state; otherwise, it returns to the I state after a period of missing new arrivals.

We note that the Q state is a waiting state. Service requests arriving during this state must wait. To achieve low service latency, batching requires substantial server bandwidth to keep the duration of the Q state short. In general, a technique is more demanding on the server bandwidth if it returns to the I state more often. Standard Chaining reduces this frequency by adding state C. Extended Chaining further reduces it by adding yet another state E. In addition to saving server resources, Chaining offers better service delays because the videos are in C state most of the time, which is not a waiting state.

3. CACHING MULTICAST PROTOCOL (CMP)

A performance limitation of Chaining is due to the fact that data must be forwarded from client to client. This results in data traveling from one edge of the network to another. Such a communication strategy makes the network cost very high. CMP addresses this problem by caching data in the network routers. This approach allows us to forward data from within the network significantly reducing the network costs. We present this novel idea in this section.

3.1. Protocol Description

We assume that the network consists of subnets, each having a dedicated router as its *multicast router*. The multicast router of the video server is referred to as the *root router* in this paper. To facilitate our protocol, routers are equipped with local storage for caching purposes. This storage is organized into chunks of equal size. Each chunk is used to cache the last chunk-size of data for a video stream currently passing through the router. Furthermore, routers must be built as active nodes which can perform computations on packet data that travel through them. This is made possible by the recent trend of active network technology.[18]

When a node (client station) X requests some movie V, it sends a request message to its multicast router. This router broadcasts the request message toward the root router. A time to live is used to prevent the message from traveling too far. Upon receiving the message, each router along the way checks its internal information to see if V is being cached in any chunk of its cache. If this condition is not satisfied, it just forwards the message. Otherwise, a router, say R, is capable of serving node X if R still has the first frame of the video. In this case, R modifies the packet to request the right to serve, and unicasts the message to the root router. Suppose that before R receives the answer from the root router, another request for the same video arrives from node Y. To handle this new request, informing the root router is not necessary. R needs only add this new request to its *group list* of nodes waiting for the video V.

As soon as the root router receives the first request to serve X, say from R, it sends a message to inform R of its right to serve along with the group address. In this case, R is referred to as the *serving router*. All subsequent requests to serve are denied; and the requesting nodes are notified accordingly. The serving router R, having received the group address from the root router is ready to build the delivery tree for the group. This tree from R to the

receivers (nodes in the group list) is the reversal of the paths from the receivers to the router on which request messages were sent. Over this tree, R notifies the receivers of the group address. In response to this group-address message, all the intermediate routers along the way update their multicast tables and prepared to cache the upcoming video stream. We will discuss the cache management policy shortly. In the case that no router has the data to serve node X, the service request will eventually reach the video server. Under this circumstance, the server must initiate a new stream to provide the service. The multicast path is the reversal of the path on which the request message is sent from node X to the video server. On the way the group address is sent back to X, each intermediate router responding by getting prepared to cache the upcoming video stream as in the previous case.

There might be a situation in which a node belongs to more than one multicast group. In this case, the node simply picks the first one. It sends back an acknowledge to the corresponding serving router, and gives up the other groups. In order to quit a group, a node needs to send a request to its multicast router. If the multicast router R of a subnet S detects that a group G no longer has members in S, R looks up its multicast table to determine the incoming router for group G and informs this router to drop R from its multicast table.

To illustrate the way CMP works, let us look at the example given in Fig. 2(a). We assume that each router has only one chunk of caching space. Each chunk can cache up to ten minutes of video data. All videos are assumed to be longer than ten minutes. VS denotes the video server. The label on each link indicates the starting time of the video service. For instance, the server starts a video stream at time zero, and deliver it to client C_1 over the routers (R_1, R_2, R_3, R_4). For simplicity, we assume that requests can be serviced instantaneously. Therefore, the request from C_1 was also initiated at time zero. Fig. 2(a) illustrates the following scenario. At time 0, a node C_1 requests some video V. Since no router has the data, the server has to allocate a new stream to serve C_1. As the data goes toward C_1, all the routers along the way, R_1, R_2, R_3, R_4, cache the data in one chunk of their local buffer. At time 7, node C_2 requests the same video. Since R_2 has not dropped the first video frame from its cache, it can serve C_2. R_2 is referred to as the *serving router*. As a result, all the routers along the path from R_2 to C_2 (i.e., R_5, R_6, R_7) are asked to cache the video if possible. Later at time 8, node C_3 also requests video V. C_3 can get the service from router R_3 since R_3 still has the first video frame in its cache. By time 10, R_1, R_2, R_3, and R_4 have replaced the first video frame in their cache. At time 11, C_4 asks for V. It can receive the service from router R_5 which still has the first video frame.

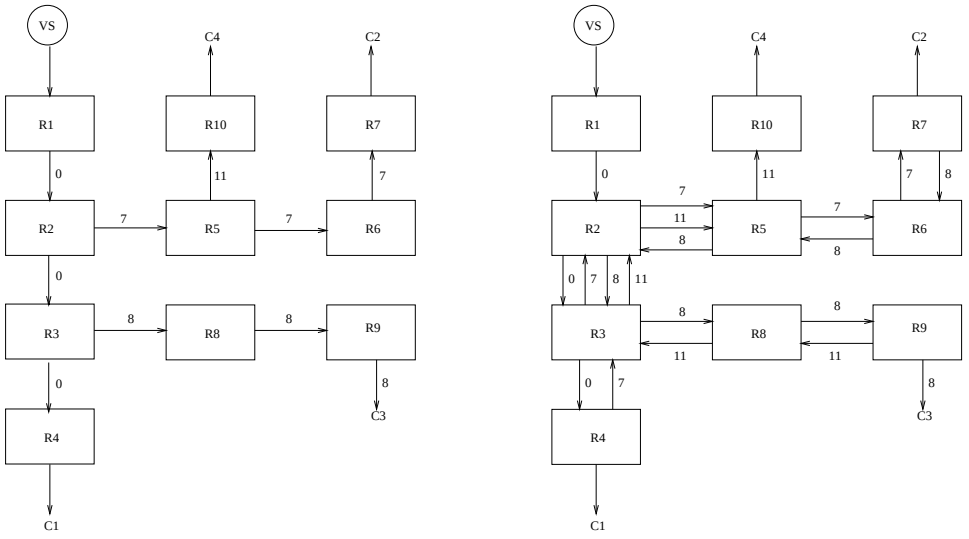

Figure 2. Examples of how CMP and Chaining work

To see the advantage of CMP over Chaining, let us refer to Fig. 2(b), where the situation and configuration are the same. However, Chaining leads to a totally different result. At time 0, C_1 requests video V. The video server allocates a stream to deliver V to C_1. As C_1 plays back the video, it caches the data in its local buffer. Node C_2 requests the same video at time 7. According to Chaining, C_2 can receive the service from C_1. A stream

(through $R_4, R_3, R_2, R_5, R_6, R_7$) is reserved for this session. Similarly, C_2 also caches the data in its local buffer. At time 8, C_3 requests also video V. A communication channel is set up to chain C_3 to C_2 through routers $R_7, R_6, R_5, R_2, R_3, R_8, R_9$. As C_3 plays back the data arriving from C_2, C_3 caches the data in its local buffer. At time 11, the request from C_4 arrives also for the same video. Similarly, a channel is set up to pipeline the data from C_3 to C_4 through $R_9, R_8, R_3, R_2, R_5, R_{10}$.

Comparing the two examples, we can make the following observations: (1) although both schemes requires only one data stream from the server, the network cost is about two times higher for Chaining. (2) he average service delay is shorter under CMP. As an example, C_4 can receive the service from R_5 under CMP, instead of from C_3 under Chaining, which is much farther away. We note that the CMP design is not distance-based as some other routing protocols. As a result, there is not much processing to determine the delivery trees.

3.2. Data structures and algorithms

In what follows, we present how CMP can be implemented in terms of packet structure, router data structures and routing algorithms. We need to categorize CMP packets into several types:

REQ: This packet is sent by a client station to its multicast router to request a service.

FIND: It is sent by a router after it receives a REQ message initiated by some client.

FOUND: If a router receives a FIND message and is able to serve the request, it sends a FOUND message to the root router to request the right to serve.

REP: The root router replies with a REP message to each router that sents a FOUND to the root router.

CACHE: When a router gets this message, it needs to be ready to cache the upcoming video stream.

ACK: An acknowledge sent by a client's multicast router to a router that is willing to serve the client (i.e., the serving router).

DAT: Video data packets.

QUIT: A node sends this packet to withdraw its participation from a multicast.

For each of these types, a corresponding routine is installed at each router in order to route the packets correctly. A dispatcher is also needed to listen to incoming packets. Based on their type, the dispatcher passes them to the responsible routine for further processing. Routers like this are active since they can perform computations on the packets and significantly become involved in the routing progress.

To support the service routines, we need to maintain the following information at each router:

Cache Directory: It maintains information about which video is being cached in which chunk of the local buffer. We use FIFO as the replacement policy within each chunk. That is, it caches the last chunk-size of data for the stream currently using the chunk. If a chunk is not caching data for a serving router, it is marked as free after dropping the first video frame. Any of these free chunks can be assigned to the next video stream. This cache management policy utilizes the buffer space very efficiently due to the following observations: (1) A chunk, at any one time, caches only a small portion of a video. (2) If a chunk is not caching data for a serving router, it is released very quickly.

Multicast Table: It stores the IDs of the incoming and outgoing routers for each multicast group.

Log Table: It stores information about the REQ packets traveling through the router. This is used to avoid some packets coming back to the router. This is also used to build the multicast trees.

Group Lists: While a router is waiting for permission to serve a client, more requests for the same video are added to a group list associated with that service. If the router is selected to provide the service, this group list and the log table are used to build the multicast tree. On the other hand, if the router is rejected (either by the root router or by the client), the group list is deleted.

To illustrate some of the packet types and data structures discussed above, we provide the service routines for the root router in Fig. 3. We note that a root router only deals with three types of packets: REQ, FIND and FOUND.

- Packet type = REQ:
  - *Check* and *update* the log table
  - *Serve* the client
- Packet type = FIND
  - *Check* and *update* the log table
  - *Set* a timer to wait until a FOUND message comes
  - If (the timer expires and no FOUND message comes)
    - *Update* the multicast table
    - *Send* a CACHE message to the client
  - Else
    - If (this is the 1st FOUND message for the same client ID)
      - *Update* the multicast table
      - *Send* a REP to the sending router to ask it to serve the client
    - Else
      - *Send* a REP to the sending router to ask it not to serve the client
- Packet type = FOUND
  - *Check* and *update* the log table
  - If (this is the 1st FOUND message for the same client ID)
    - *Update* the multicast table
    - *Send* a REP to the sending router to ask it to serve the client
  - Else
    - *Send* a REP to the sending router to ask it not to serve the client

Figure 3. Routines at the root router

3.3. Application

To demonstrate the usefulness of CMP, we discuss in this section how an existing video-on-demand multicast technique can leverage this protocol. We chose *Patching*[11–13] for this discussion.

One disadvantage of standard multicast techniques is that once a server channel is allocated to deliver a video, it is not released until the end of the video. As a result, the demand on the server bandwidth is significant. Patching can avoid this. The idea is to allow a new client to join an existing multicast by caching the multicast data in its local disk. The server needs only transmit the leading portion of the video in a new *patching stream*. When the playback of the leading portion is completed, the client continues to play back the remainder of the video using the data already buffered in the local disk. Clearly, since the video server does not always have to use a channel for the full duration of a video, the demand on the server bandwidth is much less under Patching. In,[11] *Patching* is shown to offer substantially better performance than standard multicast techniques.

By using CMP, *Patching* can be improved. For convenience, we call the new patching scheme *CMP-Patching*. Let $|v|$ represent the video length. Suppose that at time 0, a regular stream S_r is launched. At time t, a patching stream S_a is initiated for client A. Soon afterward at time $t+h$ (h is very small), a new client B arrives, and another patching stream S_b is allocated according to Patching. If we use $v[t_1, t_2]$ to denote the segment of the video from time t_1 to time t_2, then the patch required by A is $v[0, t]$, and that required by B is $v[0, t+h]$. In total, $2t+h$ time

units of video data are delivered on patching streams for the two clients. *CMP-Patching* can reduce this amount of data. Since both the streams S_r and S_a can be cached in the routers under CMP, B can receive $v[0,t]$ from a router on the path from the server to A. Consequently, the server needs to send only h amount of patching data to B. In total, the server sends only $t+h$ amount of patching data under CMP, which is about half the amount under regular Patching.

4. PERFORMANCE EVALUATION

To evaluate the efficiency of CMP, we implemented two detailed simulators to compare it to (Extended) Chaining running on the Multicast Open Shortest Path First (MOSPF) routing protocol.[16] Our performance metrics are average latency and system throughput. We explain them below:

- *Average latency*: The average waiting time it takes for a request to get service.
- *System throughput*: The mean number of requests served in a time unit.

Routers used in our simulation have a default bandwidth of 100 concurrent streams per communication port. That is, a link between any two nodes can support up to 100 video streams at any one time. Our network is assumed to consist of a root router and 1,024 multicast routers, each being dedicated to a distinct subnet. The multicast routers are interconnected to form a 10-dimensional hypercube. The root router is connected to this hypercube through links to routers located on its different dimensions. This scheme gives the same effect of placing a set of distributed servers at various strategic locations thoughout the network. For convenience, we refer to the number of links between the root router and the hypercube as the *server bandwidth* in this paper. The default value is 20 links. That is, the root router has two links connected to each of the ten dimensions of the hypercube.

Our workload consists of 100,000 service requests, each having the form *(client id, video id, router id, arrival time)*. We explain these parameters as follows:

- Our workload generator creates the service requests according to a Poisson process with the default request rate $\lambda = 0.7$. Each client is willing to wait for at most five minutes. When this time expires, the client reneges its request if resources are still not available for the service.
- The workload generator increments the client id for each new service request. In other words, each client is assumed to make only one request during the entire simulation run. Without loss of generality, this assumption simplifies the implementation of our simulators. For each new request (or client), the workload generator randomly selects the multicast rounter. Thus, clients are randomly assigned to the subnets. This strategy allows us to effectively control the sizes of the subnets by adjusting the request rate.
- To model the access pattern, videos are chosen according to a Zipf-like distribution with skew factor z. That is, a video i $(1...N)$ is selected by any request with a probability of $\frac{\frac{1}{i^z}}{\sum_{j=1}^{N}\frac{1}{j^z}}$, where N is the total number of videos. A larger skew factor represents a more skewed distribution, in which some videos are requested much more often than the others. In our experiments, we adjust this parameter to control the desired access patterns. Its default value is set at 0.7 which is typical for VOD applications.

Under CMP, each router is equipped with a small cache, whose default size is one minute of video data. We note that such a small buffer can be implemented in semiconductor memory. In the case of Chaining, each client can cache up to five minutes of video. For convenience, we use one minute of video data as the unit for caching in this paper. We summarize our system and workload parameters in Table 1. The rightmost column shows the ranges selected for our sensitivity analyses. For instance, the average request rate was varied between 0.1 to 1.0 to observe its effect on the performance metrics (i.e., mean service delay and system throughput).

Table 1. Parameters

Parameter	default	variation
Number of requests	100,000	N/A
Number of videos	100	N/A
Video length(minutes)	90	N/A
Client buffer size (minutes)	5	1-5
Router chunk size (minutes)	1	1-5
Number of memory chunks	1	N/A
Request rate (request/second)	0.7	0.1-1.0
Network bandwidth (streams/port)	100	60-140
Server bandwidth (comm. links)	20	12-28
Skew factor	0.7	0.1-1.0

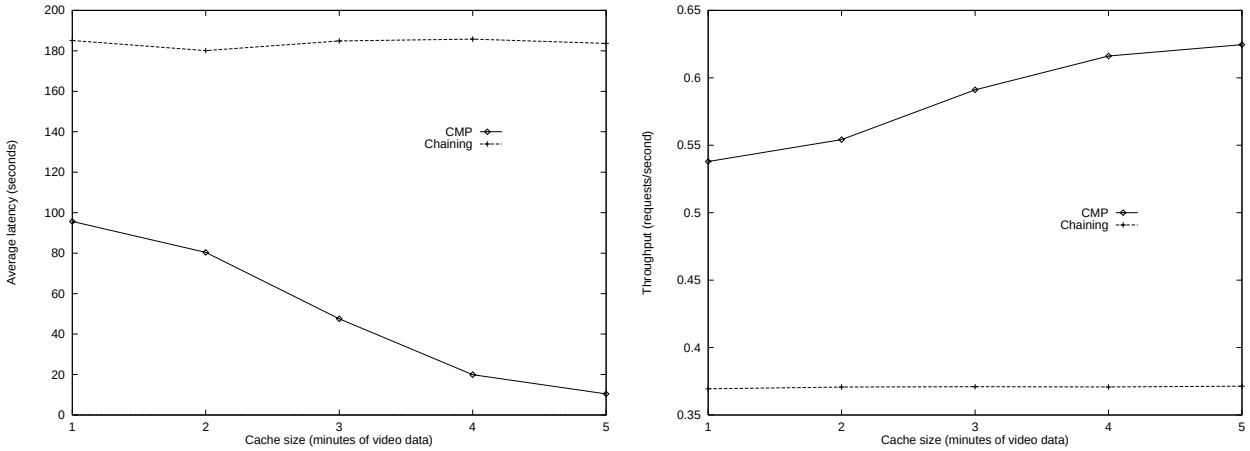

Figure 4. Effect of the cache size.

4.1. Effect of cache size

In this study, we set the router cache size under CMP to be the same as the client cache size under Chaining. We varied this cache size from 1 to 5 units over five simulation runs. The performance results are plotted in Fig. 4. We observe that CMP outperforms Chaining by a very wide margin. As an example, when the cache size is 3 units, CMP is 400% better in terms of latency, and 60% better in terms of system throughput. As the cache size increases, CMP exhibits better performance. In contrast, Chaining does not seem to be able to benefit from the additional caching space. This is due to the fact that Chaining has much higher network costs. As a result, it does not have enough network bandwidth to handle the relatively high request rate selected for this study. In summary, Chaining is much more demanding on the network bandwidth compared to CMP.

4.2. Effect of request rate

The effect of request rate on the two schemes is shown in Fig. 5. We note that since we fixed the number of requests for each simulation run, the simulated time decreases as the request rate increases. Again, CMP is significantly better in this study. The improvement in throughput is as high as 100% when the request rate is 1 request per second. This performance gap would have been larger if we had continued to increase the request rate. This outstanding performance is achieved while maintaining good service latency. The plot indicates that CMP provides at least 50% better service latency. For applications requiring true VOD (i.e., no delay), we observe that CMP and Chaining can support 0.3 and 0.2 requests per second, respectively. The improvement in this case is 33%. We again observe here that Chaining cannot handle very high request rates due to its high demand on network bandwidth. The finite network bandwidth (i.e., 100 streams/port) limited the performance of Chaining to about 0.37 requests per

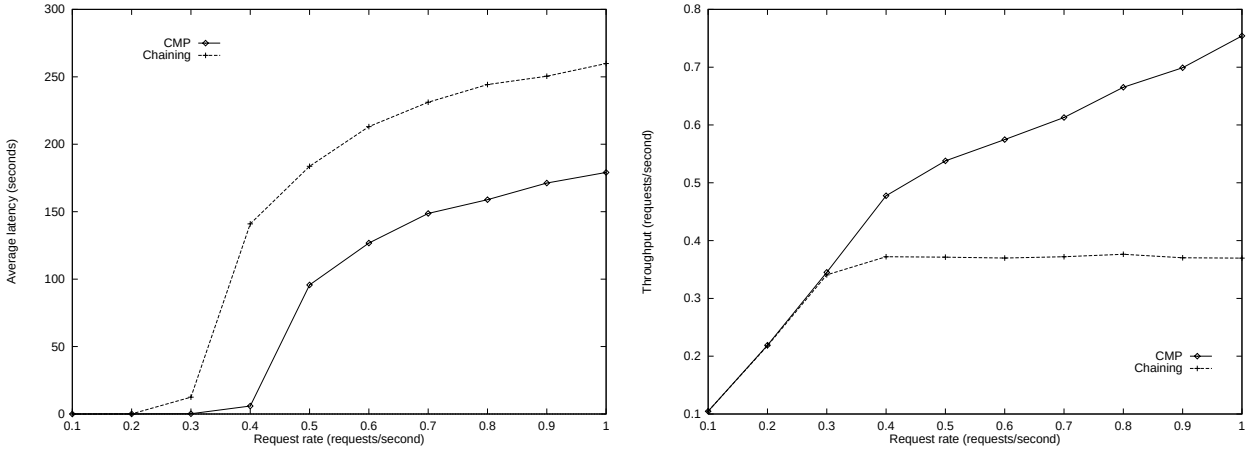

Figure 5. Effect of request rate.

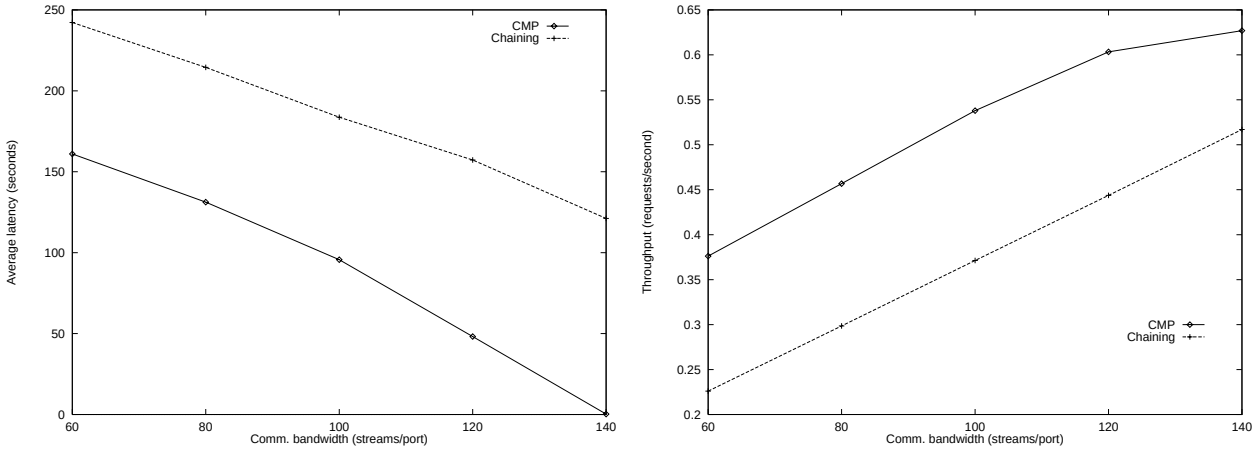

Figure 6. Effect of network bandwidth.

second. Increasing the request rate beyond this point only lengthens the service delay without further gain in system throughput. In contrast, we observe that CMP can scale well beyond one request per second.

4.3. Effect of network bandwidth

In this study, we varied the network bandwidth from 60 to 140 concurrent streams per port. The performance results are plotted in Fig. 6. It shows that CMP substantially reduces the service latency. As an example, a reduction of 70% can be achieved when the network bandwidth is 120 streams/port. Similar behavior is observed for throughput, except the curve for CMP starts to flat out at 120 streams/port. This should be interpreted as positive. It confirms the fact that CMP does not need that much network bandwidth unless the request rate is higher.

4.4. Effect of server bandwidth

We can increase the server bandwidth by adding more links between the root router and the hypercube network. In this study, we varied the server bandwidth from 12 to 28 links, each giving an effective bandwidth of 100 concurrent streams. These links are evenly distributed among different dimensions of the hypercube. The results of this study are plotted in Fig. 7. They show that Chaining is much more demanding on the server bandwidth. As an example, in order to achieve a throughput of 25 services per minute, Chaining needs a server bandwidth 400% more than that required by CMP. This is quite impressive considering the fact that CMP uses 488 ($\approx \frac{100{,}000 \times 5}{1025}$) times less caching space. We note that the two throughput curves will eventually converge if we continue to add server bandwidth. This is due to the fact that CMP does not have enough workload to fully utilize the available bandwidth. In other words, CMP does not need a lot of server bandwidth unless the request rate is higher.

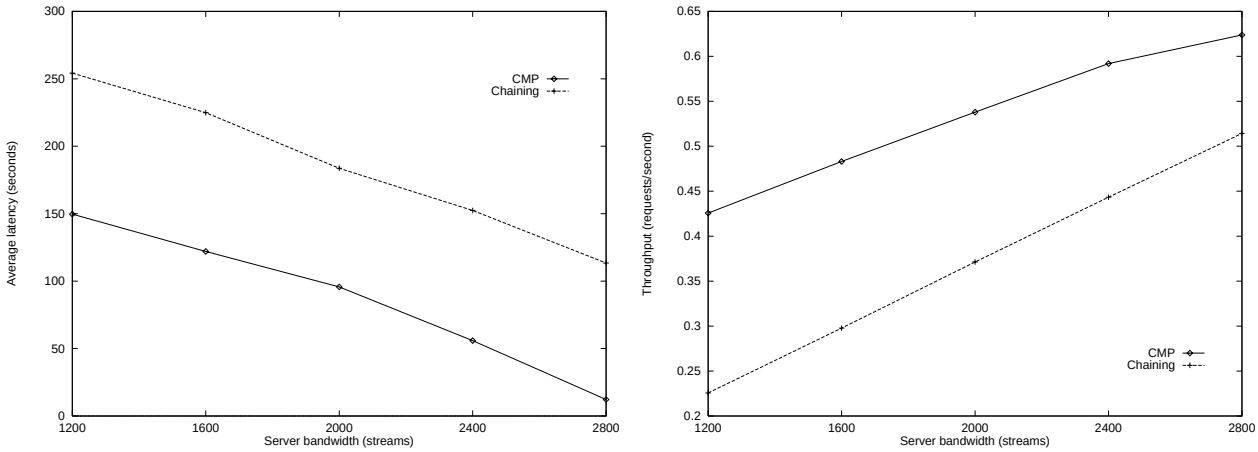

Figure 7. Effect of server bandwidth.

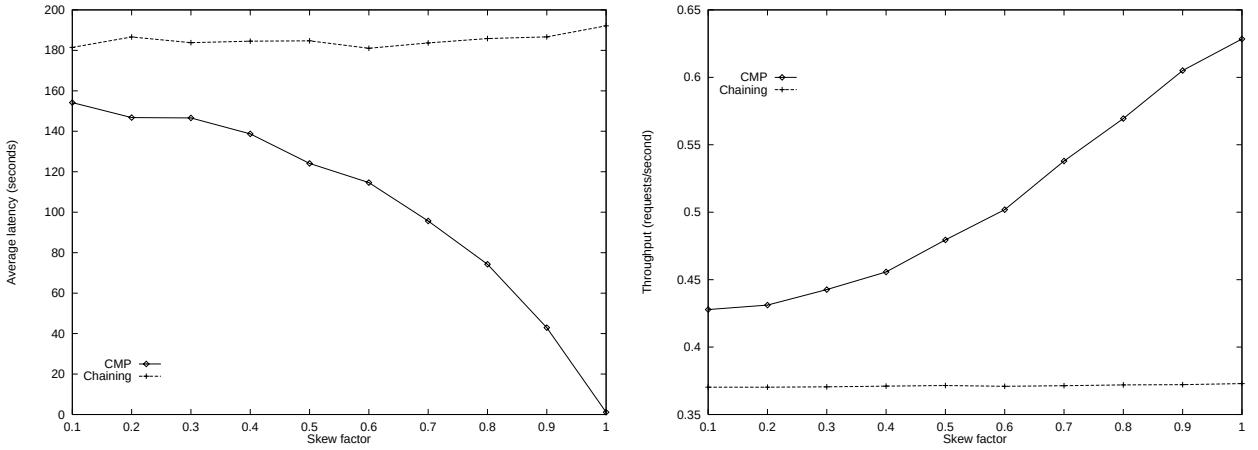

Figure 8. Effect of video access pattern.

4.5. Effect of video access pattern

The effect of video access pattern is plotted in Fig. 8. We observe that the performance of Chaining is essentially unaffected by the skew condition. On the contrary, the performance of CMP improves noticeably under a more skewed video access distribution. As an example, if more videos are selected more frequently, the average latency in CMP is close to zero and the throughput is about 100% better than that of Chaining. This is because the delivery trees generated by CMP are much larger due to delay features, cache availability, and very efficient routing and caching policies.

5. CONCLUDING REMARKS

In this paper, we surveyed different approaches for scheduling multicasts in a distributed video-on-demand system. We contend that such techniques do not offer the best performance. To further improve the performance of video multicasts, we looked deeper into the network layer and developed a new protocol called *Caching Multicast Protocol* (CMP). Unlike many existing protocols which were designed to support general-purpose applications, CMP is tuned to take advantage of the peculiar characteristics of video and audio data. In particular, it uses local storage in routers to cache network data and multicasts them to future requests on demand. Since these multicasts are done on demand, true video on demand can be achieved. Making data available at various routers also reduces the demand on the server bandwidth and allows the application to leverage the in-network bandwidth.

CMP and Chaining are fundamentally different. While Chaining operates at the application level, CMP is a network-layer protocol. As a result, a multicast tree truly expands over time under CMP. On the contrary, Chaining

must introduce new multicast groups in order to admit new members. This fundamental difference makes CMP less expensive in terms of network costs. It uses substantially fewer network links for delivering data. Another important difference between these two techniques is that CMP does not require client nodes to forward data. This characteristic makes CMP more practical for a wider range of applications.

To assess the performance of the new technique, we carried out simulation studies to compare CMP with Chaining running on the *Multicast Open Shortest Path First* (MOSPF) routing protocol.[16] The results indicate that CMP is much less demanding on network and server bandwidth. It also requires substantially less caching space. With caching space 488 times less than that used in Chaining, CMP still wins by a very significant margin.

The design of CMP is motivated by the current trend of designing routers to support more complex forwarding logic. Since CMP requires the routers to perform computation on packet data, it performs best if active routers are available. For the current IP, *Active Node Transport System* (ANTS)[19] can be used. ANTS is a reference active network implementation, in which if a node that a packet passes through contains the related code, the node initializes the code with the packet's parameter values and then executes the code. Since the default code performs only conventional IP forwarding logic, ANTS can be used to integrate active networks into IP. CMP can be implemented on IP using this model.

REFERENCES

1. T. Maufer, *Deploying IP Multicast In the Enterprise*, Prentice-Hall, Inc., 1998.
2. S. Paul, *Multicasting on the Internet and Its Application*, Kluwer Academic Publishers, 1998.
3. S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia systems* **4**, pp. 179–208, August 1996.
4. C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *Proc. of the IEEE Int'l Conf. on Multimedia Systems'96*, (Hiroshima, Japan), June 1996.
5. K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proc. of the ACM SIGCOMM'97*, (Cannes, France), Sepetember 1997.
6. J. F. Paris, S. W. Carter, and D. D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proc. of SPIE's Conf. on Multimedia Computing and Networking (MMCN'99)*, pp. 317–326, (San Jose, CA, USA), January 1999.
7. C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proc. of the IEEE Int'l Conf. on Multimedia Systems'96*, (Hiroshima, Japan), June 1996.
8. A. Dan and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *ACM Multimedia '98*, (San Francisco), October 1998.
9. J. Oh, K. A. Hua, and K. Vu, "An adaptive hybrid technique for video multicast," in *The IEEE 7th International Conference on Computer Communications and Networks*, (Louisiana, USA), October 1998.
10. S. Sheu, K. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video on demand," in *Proc. of the Int'l conference on Multimedia Computing and System*, (Ontario, Canada), June 1997.
11. K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. of ACM Multimedia*, pp. 191–200, (Bristol, U.K.), September 1998.
12. Y.Cai, K. Hua, and K. Vu, "Optimizing patching performance," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1999.
13. S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal patching schemes for efficient multimedia streaming," tech. rep., UMASS CMPSCI Technical Report 99-22, 1999.
14. A. Ballardie, "Core based trees (cbt version 2) multicasting routing - protocol specification," in *RFC-2189*, September 1997.
15. S. Deering, D. Estrin, and D. Farinacci, "Protocol independent multicast version 2, dense mode specification," *draft-ietf-idmr-pim-dm-spec-05.ps* , May 1997.
16. J. Moy, "Multicast routing extensions for ospf," *Communications of the ACM* **37**, August 1994.
17. T. Pusateri, "Distance vector multicast routing protocol," *Internet draft: draft-ietf-idmr-dvmrp-v3-05.txt* , October 1997.
18. D. Tennehouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Communication Magazine* , January 1997.
19. D. Wetherall, J. Guttag, and D. Tennehouse, "Ants: A toolkit for building and dynamically deploying network protocols," in *Proc. OpenArch 98*, IEEE CS Press, (Los Alamitos, CA), April 1998.