Layered Range Multicast for Video On Demand

Duc A. Tran Kien A. Hua Tai T. Do
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816, USA
Email: {dtran,kienhua,tdo}@cs.ucf.edu

***Abstract*— We focus on the problem of providing quality-of-service guarantee, scalability, and on-demand property to video streaming systems. We propose a solution called Layered Range Multicast (LRM). LRM allows transmitting a range of data to a multicast group's members, which helps clients who request for service at different times join a same multicast efficiently without additional server bandwidth allocation. This is more advanced than the conventional multicast in which a late client joining an existing multicast, without bandwidth support from the server, must miss a certain portion of the video requested. Another advantage of LRM is its capability to support clients requesting various levels of service quality. In addition, LRM does not assume the existence of IP Multicast and therefore it can be implemented on the current Internet without degrading to the very inefficient centralized approach. Our performance study results confirm the above benefits.**

I. Introduction

Providing video-on-demand (VOD) services to clients having different resource constraints is challenging since no single transmission rate could fully satisfy all those clients. Heterogeneity supporting solutions for multimedia communications have been proposed [11], [19], [2], [12], however, they do not scale well with the number of client requests. For instance, requests arriving at different times are not easily coalesced into a batch so that the server can serve them together using a multicast address. They are most likely to be served using separate video streams, which dramatically exhausts the server bandwidth. On the other hand, a VOD system could reduce the demand on the server bandwidth by employing techniques such as client caching [8], [15], [14], proxy caching [18], [20], [17], network caching [9], or server batching [1], [5]. However, it is not clear how these techniques are extended to address the heterogeneity in services requested by the clients.

Our target in this paper is to develop a unified communication paradigm designed to provide both QoS guarantees and scalability to VOD systems. Specifically, we propose a scheme called Layered Range Multicast (LRM). This name alludes to two important points of our approach:

- Unlike the traditional video multicast, a "range multicast" transmits a range of continuous frames to the participating clients at any one time, each frame for a particular client. Consequently, as long as the requested play point is in the range, new clients can join a multicast group at their own time and still receive the entire video without additional server bandwidth. This feature offers at least two benefits: (1) The service latency is shortened since clients may join an existing multicast late instead of waiting for the next available server stream; (2) The demand on the server bandwidth is alleviated because a multicast can expand dynamically to accommodate new service requests avoiding the need for new server streams.
- We propose to solve the heterogeneity problem by using a layered video coding approach, as in [12], [11], [19], however with a novel layered transmission and caching scheme. Layered multicast provides a finer granularity of control compared to using a single video stream since a client is able to subscribe to one or more layers depending on its capability.

LRM is realized by deploying an overlay of caching nodes on the Internet. These nodes play as application-level routers to implement the multicast paradigm. The idea of LRM is as follows. We start with a simple case where a client requests a video having some level of QoS and receives it in a video stream from the server through a number of caching nodes. This stream consists of a base layer and one or more enhancement layers, which corresponds to the requested QoS. As these layers are transmitted toward the client, intermediate nodes cache them into a fixed-size sliding buffer. Suppose at a time later another client requests the same video but possibly with a different QoS requirement. This new client looks for nodes currently holding in their cache the *first frames* of constituent layers of the requested QoS. If successful, the new client will join those nodes to get the layers of data needed. As sent to the new client from each of the joined nodes, the data are cached into intermediate nodes on the delivery path similarly to the simple case above. These cached data will be used later for providing necessary layers to subsequent clients. The rationale for using a caching buffer at each node LRM is to prolong the usefulness of a multicast, in other words, to increase the range of data available on that multicast. The larger this range, the more successful new clients join the multicast. We note that in IP multicast, the "range" of data transferred on any multicast is a single packet. Obviously, this conventional solution is limited in supporting requests arriving at different times.

Besides having the advantages of better reducing server bandwidth and accommodating client heterogeneity, LRM provides a feasible and efficient way of implementing the multicast paradigm on the Internet. Indeed, most existing VOD or layered multicast techniques assume the existence of IP Multicast [6]. However, IP Multicast deployment on the Internet has been slow and even today remains severely limited in scope and reach. LRM provides an efficient unicast-based multicast implementation by employing overlay nodes as the application-

This research is partially supported by the US National Science Foundation grant ANI-0088026.

level routers.

In the following section, we describe the system model assumed by LRM. Next in Section III, we provide LRM protocol in detail. We then explore its performance through simulation in Section IV. Finally, in Section V, we give concluding remarks and pointers to our future work.

II. System Framework

In this section, we present the network and QoS models used in the Layered Range Multicast (LRM) approach.

A. Network Model

In the LRM environment, LRM-enabled nodes are placed across the Internet, such as where distributed content servers are located, and interconnected using unicast paths to implement the range-multicast layer. For ease of exposition, we refer to the LRM-enabled nodes simply as "nodes" unless otherwise specified. They are classified into two types: *root* and *non-root*. The root node is the front-end node for the video server to communicate with the rest of the network. A non-root node may or may not be representative for a local community of clients. When representing a community including a client X, a non-root node is called the *representative* of X.

In the absence of widely available IP multicast services on the Internet, using the overlay architecture is advantageous. Indeed, the multicast paradigm can be efficiently implemented on the Internet, based on IP Unicast only, to reduce the server bottleneck. In this architecture, overlay nodes play as application-level routers. On the other hand, the deployment of overlay nodes incurs the cost of maintaining the nodes and overcoming their failures. Sharing the same insight in [10], we believe that one-time hardware costs do not drive the total cost of the system. Maintenance costs are cut by simplifying node deployment. Overlay failures have two kinds of impact: on the overlay topology and on the on-going services. Several efficient solutions [7], [13], [10], [3] have been proposed for building fault-tolerant overlays whose topology is re-configurable on failures. Such a solution can be appropriately adopted to maintain the Range Multicast overlay. For on-going services that may be interrupted due to a node failure, all the adjacent nodes that have been receiving data from this failed node are temporarily reconnected to the root node to receive the remaining data. Even though possibly sub-optimal, this solution gives a simple and quick way to guarantee the continuity of the client playback.

The above issues are important and their solutions can be employed in the LRM architecture. However, due to the lack of space, we skip discussing them in this paper, thus for clarification, we assume that the LRM overlay has a fixed and failure-free topology.

B. QoS Model

We assume that each video stream V is hierarchically encoded into L *layers*: a basic layer v^1 and $L-1$ enhancement layers v^2, v^3, .., v^L. V is decoded using v^1 and probably a number of enhancement layers based on the quality required. For example, decoding only the basic layer provides the minimum quality of the video, while decoding all the layers provides the best quality. Therefore the video V has L *levels* of quality; level j (denoted by $V^j = \{v^1, v^2, .., v^j\}$) needs a decoding of the first j layers.

At the server side, a video is transmitted in a stream of packets. Each packet includes a header field specifying what layer it belongs to. The sequence of packets belonging to a layer j is called a sub-stream corresponding to layer j. A client requests a video having a certain level of QoS which best matches the client's constrained resource. This level may also depends on the service pricing policy. As an example, a client might want to watch a video in black and white colors because its budget is not affordable for a higher quality, while a richer client might want to see that video of the best quality. For simplicity, in its request a client specifies V^j - the quality level j of the video V it wants to have. If the service does not satisfy the required QoS it is the client's responsibility to explicitly make another request (for a lower quality) or to quit the system. This QoS model for client requests is similar to that employed in [16].

III. Layered Range Multicast

In this section, we present first the design of the overlay nodes, and then the service procedure for client requests.

A. Overlay Cache Design

LRM does not request every node to cache. For instance, the root node does not need to cache. For those non-root nodes caching, an amount of the local storage is reserved for caching purposes. The caching space per node is organized as an array of equally sized *chunks* C_1, C_2, .., C_N. Such a chunk C_k is further divided into a number of *sub-chunks* C_k^1, C_k^2, .., C_k^L, each C_k^l associated with a layer l and used to cache data from a stream corresponding to that layer. For instance, if a node decides to cache a video stream V^j (i.e., video V of quality level j), the sub-stream corresponding to layer l ($l \in \{1, 2, .., j\}$) will be cached into sub-chunk C_k^l for some $k \leq N$. If $l > L$, this sub-stream will not be cached.

The reason of having a two-level (chunk/sub-schunk) cache organization is to increase each node's throughput, hence the number of chunks N in a node is dependent on how many streams that node can support simultaneously. Once this parameter has been determined, the available caching space is equally divided among the chunks. The number of sub-chunks L in a chunk must be no more than the maximum number of video layers. The size of a sub-chunk does not affect the correctness of the protocol, however a proper sizing results in a high utilization of the node's cache. We propose to size a sub-chunk corresponding to a layer proportionally to the bandwidth requirement for that layer. This is an intuitive way, but by no means the optimal, to choose the sub-chunk size. Our rationale is that if the first frame of a layer of a video stream exists in a sub-chunk, by using the proposed sub-chunk sizing, it is most likely that the first frames of the other layers also exist in the corresponding sub-chunks. Effectively, this helps increase the number of full layers that a node can provide to new clients. Without loss of generality, we denote the size of a sub-chunk corresponding to layer l by $size(l)$.

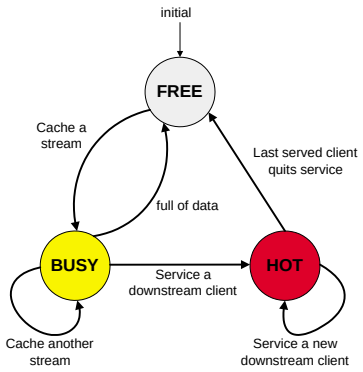

Fig. 1. State Transition in A Sub-chunk

The algorithm to cache a stream into sub-chunks is as follows. When a new stream V^j arrives at a node R at time t_0, for each sub-stream corresponding to layer l of V^j ($l \in \{1, 2, .., j\}$), R finds an empty sub-chunk corresponding to layer l for caching v^l. If there is no such empty sub-chunk, R selects the sub-chunk corresponding to layer l that has not been used for servicing a downstream client for the longest time (we will explain how a sub-chunk is used for serving a downstream shortly), and allocates that sub-chunk for sub-stream v^l. If such a sub-chunk does not exist either, the sub-stream v^l will not be cached. Supposing that node R successfully finds out a sub-chunk C_k^l, caching v^l into it follows the Interval Caching policy [4], [9]. Using interval caching is advantageous to subsequent client requests. Indeed, the data cached in sub-chunk C_k^l can provide the entire sub-stream corresponding to layer l to all clients who arrive before time $t_0 + size(l)$ requesting the same video of at least level l. C_k^l would be used as a sliding window to hold and forward the layer-l data to those new clients. If this happens, the sub-chunk C_k^l is called "servicing a downstream client" that we mention earlier. Furthermore, if a sub-chunk C_k^l is currently caching a sub-stream v^l but not full yet C_k^l is said "capable" of providing layer l of video V to any client who needs that layer. For the rest of the paper, we say that a sub-chunk C_k^l is in "free" state if it is empty, in "busy" state if it is caching but neither full nor servicing any downstream client, in "hot" state if it is servicing at least a downstream client. A sub-chunk in hot state cannot be used for caching any new stream until it becomes busy or empty. The transition between these states is illustrated in Figure 1.

At time $t_0 + size(l)$, sub-chunk C_k^l is full. If currently not in the hot state, C_k^l is cleaned up and returned to the empty state. Otherwise, it continues caching as usual and old data will be replaced with newly arriving data in the FIFO fashion. An example is given in Figure 2 where the node has three chunks, each having four sub-chunks corresponding to four video layers. U, V, and T are existing streams and already occupy some sub-chunks. Only three sub-streams of the new stream will be cached but the one corresponding to layer 2 will not since all sub-chunks of layer 2 are occupied and in the hot state.

B. Service Procedure

Servicing a request includes three phases: (1) Seeking phase, (2) Data transmission phase, and (3) Leaving phase. First, we describe each phase, and then provide an example to illustrate how LRM works.

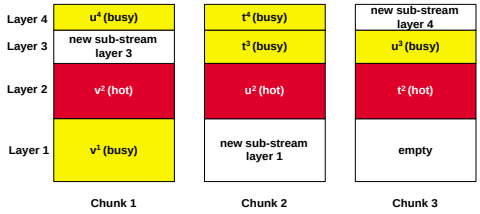

Fig. 2. A Node's Cache

1) Seeking Phase: A client node X requests a video V of quality level J (i.e., V^J) to its representative $Rep(X)$. This node is responsible for finding those nodes that collectively are capable of providing layers $\{v^1, v^2, .., v^J\}$. For this purpose, $Rep(X)$ "broadcasts" the request in a **find**(X, V, J) packet to the overlay nodes. By broadcast, we mean to apply on the overlay layer only. In other words, a node forwards the packet to all adjacent-on-overlay nodes on the corresponding unicast paths. A duplicated packet arriving at a node is ignored. A packet reaching the root node is not forwarded. Since the number of overlay nodes is not large, the network load incurred by this broadcast should not affect the network traffic severely.

Upon the first arrival of the **find** packet, the root sends back to $Rep(X)$ a **found** message whereas each receiving non-root node R follows the steps below:

- Compute the list $L = \{v^{l_1}, v^{l_2}, .., v^{l_j}\} \subset V^J$ such that each layer $v^{l_k} \in L$ still has a prefix in the cache.
- If L is empty: R forwards the **find** to its adjacent nodes on the overlay except for the origin node.
- If L is not empty: R stops forwarding and sends a **found** message including list L on a direct unicast to $Rep(X)$ to inform that client X can download the layers in this list from node R.

As a result of the above process, the representative $Rep(X)$ may progressively receive a number of **found** messages: (R_1, L_1), (R_2, L_2), .., (R_n, L_n) where (R_i, L_i) denotes the **found** message from node R_i which can provide layers in the list L_i to the client. The root is always capable to send every layer to any client. This list of **found** augments as more nodes reply. We employ a queue Q_x to store the **found** messages received by $Rep(X)$. Initially, Q_x is empty. As soon as receiving at least a **found**, $Rep(X)$ follows the selection steps below to receive all the layers of V^J:

1) Set L = EMPTY, $i = 0$
2) Put all the current **found** messages into Q_x
3) WHILE ($L \neq V^J$)
   a) WHILE (Q_x = EMPTY) Waiting
   b) Dequeue (R_i, L_i) from Q_x
   c) Send node R_i an **ack** message asking it to send the layers specified in $L_i \setminus L$ to the client
   d) $L = L \cup L_i$, $i = i + 1$
   e) If there are more coming **found** messages to $Rep(X)$, then put them into Q_x
4) Send each of the rest of nodes in Q_x a **nack** message to deny its offer.

After the selection steps are finished, any more **found** mes-

sage sent to $Rep(X)$ will be ignored. This algorithm is greedy in the sense that $Rep(X)$ attempts to receive as many layers as possible from each node. This helps reduce the number of connections used for delivering the required data to the client.

2) Transmission Phase: As a result of the seeking phase, a number of nodes, possibly including the root, are selected to provide required layers to client X. These nodes are called the serving nodes of client X. Each serving node R_{serve} transmits to the client a stream consisting of the layers specified in the **ack** that R_{serve} receives. These layers are sent from the corresponding sub-chunks (if the serving node is non-root) which automatically transition to the hot state, or sent from the video server (if the serving node is the root). The delivery path for this transmission is the reversal of the path on which R_{serve} received the **find** from $Rep(X)$. As the stream travels on this delivery path, each intermediate node may cache data into corresponding sub-chunks in the way we discuss in Section III-A.

In order to receive J layers, the client needs no more than J unicast connections to the upstream. This is possible since the total bandwidth required for these connections equals the bandwidth needed for V^J. The client is able to support this bandwidth because the client makes a request based on its bandwidth capacity (see Section II-B). The client must be able to reassemble streams coming from multiple connections into a single stream for rendering. This can be done by using a workahead buffer to store frames coming before their playback times, or by using a technique such as [19], where its authors proposed ways to reassemble "thin" streams into a single "thick" stream.

3) Leaving Phase: This phase can occur at any time client X asks its representative $Rep(X)$ for service disconnection. In response, $Rep(X)$ sends a **quit** message toward each serving node in the reverse direction of the corresponding delivery path. This **quit** includes information about the layers no longer needed. Upon receipt of this message, each intermediate node R removes the entry for X from the delivery schedule. Afterwards, suppose that R is currently caching the layers previously destined for client X into n sub-chunks, among which are $k \leq n$ sub-chunks in the hot state. Since hot sub-chunks cannot be released, node R just needs to unsubscribe for the other $n-k$ sub-chunks. For this purpose, R releases these sub-chunks and sends a **quit** containing the information about the $n-k$ layers corresponding to them to the upstream. The same procedure repeats in the upstream.

4) Example: We give an illustrative example in Figure 4 where the overlay topology is drawn in Figure 3. We assume to have only one video V encoded into four layers v^1, v^2, v^3, and v^4. Each node has only one chunk for caching, which is divided into four sub-chunks corresponding to four different layers. Suppose a new client (represented by node R) requests the video of the best quality (i.e., V^4). At this time, $R1$ and $R6$ are currently caching v^1 and v^2 as a result of an earlier request for V^3 by a client represented by node R_8. Similarly, $R2$ and $R5$ are currently caching v^2, v^3 and v^4 as a result of an earlier request for V^4 by a client represented by node R_{11}. The representative R searches for all four layers inside the overlay network. Suppose that $R6$ is selected to provide both layers v^1 and v^2 on path $R6$-$R9$-R, and $R5$ is selected to provide both layers v^3 and v^4 on path $R5$-$R7$-$R10$-R. The serving nodes $R5$ and $R6$ send those layers from their cache to the new client and all the intermediate nodes on the corresponding delivery paths do the caching if their sub-chunks are not hot. For example, $R9$ tries to cache $\{v^1, v^2\}$, $R7$ and $R10$ try to cache $\{v^3, v^4\}$, and R tries to cache all those layers.

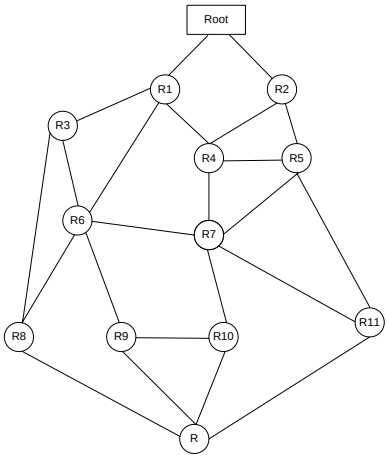

Fig. 3. Example: Overlay topology

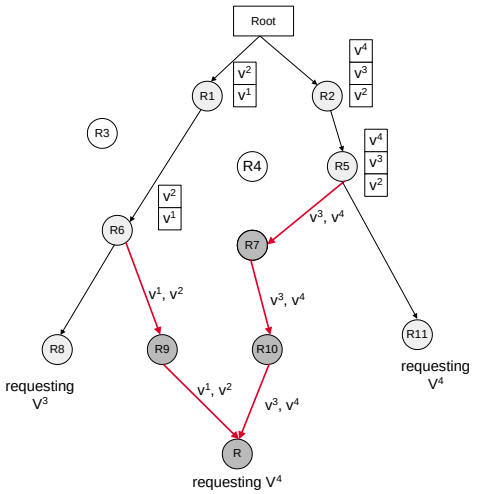

Fig. 4. Example: Late join without additional server bandwidth

This example (see Figure 4) shows that the new client can join existing server streams *late* but is still able to get the *full* requested service from some non-root nodes. This does *not* need allocation of additional server bandwidth. Therefore, LRM is very efficient in saving server bandwidth. In contrast, in the conventional multicast, since the new client cannot join any existing multicast group due to its lateness, the server has to create a new stream for this client, which is not a scalable solution.

IV. Performance Evaluation

The study on the potential performance of LRM focused on the server scalability in providing heterogeneous VOD services to clients and was done based on simulations. We adopted the topology of the IBM Global Network map[1] for the LRM overlay network. In this topology, a node is an LRM-enabled node

[1] http://www.nthelp.com/images/ibm.jpg (reachable as of July 25, 2002.)

representing a subnet of subscriber clients. The root node is located at the Chicago node in the map. We assume a discrete time model where a time unit is called a "second". The video database stores, by default, 20 90-minute videos, each encoded into L layers of sizes b, $2 \times b$, .., $2^{L-1} \times b$ where b equals 1 data block. The default value of L is three. Suppose that a so-called unit stream is needed for the least-quality layer, thus a better layer needs more unit streams than a worse layer. The bandwidth availability between any two adjacent overlay nodes is represented as a number of unit streams they can support. We call this bandwidth between two nodes "link bandwidth" between them. Each simulation run lasts 24 hours. A request is represented as a 4-tuple ($arrivalTime$, $nodeID$, $videoID$, QoS). $arrivalTime$ is generated following a Poisson distribution with rate λ requests per second. $nodeID$ and QoS values are generated randomly among the topology nodes and L layers, respectively. $videoID$ is chosen based on a Zipf-like distribution with a skew factor z.

The main purpose of LRM is to reduce the high demand on server bandwidth by providing services from intermediate nodes as much as possible. To measure how significantly the server bandwidth is saved from that, we estimate the value of *bandwidth saving* which is computed as the average ratio between the amount of data provided by intermediate nodes to the total amount of data requested. This value only applies for those requests satisfied, not for those failed. Another performance metrics for study is *system throughput*, computed as the ratio between the number of "served" requests to the total simulated time. A higher throughput implies a more scalable system.

We study the performance of LRM under effects of various factors: network link bandwidth, request rate, video access pattern, number of chunks, subchunk size, and number of nodes installing LRM software. The default values for them are 100 streams, λ = 0.5 req/sec, z = 0.7, 10 chunks, 10 minutes, and 8 nodes, respectively. We compare LRM to the conventional approach in which the conventional multicast is used to support client requests of various QoS levels. This conventional approach is actually the same as the LRM approach without employing caches. We note that the comparison between LRM and the conventional multicast (CM) is done only in terms of throughput, because the concept of server bandwidth saving in this context is irrelevant to CM where no cache is used to off load the server bandwidth.

The results for system throughput is plotted in Figure 5. When the link bandwidth increases, throughput in both methods also increases; however, LRM always performs two to four times better than CM. One interesting thing to note here is the effect of arrival request rate. Besides outscoring CM in all possible choices of request rates, throughput in LRM is monotonically increasing swhile in CM it actually starts decreasing at a certain request rate, 0.3 in this case. Also, when the request rate is small (less than 0.3), LRM achieves optimality in terms of throughput. Skew factor does not play much of a role in improving the throughput of LRM, until the skew factor reached 0.7 had we observed some small increases in the throughput. Associated features of LRM such as number of LRM-enabled nodes, subchunk size, and number of chunks can boost the throughput significantly when we increase them. Increasing each of these three factors to some reasonable extent can help to achieve 3 times better throughput comparing to CM. This study shows that although increasing more complexity, LRM pays back a significantly improved throughput.

The results for bandwidth savings is plotted in Figure 6. Though obvious, these are for the illustrative purpose only. The lower the link bandwidth is, the higher the server bandwidth savings can be obtained. It is reasonable since with more server bandwidth, the server has a higher chance to service requests. Therefore, a less saving is observed. Similarly, the lower the request rate is, the better server bandwidth saving (SBWS) we can gain. The saving tends to be a constant when a certain request rate is reached ($\lambda \geq 0.3$). Again, skew factor does not affect SBWS much. Some small gain in SBWS is possible when the skew factor is high. As in the case of throughput, increasing any of the three factors, number of LRM-enabled nodes, subchunk size and number of chunks, will improve the server bandwidth saving. The results of this study do not lie beyond our anticipation.

V. Conclusions

LRM provides many desirable benefits as follows:

- *Better service latency*: In a VOD system, the server is prone to become a bottleneck. Consequently, when a video needs to be sent from the server, there might be a blocking time until a stream is created. In LRM, since clients have a high chance to join existing multicast instead of waiting for the next available streams from the server, the average service latency is improved.
- *Reduced server bandwidth demand*: Many layers destined for earlier clients are cached in the overlay. The cached data can be used for new clients whose request includes them. Therefore, only remaining layers not cached in the overlay are transmitted from the root node. Consequently, a few additional server streams are needed.
- *Efficient and feasible implementation on the Internet*: Many works on VOD and client heterogeneous services assume the existence of IP Multicast which currently is not widely available on the Internet due to fundamental concerns. LRM deploys overlay nodes to implement the multicast paradigm based on IP Unicast only, and proposes a caching policy to efficiently provide services of various qualities to the clients.

Although our simulations confirmed the above benefits, LRM could introduce more complexity to the system in practice. We are currently building a prototype to experience the potential hurdles and to explore the advantages of the LRM approach. We are also investigating a better model for the two-level cache design at each overlay node, and how LRM works with different overlay construction and maintenance techniques.

References

[1] 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.

[2] S. Chen, K. Nahrstedt, and Y. Shavitt. A qos-aware multicast routing protocol. In *Proc. IEEE INFOCOM*, 2000.

[3] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, pages 1–12, 2000.

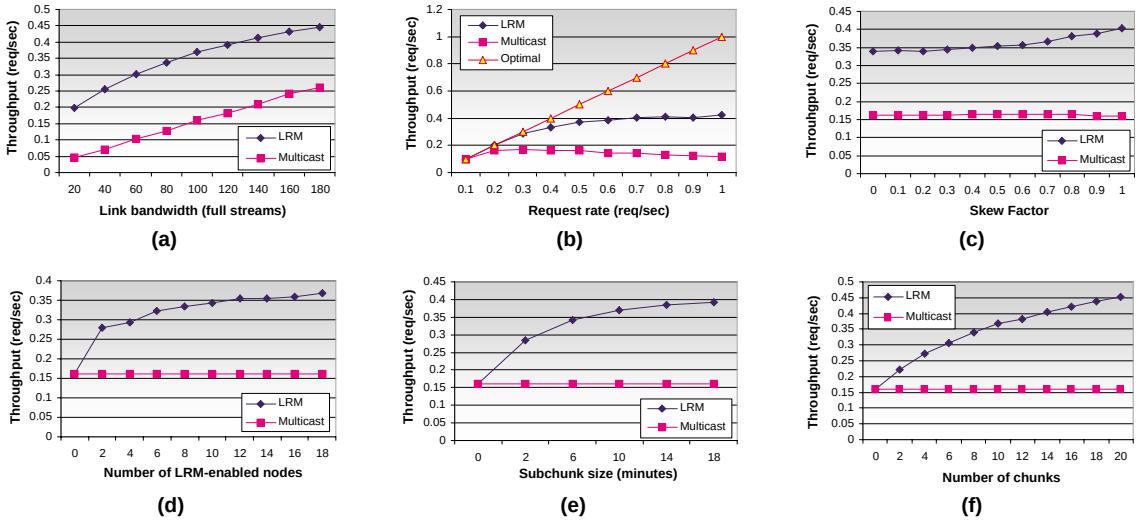

Fig. 5. Study on system throughput

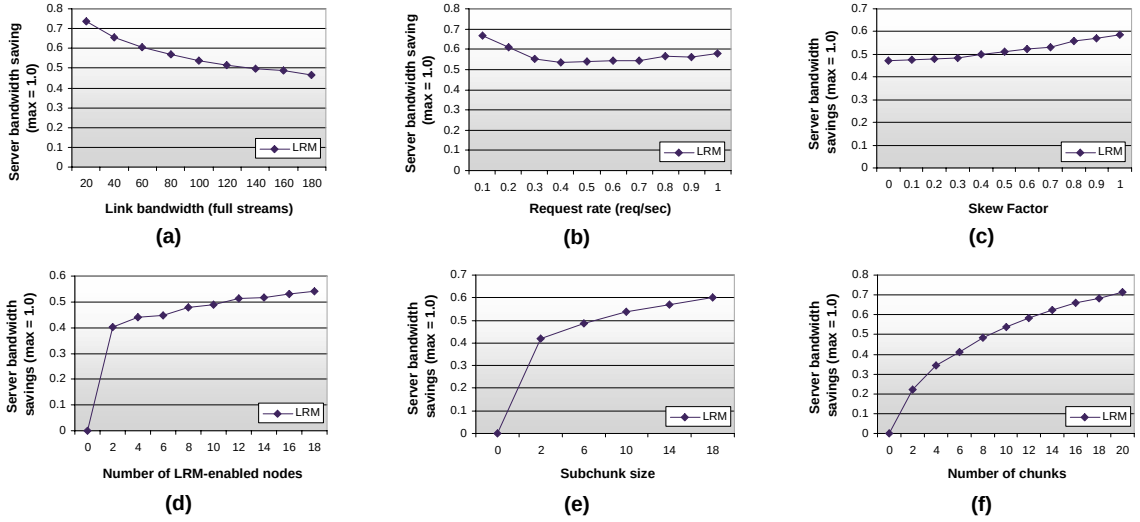

Fig. 6. Study on server bandwidth saving

[4] A. Dan, Y. Heights, and D. Sitaram. Generalized interval caching policy for mixed interactive and long video workloads. In *Proc. of SPIE/ACM Conf. on Multimedia Computing and Networking*, pages 344–351, San Jose, California, January 1996.

[5] A. Dan and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *ACM MULTIMEDIA'98*, San Francisco, October 1998.

[6] S. Deering. Host extension for ip multicasting. *RFC-1112*, August 1989.

[7] P. Francis. Yallcast: Extending the internet multicast architecture. In *http://www.yallcast.com.*, September 1999.

[8] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proc. of ACM MULTIMEDIA*, pages 191–200, Bristol, U.K., September 1998.

[9] K. A. Hua, D. A. Tran, and R. Villafane. Caching multicast protocol for on-demand video delivery. In *Proc. of the ACM/SPIE Conference on Multimedia Computing and Networking*, pages 2–13, San Jose, USA, January 2000.

[10] J. Jannotti, D. K. Gifford, and K. L. Johnson. Overcast: Reliable multicasting with an overlay network. In *USENIX Symposium on Operating System Design and Implementation*, San Diego, CA, October 2000.

[11] X. Li, S. Paul, P. Pancha, and M. Ammar. Layered video multicast with retransmission (lvmr). In *Proc. IEEE INFOCOM*, 1998.

[12] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4, pages 117–130, New York, 26–30 1996. ACM Press.

[13] D. Pendakaris and S. Shi. ALMI: An application level multicast infrastructure. In *USENIX Symposium on Internet Technologies and Systems*, Sanfrancisco, CA, March 26-28 2001.

[14] S. Ramesh, I. Rhee, and K. Guo. Multicast with cache (mcache): An adaptive zero-delay video-on-demand service. In *Proc. of IEEE INFOCOM*, San Diego, USA, 2001.

[15] S. Sheu, K. A. Hua, and W. Tavanapong. Chaining: A generalized batching technique for video-on-demand. In *Proc. of the IEEE Int'l Conf. On Multimedia Computing and System*, pages 110–117, Ottawa, Ontario, Canada, June 1997.

[16] L. Vicisano, L. Rizzo, and J. Crowcroft. Tcp-like congestion control for layered multicast data transfer. In *Proc. IEEE INFOCOM*, 1998.

[17] B. Wang, S. Sen, M. Adler, and D. Towsley. Optimal proxy cache allocation for efficient streaming media distribution. In *IEEE Infocom*, 2002.

[18] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *Proc. of the 10th International WWW Conference*, Hong Kong, 2001.

[19] L. Wu, R. Sharma, and B. Smith. Thin streams: An architecture for multicasting layered video. In *Proc. IEEE NOSSDAV*, 1997.

[20] Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Su. Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Transactions on Networking*, 8(4), August 2000.