

Application-Perceived Multicast Push Performance *

Wenhui Zhang, Wei Li, Vincenzo Liberatore
Division of Computer Science
Case Western Reserve University
10900 Euclid Av., Cleveland, Ohio 44106
{wxz24, wxl33, vxl11}@case.edu

Abstract

Multicast is an effective and scalable solution to disseminate data in the Internet. Middleware support for multicast-based data dissemination (MSMDD) aims to integrate state-of-the-art data management methods and multicast communication techniques and provide a scalable multicast-based data management layer to applications. By using multicast in MSMDD, the server can disseminate hot documents effectively. To evaluate multicast performance, previous work often utilizes network-level metrics. In this paper, we employ application level criteria to analyze the multicast push performance in MSMDD. When packets are not lost on links, our result shows that the application-perceived (or client-perceived) performance is independent of the delay from the server to clients; the multicast rate is a major factor for the performance; an end-to-end multicast system achieves almost the same application-perceived performance as IP multicast under the same multicast rates. However, IP multicast can tolerate much more packet losses than the end-to-end multicast system.

Keywords

multicast, middleware, performance

1. Introduction

Multicast is an effective and scalable solution to disseminate hot data in the Internet. A critical issue of a multicast system is to evaluate its performance. Previous work uses metrics such as available bandwidth, delay, link stress, resource usage, and control traffic overhead, which have given an effective evaluation on the network level efficiency and cost [4, 12, 9, 15]. In a bulk data multicast environment like MSMDD [8], the application-perceived (or client-perceived) performance often cannot be directly derived from the network-level metrics. Therefore, in addition

to the network-level criteria, application-level metrics are necessary to adequately capture the performance from the application viewpoint. This paper gives both theoretical and experimental analysis on the multicast push performance utilizing application-level criteria.

A variety of multicast schemes have been provided ranging from IP multicast [10] to end-to-end multicast [4, 12, 9, 11, 15]. However, allowing multicast communication introduces many non-trivial data management problems such as caching, consistency, and scheduling. We have built a middleware (MSMDD) [8, 13, 2] which aims at integrating state-of-the-art data dissemination and multicast communication techniques into a software distribution. MSMDD provides a scalable multicast-based data management layer to applications. Hence in MSMDD, it is important to optimize the application-perceived level of service.

Clients generally attempt to download multiple items. For example, in a web multicast system, a client request requires not only in HTML resource, but also the items embedded in the HTML document. The application-perceived latency is the time to receive all the items targeted by the client request. In most cases, the application-perceived latency is more meaningful to end-users because information is often inaccurate and incomplete until all the targeted items are received. Therefore, the application-perceived performance is an important factor when evaluating the feasibility and efficiency of a multicast system. On the other hand, while the network-level performance factors affect the application-perceived performance, their relationship has not been adequately investigated in earlier research. In many cases, the application-perceived performance cannot be directly reflected by the network-level factors. For instance, a small loss rate on the links may not largely reduce the bandwidth available to clients, but we will show that it can substantially increase the latency to satisfy a client request.

In this paper, we study the application-perceived performance in multicast push systems. We first give a theoretical analysis on the question, then we present the methodology

*This research is supported in part under NSF grant ANI-0123929.

and results of our experiments conducted at Emulab [3]. In the experiments, application data are multicast through the MSMDD platform from the server to clients. Meanwhile, we also make comparisons between IP multicast and an end-to-end multicast scheme. The rest of the paper is structured as follows. We give more details on the background of multicast schemes and the architecture of MSMDD in Section 2. In Section 3, we analyze the application-perceived multicast performance from a theoretical perspective. In Section 4 and 5, the methodology and results for the experimental evaluation are presented respectively. We draw the conclusions of our investigation in Section 6.

2. Background

2.1. Multicast schemes

According to the layer on which multicast is implemented, there are mainly two types of multicast schemes: IP multicast and end-to-end multicast. IP multicast is implemented at the IP layer. It is efficient since a server sends a single data copy to the network and packets are replicated and forwarded to clients by multicast-enabled routers. Reliable multicast over IP multicast, such as JRMS [20], provides reliable data delivery and higher level features such as congestion control, scalability, and security. However, the deployment of IP multicast has so far been slowed down by several challenging issues, such as the management of globally unique addresses, maintaining per group state at routers, defending from flooding attacks and needs for changes at the infrastructure level.

End-to-end multicast is implemented at the application layer. Applications maintain a self-organized logical overlay network and transmit packets along overlay edges. Communication between the nodes is based on unicast. This enables end-to-end multicast to be easily deployed in the Internet. End-to-end multicast can utilize features such as flow control, congestion control and reliable delivery that are available for point-to-point connections. However, this approach cannot utilize bandwidth as efficiently as IP multicast because copies of the same data can be transmitted over the same physical link multiple times (link stress). End-to-end multicast can also increase the delay between the server and clients since packets are relayed by end hosts.

In our experiments, we employed two multicast schemes: IP multicast and HyperCast. HyperCast is a representative of end-to-end multicast schemes and well implemented at [1]. Currently, HyperCast supports two types of overlay topologies: logical hypercubes [16] and logical Delaunay triangulations [15]. We only use the logical Delaunay triangulations scheme in the end-to-end multicast experiments because the logical hypercubes scheme has IP

multicast involved in its implementation and thus it is not a pure end-to-end multicast scheme.

HyperCast supports both UDP and TCP communication between the nodes. If TCP connections are used, HyperCast provides a reliable end-to-end multicast data delivery. We utilize the UDP communications for our experiments because of the following reasons: (1) since the other multicast system we use is IP multicast that is based on UDP, using UDP communications in HyperCast facilitates the comparison between these two schemes; (2) the server multicasts data cyclically (we will describe it in Section 2.2); even if some packets are missed, the client can get the data later so that reliable data delivery as in TCP is not required.

2.2. Multicast-based data dissemination middleware

Middleware support for multicast-based data dissemination (MSMDD) provides applications with a scalable data management layer. This middleware integrates and extends latest research work on data management issues in multicast environments. Three data dissemination methods, i.e., multicast push, multicast pull and unicast pull, are supported by this middleware. When using multicast push, the server repeatedly multicasts information. If a client wishes some information, it simply listens to the multicast channel until the requested information appears without sending an explicit request to the server. Multicast push is suitable for disseminating hot information. In multicast pull, when a client needs some information, it sends a request to the server. The server multicasts the information to all members in the multicast group. When there are multiple clients requesting the same information at roughly the same time, the server can aggregate the requests and multicast the information only once. Multicast pull is an ideal fit to disseminate warm data for which multicast push cannot be justified, while there is an advantage in aggregating concurrent client requests. Traditional unicast pull can be used to disseminate cold data. A middleware application can choose to use any combination of multicast push, multicast pull and unicast push to disseminate data. Since the details of the middleware are concealed from applications, applications cannot distinguish which data dissemination method(s) are used.

The middleware is composed of multiple components that can be switched off or on according to the needs of the middleware application and multicast scheme used. Therefore, the middleware has a flexible and extensible architecture. Figure 1 shows a possible architecture of the middleware and its relationship with the application and underlying transportation layer. The document selection component periodically collects statistics on document (identified by identifiers such as URLs) popularity, which is the probability that clients request a document. Based on this statis-

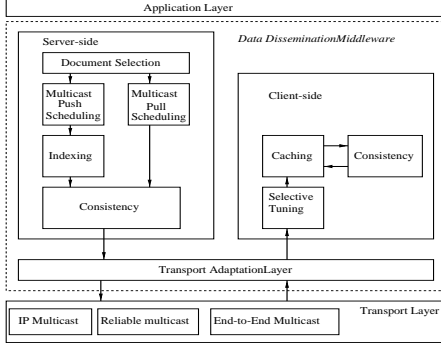


Figure 1. Middleware data dissemination architecture and its relationship with the transport and application layers.

tics, the component decides if a document is hot, warm or cold. The middleware uses multicast push to disseminate hot documents, and multicast pull and unicast pull to disseminate warm and cold documents respectively. An index of sorted index digest is multicast on the multicast push channel, which enables clients to quickly decide if a document is multicast by the push channel. The multicast push scheduling component determines the frequency and order by which hot documents are multicast. The multicast pull scheduling component resolves contention among client requests to use multicast pull channel and decides the order in which warm documents are disseminated. Caching is used to improve performance at the client side. The consistency component ensures that only the most recent committed value of a document is disseminated or stored.

In this paper, we focus our research on multicast push. In multicast push, the transmission unit is a *page*. Each multicast document is divided into a sequence of pages before it is multicast.

Different applications may require different multicast schemes, which in turn present different APIs and different capabilities. The objective of Transport Adaptation Layer (TAL) is to enable middleware to interact with various multicast schemes within a uniform interface. During the emulation, we activate the TALs for IP multicast and HyperCast.

3. Theoretical analysis

Documents denote items which can be identified by clients using identifiers (ids) such as URLs. We give the definition of an *object* as follows: the collection of documents targeted by a client request is called an object. In this sense, a client requests objects. We employ three application-level performance metrics in both theoretical analyses and emulations. (1) *Seek time*: the time that a client waits to receive the first multicast page that belongs to

the requested object. Seek time reflects how quickly users start to receive the requested information. (2) *Transfer time*: the time that the client waits to receive the rest of the object; (3) *Latency time*: the time that the client waits to receive the complete object, i.e., the sum of seek time and transfer time. Latency time denotes the user-perceived delay, which multicast systems aim to reduce. The values of these metrics perceived by the client are independent of the presence of other clients in a multicast push environment. In this section, we analyze the expected values of these metrics based on the assumption that object requests are randomly generated by clients.

The performance is influenced by the multicast scheduling at the server. It has been proved in [5] that it is sufficient to consider *cyclical* schedules in multicast scheduling algorithms. There are a variety of strategies for cyclical multicast. The MAD schedule [21, 5], which is used in our experiments, schedules pages on the basis of their access probabilities by clients. The algorithm is proved to achieve 2-approximation factor and is easy to be implemented. In practice, the MAD achieves performance close to the optimal lower bound [21]. Performance is improved when the pages of the same document are multicast contiguously in the schedule [14]. We assume that this strategy is adopted by the scheduler.

The *flat multicast schedule* is a simple cyclical schedule, whereby each page is transmitted once in every period. We limit our theoretical analysis to the case of flat schedules for the following two reasons. First, the theoretical analysis of the multicast performance is manageable when the server uses flat schedules. Second, the MAD degenerates into a basically flat schedule when the page access probabilities are close to each other. In particular, the MAD generates a flat multicast when the access probabilities are equal. Furthermore, if a subset A of the multicast documents S are much more popular than the documents in $S - A$, we would multicast A and $S - A$ in separate channels in order to reduce latency time [8]. In this way, the difference between access probabilities of pages within each channel is small.

We make some assumptions that are often employed in this research area [5, 21, 14]. The server multicasts *equal* length pages at a constant rate and at each *time-slot* one page is multicast. The length of time-slot u is inversely proportional to the multicast rate R since $u = P/R$ where P is the page size. Seek time, transfer time and latency time are measured in terms of time-slots. The set of multicast pages does not change. The pages are received by clients in the same order as they are multicast by the server although some pages can be lost. Client requests arrive at the *beginning* of time-slots at a random location of the multicasting schedule.

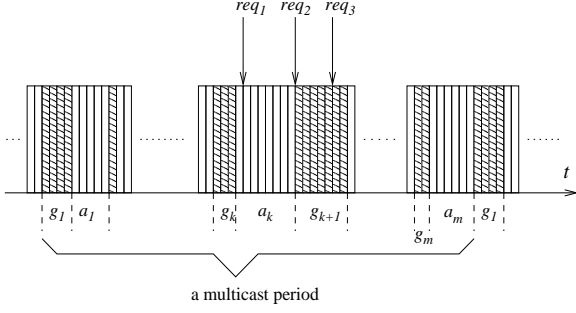


Figure 2. The requested object in a flat schedule.

3.1. Reliable delivery

First, we discuss the case where links are reliable. We define s as the total number of multicast pages. Hence, the period of a flat schedule is s time-slots. As the schedule is cyclical and object requests are uniformly distributed, we can focus the analysis on the requests arriving at random locations within one multicast period. Suppose the requested object contains m documents and document i has g_i pages for $1 \leq i \leq m$. Also, let g_{obj} denote the total pages of the object, then $g_{obj} = \sum_{i=1}^m g_i$ and $m \leq g_{obj} \leq s$. Figure 2 presents how the object is scheduled in a flat schedule, where a_i denotes the number of time-slots between the last page of a document i and the first page of the following document $(i + 1) \bmod m$ ($0 \leq i \leq m$). Since each document is multicast once in a period, $s = \sum_{i=1}^m (g_i + a_i) = g_{obj} + \sum_{i=1}^m a_i$.

Let X be the location where the request arrives within the investigated period. We have $0 \leq X \leq s - 1$. Define $L(X)$, $S(X)$, $T(X)$ as the latency time, seek time and transfer time respectively. In a specific flat schedule, a_i is fixed for $1 \leq i \leq m$. As X is uniformly distributed, we have the expected latency time ¹

$$E[L(X)] = s - \frac{1}{2} + \frac{g_{obj}}{2s} - \frac{1}{2s} \sum_{i=1}^m a_i^2,$$

the expected seek time

$$E[S(X)] = s + 1 - E[L(X)],$$

and the expectation of the transfer time

$$E[T(X)] = s + 1 - 2E[S(X)].$$

Hence, the summation of the expected seek time and latency time is roughly equal to the multicast period. When the

¹Due to the limited space, we omit the derivation here. For the same reason, we refine our presentation in the following sections. Please refer to our full paper for more detail.

expected seek time increases, both transfer and latency time linearly decrease. While the expectations are decided by the values of s , m , g_{obj} and a_i , they are independent of the size g_i of any single document. Since the length of time-slot $u = P/R$, the above analysis shows that the expected seek time, transfer time and latency time are also inversely proportional to R , therefore R is a major performance factor when data are reliably delivered.

Now we extend the analysis to the cases that a_1, a_2, \dots, a_m are also random variables which satisfy $\sum_{i=1}^m a_i = s - g_{obj}$ and $a_i \geq 0$ for $1 \leq i \leq m$. We introduce $V = (a_1, a_2, \dots, a_m)$, and an assignment $v = (v_1, v_2, \dots, v_m)$ to V implies $a_i = v_i$ for $1 \leq i \leq m$. We aim at obtaining the following expectations: $E[L(X, V)]$, $E[S(X, V)]$ and $E[T(X, V)]$.

We assume that any of the $s - g_{obj}$ pages that do not belong to the object is randomly scheduled in one of the m gaps between the object's documents. Then, each page that does not belong to the object has a probability $1/m$ to be scheduled in gap i which is between document i and the following document $(i + 1) \bmod m$ ($0 \leq i \leq m$). Therefore, a_i is a random variable that follows a binomial distribution with parameters $1/m$ and $s - g_{obj}$. We have the following expected values

$$E[L(X, V)] = s - \frac{(s - g_{obj})(s - g_{obj} + 2m - 1)}{2ms},$$

$$E[S(X, V)] = s + 1 - E[L(X, V)],$$

and

$$E[T(X, V)] = s + 1 - 2E[S(X, V)].$$

When g_{obj} is small compared with s , $E[L(X, V)] \approx (1 - \frac{1}{2m})s$; Figure 3(a) presents the expectation values and the corresponding m where $s = 200$. As the number of documents m grows, the expected transfer time and latency time increases while the expected seek time is reduced. Figure 3(b) investigates the relationship between the expected values and g_{obj} when $s = 200$ and $m = 3$. In both figures, as the number of documents or pages grows, its effect on the expectations flattens out.

3.2. Packet loss

Network links may drop packets and performance is impaired by this data loss. Suppose that p ($0 < p < 1$) is the packet loss rate of the multicast path between the server and a client. When we set the page length properly, a page is encapsulated in an IP packet without being fragmented. Thus, the probability of reliably delivering a page from the server to the client is $1 - p$. In any cyclical multicast schedule, let c_i be the number of times document i appears. In

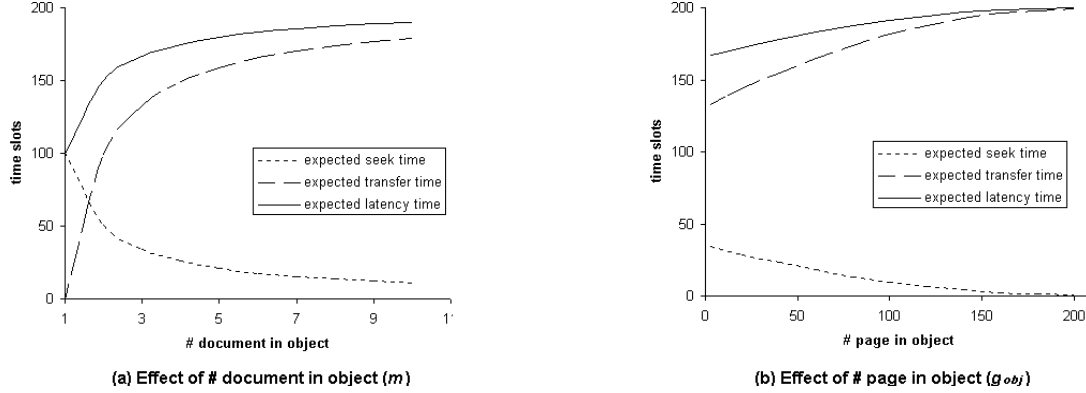


Figure 3. Effect of object size.

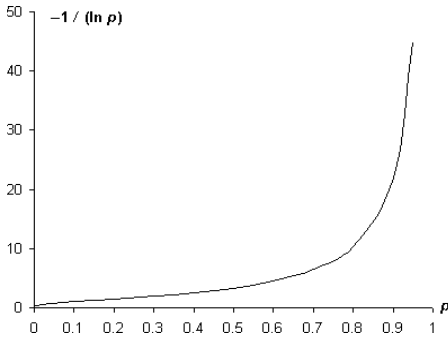


Figure 4. Effect of packet loss.

each period, the client is expected to miss $g_i p^{c_i}$ pages of document i and $\sum_{i=1}^m g_i p^{c_i}$ pages of the object. We are interested in the number of cycles k within which the client is expected to miss less than 1 page of the object. The value of k satisfies

$$\sum_{i=1}^m g_i p^{c_i k} < 1, \quad 0 \leq k, \quad 0 < p < 1.$$

In a flat multicast schedule, where $c_1 = c_2 = \dots = c_m = 1$, we have

$$k > \frac{\ln \sum_{i=1}^m g_i}{\ln \frac{1}{p}} = -\frac{\ln g_{obj}}{\ln p}, \quad 0 < p < 1.$$

Figure 4 presents the values of $-1/\ln p$ (proportional to k) corresponding to the values of p ranging from 0 to 0.95. The performance deteriorates as the packet loss rate grows. Moreover, as k depends logarithmically on g_{obj} , larger objects suffer packet losses more than smaller objects.

4. Methodology

We have given a theoretical analysis of application-perceived multicast push performance in the above section;

however, there are several limitations. Firstly, we assume the server utilizes flat schedule in the theoretical analysis, but in practice, the schedule is not necessarily flat. For instance, the schedule generated by the MAD algorithm is often not flat since the access probabilities of different documents are seldom the same in practice. Secondly, the distribution of client object requests is not necessarily uniformly distributed across the multicast session. Thirdly, when we discuss the case where a_1, a_2, \dots, a_m are also random variables, we assume that the $s - g_{obj}$ pages, which do not belong to the investigated object, are independent of each other. In fact, the pages that belong to the same object are dependent of each other as they are multicast contiguously. Therefore, in this paper, we also give an experimental analysis based on emulations. In this section, we first discuss the strategies taken by the middleware, and then describe how we obtain the traces for the experiments.

4.1. Strategies in middleware

At the server side, the middleware maintains a set of multicast documents specified by the application via the middleware interface. The documents are associated with attributes including document URIs, size and their access probabilities. The middleware does initializations according to the selected multicast scheme. For example, the URIs are transformed into the specific formats known to the client middleware. When the application intends to multicast data, the middleware activates the multicast scheme at the transport layer and establishes a multicast channel. The middleware starts to disseminate data as soon as it detects that a client joins the channel.

The server's multicast push scheduling component uses the MAD algorithm to determine the frequency and order in which documents are multicast. The MAD is a simple and practical 2-approximation cyclical algorithm. Each document is assigned a value p_i which is the probability that document i is requested by clients. When multicasting the

data, the MAD algorithm also maintains a value s_i for document i recording the time since the last time document i was multicast. The MAD algorithm multicasts a document i with the maximum value of $(s_i + 1)^2 p_i$. The pages of the document are multicast contiguously in the schedule.

We assume clients have knowledge of document ids of the documents within an object, so that it triggers requests for all the documents of the object when the object is requested. This scheme can be easily implemented in the multicast middleware[19]. Although we believe cache will improve the performance, the client switches off cache function since the HTTP server logs used for the emulations (discussed in Section 4.2) do not report requests that are satisfied by intermediate caches.

During the multicast session, a page is encapsulated in a middleware frame. The frame is composed of a header of 30 bytes and a data page. To control accurately the multicast rates in emulation, we set the maximum page length to be 512 bytes. This prevents a page from being fragmented at the IP layer since most links in the Internet have MTUs that are no less than 1500 bytes.

4.2. The traces

The HTTP servers of Soccer World Cup are among the hottest spots of the Internet. During peak time, the servers receive 10 million requests per hour. As a result, the World Cup site is one of the busiest recorded so far, which makes it an ideal testbed for multicast data dissemination. We extract the experiment traces from the log of HTTP servers for the Soccer World Cup 98. The traces begin at 30/Jun/1998:21:00:00 (coord.univ.time), and end at 30/Jun/1998:21:40:00 (coord.univ.time), which is one of the most active periods in the server logs. In [14], we have elaborately discussed how to decide the hottest χ fraction of bytes to be multicast. In this paper, we will take the same χ value that is 0.0007. These hottest data include 65 documents composed of 208 pages.

When we extract the client traces, only those client requests for hot documents that received 200 (OK) and 304 (Not Modified) response code are considered. There are 25149 clients that request for these hot documents. These clients created 5390667 requests, which corresponds to about 2246 requests per second on average. We choose 8 client traces for the emulation experiments. The numbers of requests in these traces range from 196 to 14060. Every request accesses a document identified by its id. We limit our performance analysis on the period when the multicast tree is stable because the efficiency of initialization and decomposition of a multicast tree is a big topic which exceeds the scope of this paper. We discard the first and last 5 minutes running result in order to remove the influence of multicast tree initialization and decomposition. The total remaining

| Trace | tot. req. | eff. req. | eff. obj. | doc/obj |
|---------|-----------|-----------|-----------|---------|
| trace 1 | 196 | 177 | 163 | 1.086 |
| trace 2 | 570 | 447 | 379 | 1.179 |
| trace 3 | 1001 | 710 | 395 | 1.797 |
| trace 4 | 1995 | 1608 | 788 | 2.041 |
| trace 5 | 4010 | 2877 | 1057 | 2.722 |
| trace 6 | 7845 | 5666 | 1598 | 3.546 |
| trace 7 | 10231 | 7997 | 1716 | 4.66 |
| trace 8 | 14060 | 10521 | 1458 | 7.216 |

Table 1. Client traces.

emulation time is 30 minutes.

Although clients are often interested in a collection of related documents, the HTTP protocol does not aggregate document requests into object requests. In order to get objects for our experiments, we take the heuristic view that if two documents are requested within one second of each other in the client trace, they belong to the same object. In every client trace, there are at most 1800 objects during the effective time. Table 1 is a summary of the 8 client traces. Column 2 denotes the total requests in the traces. Column 3 and 4 denote the number of requests and the number of objects respectively within the effective time (30 minutes). Column 5 denotes the average documents of an object during the effective time.

5. Evaluation

In this section, we present our emulation results. We create the topologies for the emulation using the BRITE topology generator [18]. In the experiments running on reliable links, the topology is a Waxman model [22] and consists of 16 routers and 32 links. Thus, the average node degree is 4. We attach 9 end-hosts separately to 9 routers labeled from R0 to R9. The multicast server C0 runs on the end host connecting to R0, and the clients run on the rest end hosts. We call these 8 clients C1 through C8 according to the routers to which they are connected. During each experiment, the server multicasts data at a stable rate and the 8 clients use the same client trace selected from Table 1. For example, in one experiment, clients C1 through C8 all use trace 1; in another experiment all use trace 2, and so on. The server is launched at the beginning of an experiment, and then the clients join the multicast channel one by one following a fixed sequence used by every experiment. The IP multicast tree is observed to be fixed in the IP experiment. Since we specify the logical addresses of the clients in the HyperCast experiments, the overlay multicast tree is also fixed. Fixing multicast tree facilitates the comparison between the experiments and the assignment of server-to-client delays for the clients (we will describe the delay assignment in the Section 5.1). A client leaves the channel

| client | delay(ms) | client | delay(ms) |
|--------|-----------|--------|-----------|
| C1 | 0 | C5 | 1200 |
| C2 | 300 | C6 | 1500 |
| C3 | 600 | C7 | 1800 |
| C4 | 900 | C8 | 2100 |

Table 2. Server-to-client delays.

after all its requests have been satisfied and the experiment ends if all the 8 clients have left the channel. As discussed in Section 4.2, we discard the first and last 300 seconds’ performance result recorded at every client to remove the effect of multicast tree initialization and decomposition.

We select six multicast rates: 9.6 Kbps, 56 Kbps, 300 Kbps, 512 Kbps, 1 Mbps, 1.544 Mbps. Among them, 56 Kbps and 1.544 Mbps are equal to the bandwidths of modems and T1 lines respectively, and 9.6 Kbps and 512 Kbps are the bandwidths mobile devices may experience. We assign 100M bandwidth to each physical link so that data delivery is reliable unless we intentionally assign a packet loss rate to the link.

The experiments are conducted at Emulab. All the machines run on FreeBSD 4.7 and are equipped with 850MHz Intel Pentium III processors and 512 MB RAMs.

5.1. Effect of delay

Delay measures the time for a packet to be transmitted from the server to a client over the multicast tree. We assign different server-to-client delays to the 8 clients as shown in table 2. In the IP multicast experiments, this is done by assigning delays to links on the IP multicast tree. For instance, on the multicast path from the server C0 to client C3, we assign 600 ms to the link between router R3 to C3 and assign 0 ms to the other links. Thus the delay from C0 to C3 is 600 ms. In the HyperCast experiments, we first fix the overlay multicast tree by specifying the logical addresses of the server and clients, then assign delays to links according to the overlay tree. Let $d(s, u)$ denote the unicast delay from the server s to client u , and $d(u, v)$ denote the delay on the unicast path from client u to client v . Then $d(s, v) = d(s, u) + d(u, v)$ where client u is the parent of client v . For example, the delay on the unicast path from C0 to C2 is 300 ms; since C2 is the parent of C7 and the server-to-client delay from C0 to C7 is 1800 ms, we assign a delay of 1500 ms to the unicast path from C2 to C7.

Figure 5 gives the performance results when the server multicast rate is 56 Kbps and all 8 clients emulate client trace 6. The multicast scheme in the Figure 5(a) is IP multicast while it is HyperCast in Figure 5(b). The three lines in the figures represent average seek time, transfer time and latency time respectively. In either IP multicast or Hypercast case, although the clients’ delays range from 0 ms to

2100 ms, the clients have almost the same average seek time, transfer time and latency time. Larger delays do not imply worse performance. Similar results are observed in other experiments with different multicast rates and client traces. Based on the results, we conclude that none of seek time, transfer time and latency time is sensitive to delays.

5.2. Effect of multicast rate

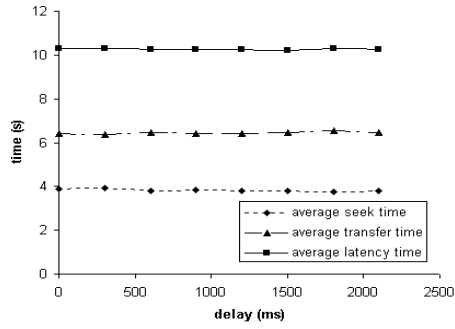
While we investigate the relationship between the multicast rate and client-perceived performance, we average the performance values on all the 8 clients in each experiment. Figure 6(a) and 6(b) show the results of the clients running on client trace 3 using IP multicast and HyperCast respectively. In the figures, as the multicast rate increases, the average seek time, transfer time and latency time decrease dramatically. The performance values are roughly inversely proportional to the multicast rate. We have similar emulation results in the experiments on other client traces. Therefore, to improve performance, the multicast server should use as much bandwidth as it can get between the server and clients during the multicast session. However, the rate should not exceed the available bandwidth in order to avoid packet loss on links because packet loss impairs performance remarkably (we will discuss the cases of packet loss in the Section 5.4).

The result also proves the effectiveness of multicast. Even a rate of 56 Kbps, which is no more than a modem’s bandwidth, can provide tolerable average latencies. When the rate is 300 Kbps, which is applicable in most broadband networks, the average latency time is less than 4 seconds.

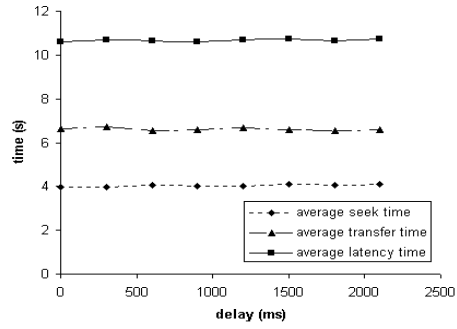
The latency time using HyperCast is outperformed from 2% to 5% by that in IP multicast. In the HyperCast experiments, when a middleware data segment is sent to the transport layer, HyperCast adds its own header to the front of the segment to form an HyperCast frame before sending the data to the underlying network. The overhead of the HyperCast frame header prevents HyperCast from delivering application data as efficiently as IP multicast does. However, the performance difference is small and HyperCast is still effective for disseminating data.

5.3. Effect of object size

As we look into an object, we find that the number of documents and pages within the object also affects the performance. Figure 7 gives the performance versus the average number of documents per object in the 8 client traces, where the server multicast rate is 56 Kbps. The seek time decreases with the growth of the number of documents per object. An intuitive explanation is that a new received page is more likely to belong to a requested object when the object contains more documents. On the other hand, the trans-

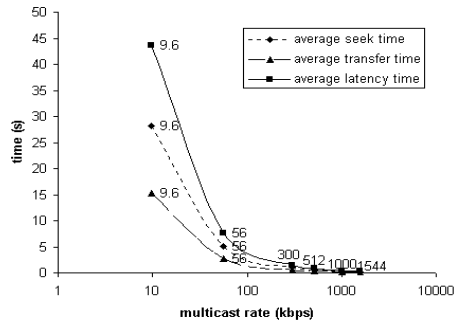


(a) IP Multicast

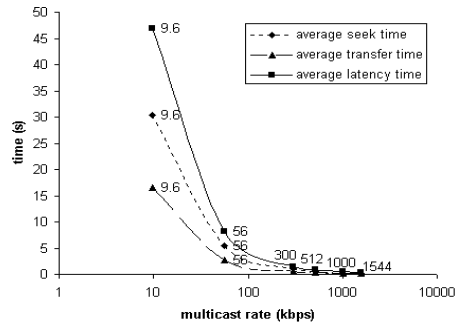


(b) Hypercast

Figure 5. Effect of delay.

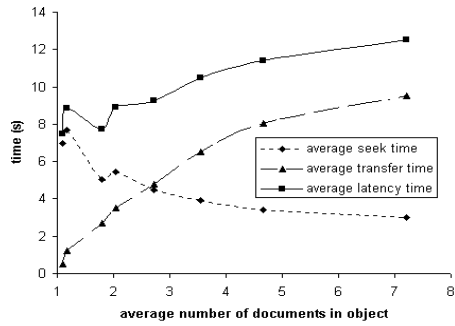


(a) IP Multicast

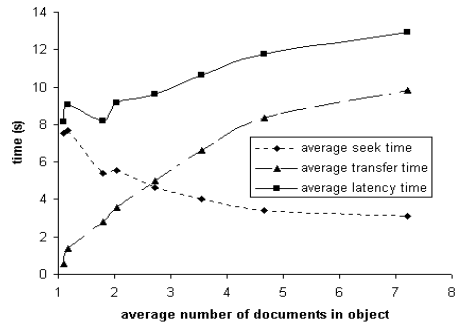


(b) Hypercast

Figure 6. Effect of multicast rate.

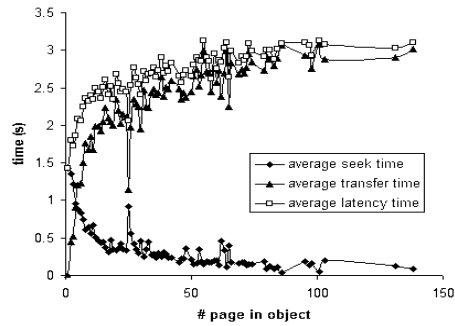


(a) IP Multicast

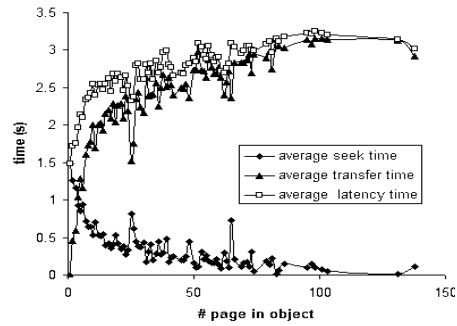


(b) Hypercast

Figure 7. Effect of doc/obj.



(a) IP Multicast



(b) Hypercast

Figure 8. Effect of page/obj.

fer time and latency time increase as the number of documents per object grows. This is because an object with more documents needs more time to receive all its pages from the multicast channel. In the figures, when the number of documents per object is less than 2.7 docs/obj, the seek time is the dominating factor of the latency time. The transfer time becomes the dominating factor when the number of documents per object is more than 2.7 docs/obj. The relationship between the number of documents per object and the transfer time shown in the figures provides additional evidence of the importance of considering document dependencies in multicast scheduling (as in [14]).

Figure 8 plots the performance corresponding to the number of pages in an object. They are running on client trace 8 with the multicast rate of 300 Kbps. With the increment of the number of pages, the seek time decreases while the transfer time and latency time increase. However, the effect becomes less pronounced when the number of pages turns larger.

5.4. Packet loss

In the previous experiments, the multicast tree is reliable in delivering pages. However in the Internet, links are likely to experience packet loss. This subsection discusses the cases in which packets are lost. We set up two machines in Emulab to run a multicast server and client respectively. We assign different loss rates on their connection to emulate the various degrees of packet loss.

Figure 9 shows how the packet loss affects the performance. The experiments emulate client trace 8 with the multicast rate of 56 Kbps. While the packet loss has a small impact on the seek time, it substantially increases the transfer time which makes up most of the latency time. We observe that a small loss rate 1% can cause the latency time to increase 14% and 15% in the IP multicast and HyperCast experiments respectively. When the loss rate is 9%, the latency time is increased by 86% and 91%. Figure 9(a) and 9(b) plot the performance under a wide range of packet loss rate. The IP multicast experiments' results follow our theoretical analysis in Figure 4. The performance degenerates with the growth of packet loss and drops dramatically around $p = 0.8$. However, the curve of the HyperCast in Figure 9(d) reaches a knee around $p = 0.5$. The reason is that HyperCast overlay multicast network becomes very difficult to be maintained when more than 50% messages are lost. The HyperCast overlay multicast tree periodically crashes during the multicast session². Therefore, it is important for an end-to-end multicast system in challenged networks to have its overlay network strongly resistant to packet loss.

²Note that in the case of extremely high rates of loss, the IP multicast tree can also crash in similar ways.

6. Conclusions

Based on our theoretical and experimental analysis, we make the following observations.

In the cases that there is no packet loss, the application-perceived multicast push performance is insensitive to the server-to-client delay. The multicast rate, which is limited by the available bandwidth from the server to clients, is a dominating factor that affects performance. The average seek time, transfer time and latency time are all roughly inversely proportional to the multicast rate. At the same time, the object size also remarkably affects the performance. The more documents and pages in a requested object, the more waiting time the client will experience. However, as the number of documents or pages increases, its effect on performance flattens out.

Experiments also show that, using multicast push in MSMDD, the server satisfies client requests within short latency time with low bandwidth cost. When the server multicasts at the same rate, the average latency time perceived by applications when using HyperCast is almost the same as that perceived by applications when using IP multicast.

The application-perceived performance is notably impaired by packet loss on the multicast trees. When the loss rate is below 0.5, the performance damage in the HyperCast experiments is near that in the IP multicast experiments. However, when the loss rate becomes more serious, performance in the HyperCast experiments drops dramatically due to the crashes of the HyperCast overlay network. Thereby, HyperCast does not tolerate packet loss as much as IP multicast. It is expected that if Forward Error Correction based on erasure codes [17] is applied to the data transmission in the multicast system, it can reduce the performance damage caused by packet losses. Further work is needed to evaluate how effective the erasure codes improve the application-perceived performance.

As shown in the paper, a large multicast rate can effectively improve the performance. However, the evident impairment on the performance by packet losses requires the rate to keep below the available server-to-client bandwidth. The selection of multicast rates is made more complicated by the fact that the available bandwidth to different clients is often various in a heterogeneous network. A solution to this problem is *layered multicast* [6, 7], in which the server multicasts the data at different layers (channels) with different rates and clients subscribe to as many layers as they can only if the accumulated rate at the subscribed layers does not exceeds the available bandwidth. The scope of our evaluation of the application-perceived performance is to be extended to the layered multicast environment. This study concerns to a number of challenging research topics such as the layered multicast scheduling that we are actively working on.

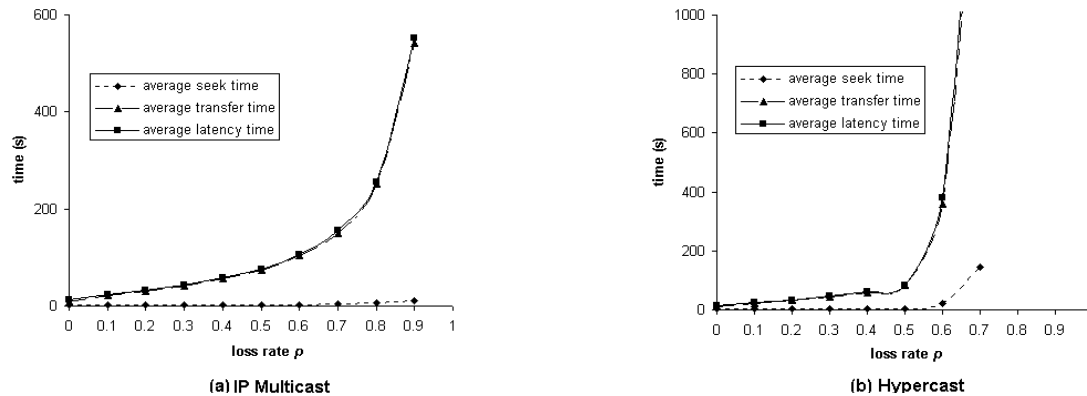


Figure 9. Effect of packet loss

Acknowledgements

We would like to thank the Flux research group at the University of Utah for providing the network testbed and the anonymous IPDPS 2004 reviewers for their helpful comments.

References

- [1] Hypercast. <http://www.cs.virginia.edu/~mngroup/hypercast/>.
- [2] Middleware support for multicast-based data dissemination. <http://dora.eeap.cwru.edu/mware/>.
- [3] Utah emulation facility. <http://www.emulab.net/>.
- [4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *In Proc. of ACM SIGCOMM*, Apr. 2002.
- [5] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27:518–544, 2002.
- [6] S. Bhattacharyya, J. Kurose, D. Towsley, and R. Nagara-jan. Efficient rate-controlled bulk data transfer using multiple multicast groups. *In Proc. of IEEE INFOCOM 1998*, pages 1172–1179.
- [7] J. Byers, M. Luby, and M. Mitzenmacher. Fine-grained layered multicast. *In Proc. of IEEE INFOCOM 2001*, 2001.
- [8] P. Chrysanthis, V. Liberatore, and K. Pruhs. Middleware support for multicast-based data dissemination: A working reality. *WORDS 2003*, pages 265–272, 2003.
- [9] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. *In Proc. of ACM SIGMETRICS*, pages 1–12, 2000.
- [10] S. Deering. Multicast routing in datagram internetworks and extended lans. *In Proc. of ACM SIGCOMM*, pages 55–64, 1988.
- [11] P. Francis. Yoid: Extending the internet multicast architecture. *Technical report, AT&T Center for Internet Research at ICSI (ACIRI)*, 2000.
- [12] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. J. W. W. O’Toole. Overcast: Reliable multicasting with an overlay network. *In Proc. of OSDI*, 2000.
- [13] W. Li, V. Penkrot, S. Roychowdhury, W. Zhang, P. Chrysanthis, V. Liberatore, and K. Pruhs. An optimized multicast-based data dissemination middleware: A demonstration. *In proc. of the 19nd International Conference on Data Engineering (ICDE 2003)*, 2003.
- [14] V. Liberatore. Multicast scheduling for list requests. *In Proc. of IEEE INFOCOM 2002*, 2002.
- [15] J. Liebeherr and M. Nahas. Application-layer multicast with delaunay triangulations. *IEEE Globecom*, 2001.
- [16] J. Liebeherr and B. S. Sethi. A scalable control topology for multicast communications. *In Proc. of IEEE INFOCOM 1998*, 1998.
- [17] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *In Proc., 29th ACM Symposium on Theory of Computing (STOC’97)*, pages 150–159, 1997.
- [18] A. Mdina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. *In Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MAS-COTS ’01*, 2001.
- [19] R. Agrawal and P.K. Chrysanthis. Efficient data dissemination to mobile clients in e-commerce applications. *In proc. of the Third IEEE Int’l Workshop on Electronic Commerce and Web-based Information Systems*, 2001.
- [20] P. Rosenzweig, M. Kadansky, and S. Hanna. The java reliable multicast service: A reliable multicast library. *Technical Report SMLI TR-98-68, Sun Microsystems*, 1998.
- [21] C. Su, L. Tassiulas, and V. Tsotras. Broadcast scheduling for information distribution. *In Proc. of IEEE INFOCOM 1997*, 1997.
- [22] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1671–1622, 1988.