WWW Multicast Delivery with Classes of Service

Paul Patrick White
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
England

email: p.white.cs.ucl.ac.uk
phone: +44 171 419 3701

Jon Crowcroft
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
England

email: j.crowcroft.cs.ucl.ac.uk
phone: +44 171 380 7296

Abstract

At busy WWW servers the inter-reference times of some documents may be low enough for duplicate requests to be accumulated and held over some small time interval before servicing the duplicate requests via a single multicast at the end of the time interval. The use of multicast in this way will reduce the bandwidth consumption on the server output link. We present a scheme that extends these principles so that each request from a client includes a latency class which reflects the requesting client's latency requirements. The server will consider the latency class of a request when determining the maximum holding time applicable to the request, that is the maximum delay that may occur between the server receiving the request and passing the requested file to the output buffer queue for transmission. Compared to a scheme that applies the same maximum holding time to all requests, our scheme yields a superior resource saving in an environment with heterogeneous client latency requirements. We evaluate the performance of our scheme through simulation using a request access pattern obtained from a WWW server.

1. Introduction

The majority of traffic in the World-Wide-Web(WWW) is generated by Hypertext Transport Protocol(HTTP) [11][12] transactions between clients and WWW servers. In its simplest form an HTTP transaction consists of a request from client to server and a response from server to client. Both request and response have traditionally been carried over a single Transmission Control Protocol(TCP) connection which offers a reliable, unicast, full-duplex service. With a unicast-only server and no co-operating intermediate entities between the server and clients, some redundancy among the servicing of requests can be noted as follows:

1) Most of the requests are not unique, that is they represent a file that has been requested before and which is likely to be requested again.
2) Over a period of time, the same file may be transmitted across some network links more than once.
3) At a busy server, identical requests may arrive within a short time period

1) and 2) permit bandwidth savings to be achieved using caching while the addition of 3) permits bandwidth savings to be achieved using multicast. The multicast approach is the focus of this paper and realises bandwidth savings by satisfying a set of identical requests received over some short time period using a single multicast transmission at the end of the time period rather than a separate unicast transmission for each identical request.

Since a common goal of both caching and multicast is high bandwidth efficiency, it is appropriate to first consider caching and how it fits in with our scheme. Caching involves storing popular documents at various points between the origin server and the requesting client(s) in order to minimise the number of hops that a copy of the requested file must traverse in order to reach the client. In a basic implementation of caching, a cache might be

present in the client itself and a proxy server for the client's subnetwork. In more advanced schemes such as [3] a hierarchy of caches may be used within the network.

In this paper we do not attempt to make a direct comparison between the performance of caching and our scheme since we do not view the two techniques as being mutually exclusive. Rather we intend our scheme to complement the benefits of caching by reducing the bandwidth consumed in satisfying those requests that do result in transmission of the requested file from the origin server. In fact our scheme is not just limited to the origin server but could also be implemented at cache proxy servers situated between the origin server and the clients.

The use of multicast for distribution of web documents has been proposed elsewhere in the literature. [4] presents a technique whereby outstanding requests at the server are queued in a central queue prior to commencement of service. Associated with each such request is a list of clients awaiting delivery of the requested file. Files at the server are labelled as either hot(popular) or cold(not popular) according to their access history. When a request for a so-called hot-file arrives a check is made to see if it matches on any of the requests in the central queue. If no match occurs then the new request is added to the end of the central queue. If a match does occur then the client list associated with the matching request is updated to include the new requesting client and no new entry in the central queue results. When each request reaches the head of the central queue the request is scheduled for multicast delivery if it has more than one client in its associated client list. Otherwise the request is scheduled for unicast delivery. [4] suggests that a portion of the multicast address space could be reserved for WWW multicast and each server could be allocated a contiguous block of addresses within this WWW multicast address space. [4] presents a many-to-one mapping of a server's files to the individual addresses within the server's allocated multicast address space. This is done using a hash function and has the advantage that clients with knowledge of the hash function can join the multicast group for the requested file without requiring additional information from the server. The disadvantage is that the many-to-one mapping function means that some clients may receive packets from files that they did not request which is obviously wasteful in terms of bandwidth consumption.

With the technique mentioned in [4], multicast delivery will only occur if the time between duplicate requests is less than the queuing time in the central queue. This queuing time and consequently the probability of multicast delivery will increase with overall server load. Consequently significant bandwidth savings will only occur when the server is significantly loaded. The scheme in [5] which is referred to as Asynchronous Multicast Push(AMP) facilitates significant bandwidth savings over a wider range of server loads by explicitly specifying the request accumulation period rather than allowing it to be directly controlled by the volume of data queued for transmission.

In addition to request-initiated multicast of web pages, other techniques have been proposed[1][5][7] that continuously multicast popular web pages once per some time period. This approach which [5] refers to as Continuous Multicast Push(CMP) enables best-effort UDP multicast to be used without any additional loss recovery mechanisms since any lost packets can be automatically recovered when the file is transmitted in the next cycle. AMP schemes on the other hand typically use some kind of reliable multicast protocol to handle lost packets. In addition, with CMP, clients that have advance knowledge of the multicast address that is being used can join the group without sending a request to the server. [1] and [5] present integrated server architectures that combine versions of AMP and CMP together with standard unicast delivery in order to provide each request with the most appropriate mode of delivery.

For AMP schemes, we define the maximum holding time(MHT) to be the maximum delay that may occur between the server receiving a request and passing the requested file to the output buffer queue for transmission. The higher the MHT the server is permitted to use, the greater the probability that a request matches on a duplicate before it is serviced. Since accumulated duplicate requests can be served via multicast to achieve a bandwidth saving compared to a unicast-only server, increasing the MHT will increase the bandwidth saving. Existing AMP schemes apply the same MHT to each request for a given file. Consequently in

an environment with heterogeneity among the request latency requirements of clients, the MHT for a given file is limited by the most stringent of the clients' delay constraints. This means that the MHT applied to many of the requests would be lower than necessary to satisfy the latency requirements of the requesting client. Clearly this is sub-optimal in terms of potential bandwidth saving. A more efficient approach would support heterogeneity among the MHTs of different requests for the same file. In this case each individual request MHT would reflect the requesting client's latency requirements. For example, the MHT of a business user's request might be set lower than the MHT of a casual user's request. In the next section we present a scheme that extends the basic principles of AMP to support heterogeneity of client's latency requirements together with any specific latency requirements of the requested file.

2. Asynchronous Multicast Push(AMP) with Classes of Service

Overview

In our scheme which we refer to as Asynchronous Multicast Push with Classes of Service, AMP with COS, the server divides time into epochs of duration *T*. Each client request for a file has an associated Client Designated Class(*CDC*) which is an integer indicating the maximum 'decision time' measured in epochs that the client is willing to let the server apply to the request. We define decision time as the delay between the server receiving a request and making the 'final service decision', that is determining its method of service, unicast or multicast. In the case of a unicast 'final service decision' the requested file is sent to the output buffer for transmission immediately after making the decision. In the case of a multicast final service decision the requested file is sent to the output buffer for transmssion half an epoch after making the decision. Each file at the server has an associated Server Designated Class(*SDC*) which is configured a priori to impose an upper limit on the maximum decision time for any request for the file. The SDC provides for files of a real-time nature that may need to be sent to the requesting client within certain time constraints in order to remain useful. Files with no specific real-time characteristics would simply have their SDC set to the default of ∞. When a request arrives at the server, the *SDC* of the requested file is combined with the *CDC* of the request in order to produce an "effective class"(*c*) for the request according to equation (1)

$$c = MIN(SDC, CDC) \tag{1}$$

The effective class of a request reflects the client's general latency requirements together with any specific latency requirements of the requested file, and equals the maximum delay in epochs between the server receiving a request and making the final service decision for the request. For the remainder of this paper we use the term 'class' to mean 'effective class'. Although in principle the scheme is extendable to any number of classes, in our analysis we assume just 5 classes, 0 to 4. Files with *SDC*>0 are known as multicast pool files and each request for such a file is eligible for multicast delivery provided both *CDC*>0 and the condition in equation (2) is satisfied.

$$cR_i \geq eligibility\ threshold \tag{2}$$

where R_i is the weighted average number of requests per epoch for each file, i. Any requested file not eligible for multicast is unicast immediately over the existing TCP connection that was used to carry the request. The set $\{R_i\}$ is updated once every analysis period of *Ea* epochs according to equation (3)

$$R_i = \alpha R_i + \frac{(1-\alpha)\, R_{sample}{}^i}{E_a} \quad (3)$$

where $R_{sample}{}^i$ is the number of requests received for file *i* over the previous *Ea* epochs, and α is a value between 0 and 1 which controls how fast the weighted average responds to changes in the request arrival rate(a lower value of alpha causes it to respond faster).

We refer to those requests whose mode of delivery has yet to be decided as outstanding requests. At each epoch boundary the server applies a service-allocation procedure to the group of outstanding requests to decide how each one is to be handled. During this procedure, the server determines which files with outstanding requests are to undergo multicast transmission commencing in the next epoch. We call these files 'multicast-active' files and in the next section explain the full procedure for determining them. Once the set of 'multicast-active files' has been selected the service-allocation procedure determines the handling of each outstanding request as follows:

1) If the request is for a multicast-active file then it will be serviced via multicast transmission which will commence in the next epoch.
2) If the request is not for a multicast-active file but the class of the requested file is equal to 1 then the requested file is sent to the output buffer queue immediately for unicast delivery over the existing TCP connection.
3) If neither 1) nor 2) applies then the request will not be serviced during the next epoch in which case its request class, c is changed to c-1 and the request is held until the next epoch boundary where it will once again be subjected to the above service allocation procedure.

3. Service Allocation Procedure for AMP with COS

As in [4] we assume that a portion of the multicast address space is reserved for WWW multicast and that each server using our scheme is allocated a contiguous block of addresses within this WWW multicast address space. However, the mapping between a server's files and its available multicast addresses differs significantly between our scheme and that in [4] where a static many-to-one mapping is used. By contrast, in our scheme the mapping which will be explained later in this section is one-one-one and dynamic. We assume that only N multicast addresses are available for use by the server while the number of multicast pool files is equal to *F* where *F>N*. Consequently even if duplicate requests are outstanding at an epoch boundary at which their final service decisions must be made it may not always be possible to assign them to multicast delivery since all of the available multicast addresses may be in use for the delivery of other files. We assume a limit on the number of multicast addresses used by each server to ensure that the technique will scale to a large number of servers without depletion of the available multicast address space.

We now describe in detail the service allocation procedure that AMP with COS applies at each epoch boundary to the group of outstanding requests which are recorded in the set of hashtables $\{hashCount_c\}$ where c = 1,2,3,4 denotes request class. Before the server begins its service-allocation procedure it first places a lock on the hashtables and any associated parameters in order to prevent their alteration by any new connections that it may accept during the service-allocation process. The necessary steps in the service allocation procedure are now as follows. First the server divides the set of *N* multicast addresses into 2 sets, {*freeAddress*} and {*busyAddress*} containing *Nfree* and (*N-Nfree*) addresses respectively. {*freeAddress*} comprises all addresses from the server's *N* multicast addresses that are currently free, that is no longer busy multicasting initial transmissions that were started in previous epoch(s). The service-allocation procedure can determine the current status of each address, that is busy or free, by inspecting a so-called *busy flag* that is associated with each address.

Next the server must re-allocate each of the *Nfree* free addresses to a separate file. The *Nfree* files to be used for this purpose are obtained as follows. Firstly, an initial set of candidate files, {*fcandidate*} is chosen according to selection criteria that vary according to

which mode of AMP with COS is being used. We define two modes of operation, Holdback and No-Holdback mode. With No-Holdback mode, {*fcandidate*} contains all files whose total number of outstanding requests is greater than one. With Holdback mode {*fcandidate*} is the same as with No-Holdback mode except all files whose number of outstanding class 1 requests are equal to 0 are excluded. Once the set of files {*fcandidate*} has been obtained in accordance with the particular mode of operation in use, the next step is to add or remove files from {fcandidate} in order to obtain a set of files {*felected*} containing *Nfree* members. If the number of files in {*fcandidate*} is denoted as *Ncandidate* then the set {*felected*} is obtained as follows.

If *Ncandidate=Nfree:*

In this case {*felected*}={*fcandidate*}.

If *Ncandidate<Nfree:*

An additional (*Nfree-Ncandidate*) files must be added to {*fcandidate*}. These are obtained from those files with the highest values of *Ri* not currently in {*fcandidate*} but which were allocated multicast addresses at the previous epoch boundary.

If *Ncandidate>Nfree:*

For each file in {*fcandidate*} a measure of its suitability for multicast, known as its multicast suitability factor, *Sm* is calculated. This is defined as the reduction in the number of bytes that will be transmitted if the file's multiple outstanding requests are served by a single multicast rather than a separate unicast transmission for each outstanding request. This definition of *Sm* can be written more concisely in the form of equation (4) where count is the number of outstanding requests for the file.

$$Sm = (count - 1)\, filesize \tag{4}$$

{felected} is now formed from the *Nfree* most desirable files, that is those with the highest values of *Sm*, from the set *{fcandidate}*.

To illustrate the benefits of our scheme our definition of Sm is concerned solely with the reduction in the number of transmitted bytes on the server output link. However, AMP with COS is also suited to more elaborate definitions of Sm that take into account other factors regarding the cost of multicast transmission. This is outside the scope of this paper but the interested reader can consult [9] for some examples.

It is now necessary to assign each of the files contained in {*felected*} to a separate multicast address from the *Nfree* addresses contained in *{freeAddress}*. In doing this the following rules are applied.

1) Any file in *{felected}* which, at the previous epoch boundary, was assigned a multicast address which is now contained in {freeAddress} is assigned the same address at this epoch boundary. This rule prevents unnecessary address allocation flapping.
2) The assignment of the remaining multicast addresses in *{freeAddress}* to the remaining files in *{felected}* is arbitrary.

The next step is to establish which of the free multicast addresses will actually be used for transmission during the next epoch. Any free multicast address assigned to a file with fewer than two outstanding requests will not be used during the next epoch since doing so would not yield any resource saving compared to conventional unicast delivery. All remaining free multicast addresses will be used during the next epoch and we refer to all such addresses as 'multicast-active'. For each free multicast address the server indicates whether or not it is multicast-active by setting its associated multicast-active flag, *mcastActive* accordingly.

For each outstanding request for a file that has been designated a free multicast-active address the server sends a HTTP 302 response (redirect) over the existing HTTP/TCP connection. Included in this response will be the file's allocated multicast address together

with the UDP destination port number to be used for the multicast. The server then removes the request from the appropriate hashtable in the set {*hashCountc*}.

Any remaining outstanding class 1 requests are now serviced over the existing HTTP/TCP connections and removed from $hashCount_1$, the hashtable for outstanding class 1 requests. Next each outstanding request of class n(n>1) is changed to class n-1 and the hashtables {$hashCount_c$} updated accordingly. The server then removes its lock on the hashtables and any requests that were received during the service allocation process will now be added to the hashtables which will subsequently be updated every time a new request arrives. The actual multicasting of the multicast-active files is not done immediately after the service-allocation process. Instead the server first waits until the mid-epoch point in order to allow sufficient time for all affected clients to receive the necessary information and join the appropriate multicast group.

4. Implementation

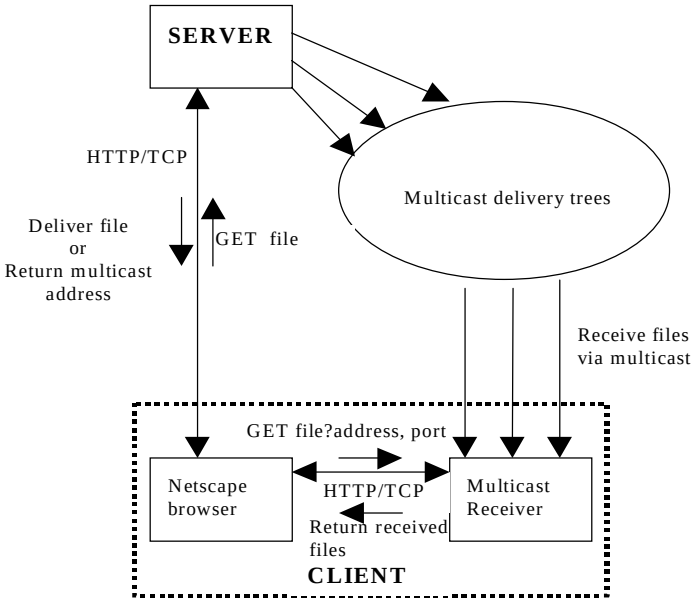

Figure 1: Implementation of Our Scheme

Our scheme has been implemented for a Netscape 3 browser as shown in
Figure 1 where the server and multicast receiver have been written in the Java language. For each request that the server chooses to serve via multicast, a user-defined MIME type contained within the HTTP 302 redirect response serves as the trigger to start a multicast receiver application, to which are passed details of the multicast group to be used for transmission of the requested file. The response also contains the HTTP entity-header relating to the requested file and including fields regarding the encoding and length of the file. Because the response contains all of the necessary HTTP-specific information and is carried over a reliable TCP connection, no HTTP-specific information whatsoever needs to be included in the subsequently multicasted packets. With regard to the multicast, a lightweight protocol is used above the UDP-layer in order to indicate the name of the file and the sequence number of the packet. The sequence number is set to 1 for the first packet sent and is incremented for each subsequent packet sent. The sequence numbers are used at the receiver to correctly order the received packets and to detect packet loss which is indicated by gaps in the sequence number space. The receiver obtains complete knowledge of the full sequence number space from the user-defined MIME type contained in the response which includes an integer indicating the number of packets that will comprise the multicast of the requested file. As soon as a receiver has obtained all of the requested file it immediately leaves the multicast group. At present we have not yet added any specific loss recovery

mechanisms to the implementation. This represents work in progress and proposals for how we intend doing this are discussed in section 6.

5. Simulation

In the longer term we intend to analyse the performance of our server in a real network environment. However, in order to assess some of the characteristics of our server at a much earlier stage we have built a simulator written in the Java language and incorporating the same service allocation code used by our server while accepting existing HTTP server logs as input. The server log used as input to our simulations represents all accesses over the 12 hour period 10am-10pm, March 7th 1998 of a British Telecom(BT) server at Ipswich, England. The level of traffic reflected in this log is insufficient to notice any significant resource savings using our scheme with reasonable values for the epoch period. However, the log is still useful in terms of its access pattern. [2] examined the access pattern of 6 WWW servers and noted many invariants among them. For example, each of these 6 servers displays non-uniform referencing behaviour, that is concentration of requests among a small percentage of the requested files, the so-called hot files. For each of the 6 servers, 10% of the distinct files were responsible for 85-95% of all requests while less than 3% of all requests were for distinct files. Other invariants that [2] noted among the set of 6 server logs examined include Pareto[6] file size distribution and for files accessed more than once, inter-reference times that were independent and exponentially distributed. Moreover [2] notes that the set of 6 servers examined cover different orders of magnitude of server activity. Consequently, we believe that simulation results obtained from the BT server log will remain meaningful even if the traffic level is scaled up by an order of magnitude, which is necessary in order to illustrate any potential resource savings of our scheme. If we assume that a real implementation of our scheme may have an epoch duration of say 3 seconds, then multiplying the traffic level by 10 times for this epoch duration will result in the same rate of requests per epoch as keeping the traffic rate the same but multiplying the epoch duration by 10. This is the approach taken in our simulations, that is, we use the unaltered BT server log but assume an epoch duration of 30 seconds.

The simulation assumes lossless transmission for both unicast and multicast. We discuss the impact of packet loss on our scheme in section 6. In addition the simulation assumes that each multicast transmission has been completed by the first epoch boundary after commencement of the multicast.

The simulation analyses each request in the log file and records the total number of bytes, $bytesTransmitted_{log}$ that were transferred by the BT server[1]. This is actually less than the sum of the file sizes of all valid requests since sometimes the user may terminate the connection before the requested file has completed transmission. We use $bytesTransmitted_{log}$ as the benchmark against which we measure the bandwidth resource saving of our system on the server output link.

If we denote the total number of bytes transmitted by the server using our scheme as $bytesTransmitted_{ourScheme}$ then the percentage fewer bytes, *Bsave* that the server will transmit with our scheme compared to a traditional unicast-only server can be written as equation (5)

$$Bsave = \frac{100(bytesTransmitted_{\log} - bytesTransmitted_{ourScheme})}{bytesTransmitted_{\log}} \quad (5)$$

Each request line in the server log with both a GET command and a 200 response code[2], and which is not a CGI request is passed to our service allocation module. For all other request

[1] In the case of complete transfer of a file, the value in the server log represents the number of bytes transferred in a loss-free environment, that is it does not include the additional bytes transmitted due to packet loss

lines the number of bytes transferred is obtained from the server log and added to $bytesTransmitted_{ourScheme}$. The simulation assumes that each request passed to the service allocation module experiences full transmission of the requested file via either unicast or multicast and consequently the full file size is added to $bytesTransmitted_{ourScheme}$ at each such transmission.

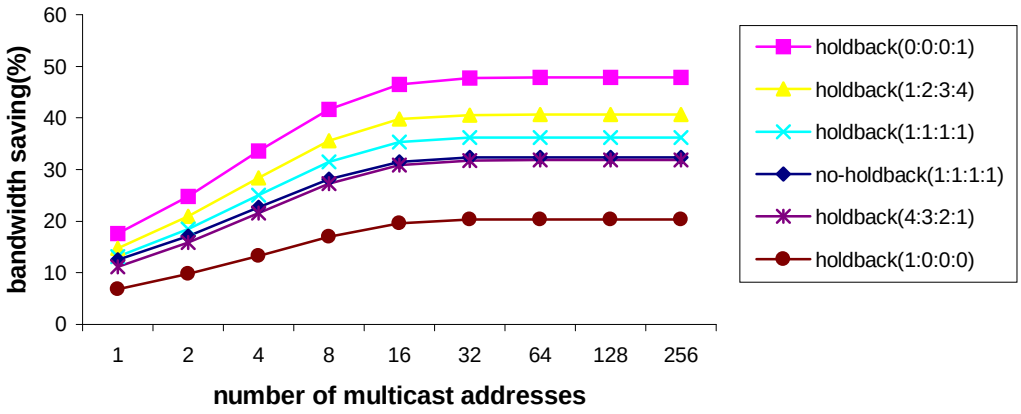

Figure 2: Bandwidth saving of holdback and no-holdback mode(M=0)

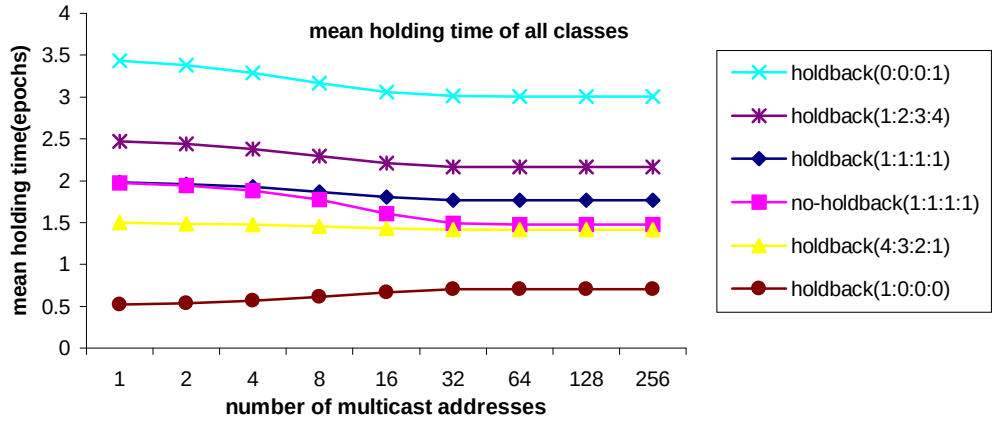

Figure 3: Mean holding time(all classes) of holdback and no-holdback mode(M=0)

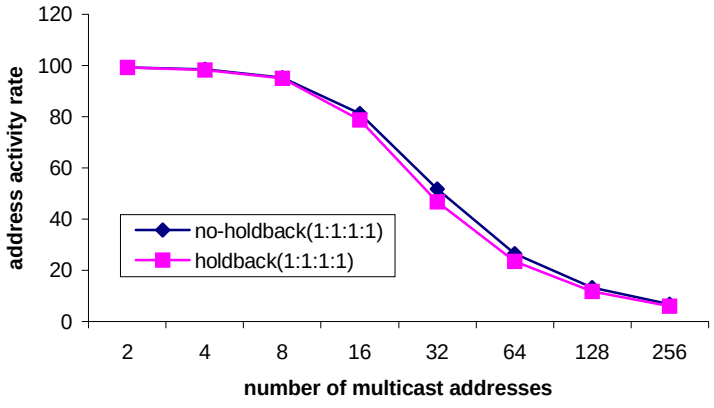

Figure 4: Multicast activity rate of holdback and non-holdback mode(M=0)

[2] A 200 response code indicates that the request for a file resulted in its transmission.

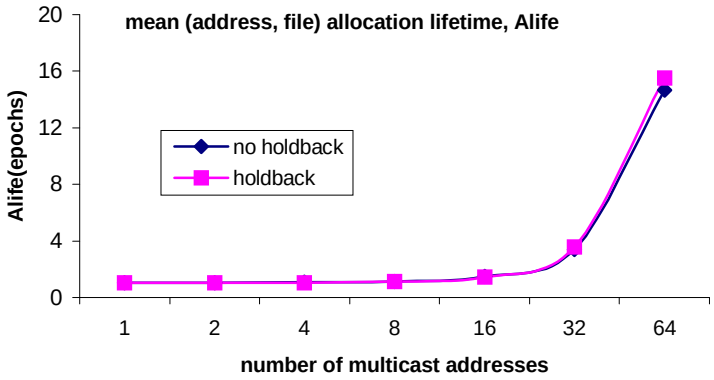

Figure 5: mean (address, file) allocation lifetime of holdback and no-holdback mode (M=0)

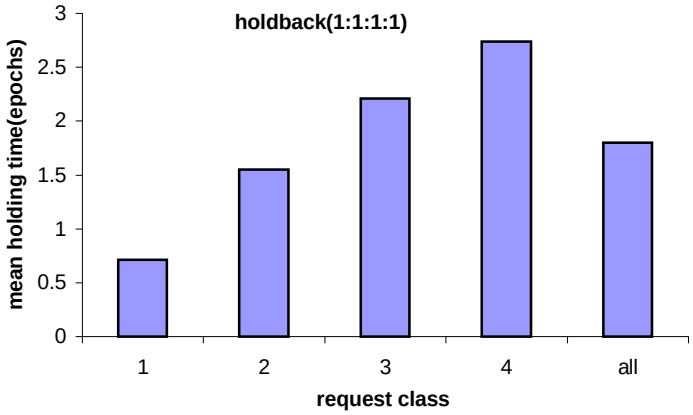

Figure 6: Mean holding time of each class for holdback(1:1:1:1) (M=0, N=16)

Figure 7: % of each request class served via multicast with holdback(1:1:1:1) (M=0, N=16)

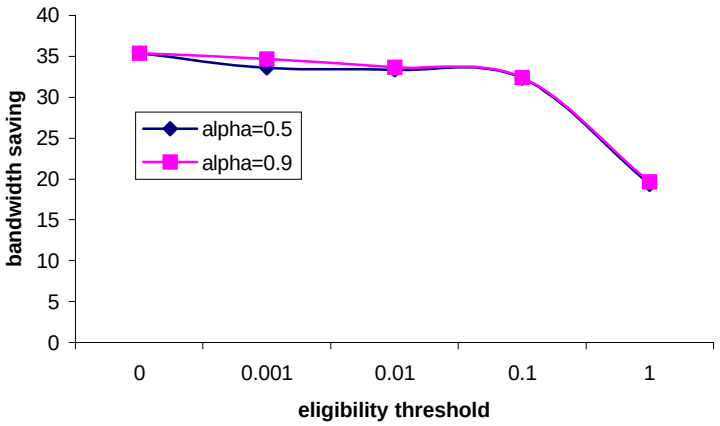

Figure 8: Effect of multicast eligibility threshold(M) on bandwidth saving of holdback(1:1:1:1) for N=16.

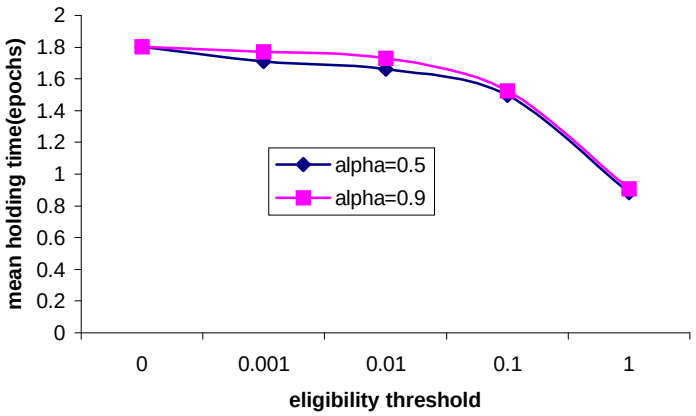

Figure 9: Effect of multicast eligibility threshold(M) on mean holding time of holdback(1:1:1:1) for N=16.

The results of our simulations are shown in Figures 2-9 where we assume that each request has a non-zero class. We use the notation *mode(share1:share2:share3:share4)* to denote the mode used and the expected proportion of requests taken up by each request class according to their probabilities. For example holdback(1:1:1:1) indicates that a given request could be any class between 1 and 4 inclusive with an equal probability. Holdback(1:0:0:0) indicates that all requests are of class 1. We assume an epoch period of 30 seconds in all our simulations. In each of Figures 2-5 we include the simulation results for the holdback(1:1:1:1) and no-holdback(1:1:1:1) in order to allow a direct comparison between the two different modes of our scheme in an environment with heterogeneity among client's latency requirements. The multicast eligibility threshold, M is set to 0 which essentially means that all requests passed to the service allocation module will be multicast eligible in accordance with equation (2).

As can be seen in Figure 3, holdback(1:1:1:1) gives a higher mean holding time than no-holdback(1:1:1:1) since it delays the multicast of files for as long as possible. A consequence of this is a better bandwidth saving than no-holdback since the holdback variant satisfies a higher number of requests per multicast. This improvement in bandwidth saving is illustrated in Figure 2.

Figure 4 shows what we call the multicast activity rate which is the percentage of potential multicast opportunities at which a multicast transmission occurs. There are *N* potential multicast opportunities per epoch since up to one multicast may occur on each of the *N* multicast addresses every epoch. As expected, the multicast activity rate of the no-holdback

scheme is greater than that of the holdback scheme although the difference is slight. The difference is due to the fact that with no-holdback multiple requests for the same file at an epoch boundary will always lead to a multicast in the next epoch providing a free address is available. By contrast with holdback such a situation will only lead to a multicast provided at least one of the requests is of class 1.

Figure 5 shows the address allocation lifetime of the two variants, that is the average number of epochs that each (file,multicast address) association lasts for. As can be seen the holdback variant displays a slightly more stable address allocation, that is a higher average value of address allocation lifetime. The more stable address allocation of holdback is due to a lower multicast activity rate which means a lower probability of a file with an assigned multicast address losing its address allocation at a given epoch boundary.

Although there is little to choose between the holdback and no-holdback modes of our scheme, we prefer the holdback mode on the grounds that it offers better bandwidth saving and more stable address allocation while we do not regard the extra delay incurred as significant. For N=16 no-holdback(1:1:1:1) displayed a mean delay(all classes) of 1.6 epochs and a bandwidth saving of 31.5% while holdback(1;1:1:1) displayed a mean delay(all classes) of 1.80 epochs and a bandwidth saving of 35.4%.

In Figure 2 and Figure 3, in addition to holdback(1:1:1:1) and no-holdback(1:1:1:1) we also include results for the holdback scheme with different proportions of each class of request. As expected the greater the proportion of requests that are of a higher class(class 4 being the highest) the greater the overall bandwidth saving and mean holding time. Holdback(1:0:0:0) is equivalent to AMP with no classes of service and a request accumulation period of 30 seconds which is the maximum value capable of satisfying the latency requirements of all clients in our heterogeneous environment. Holdback(1:0:0:0) displays a much lower bandwidth saving than holdback(1:1:1:1) thereby illustrating the primary benefit of our scheme.

Figure 6 illustrates the service differentiation given to each request class for holdback(1:1:1:1) with 16 multicast addresses. There is a near-linear relationship between mean holding time and request class, that is the mean holding time of request class n is approximately n times that of request class 1. Figure 7 illustrates the % of requests that experience multicast delivery for each of the 4 request classes. As expected, higher request classes experience a higher % of requests undergoing multicast delivery.

It is now worth highlighting 2 important observations regarding the results. First, a near optimal bandwidth saving is achieved with a relatively low number of multicast addresses, 16. Second, substantially more multicast addresses than this are required to achieve relatively stable (file, address) allocations. For example, in the case of holdback(1:1:1:1) with 16 addresses the average lifetime of each (file, address) allocation is only 1.6 epochs whereas with 64 addresses it is almost 16. A high average lifetime for address allocations may be desirable when we consider the issue of reliability and retransmissions and is discussed in more detail in section 6.

Next we investigate the effectiveness of the multicast eligibility algorithm of equation (2), the aim of which is minimise the mean holding time of requests without significantly reducing the bandwidth saving. Each request that arrives at the service allocation module is first subjected to the multicast eligibility test and if unsuccessful (i.e. deemed to be a request for a less popular file) is sent immediately to the output buffer queue for unicast transmission over the existing TCP connection. Figures 8 and 9 analyse the effectiveness of this algorithm. As can be seen, for the particular access pattern used, the mean holding time cannot be significantly reduced without also significantly reducing the bandwidth saving. Consequently, the benefits of categorising files as either hot or cold as suggested in [1][4] may not be that useful in practice if our scheme is used in the absence of any additional multicast delivery techniques. This categorisation is essential in [1][4] to determine when to switch delivery of a file between the different multicast delivery schemes in the integrated server architecture.

6. Implementation Issues

Reliability

The simulation and discussions presented in the previous sections of this paper assumed a loss-free environment. In reality however, some degree of loss is inevitable and since clients will usually require complete reception of the documents that they request, some means of providing a reliable service above the UDP multicast layer needs to be included. Our simulation assumed loss-free unicast as well as loss-free multicast and so we do not expect the bandwidth savings to differ significantly when packet loss is taken into account. Moreover the relative performance of AMP with classes of service compared to single class AMP should change very little when packet loss is taken into account if we assume that they both use the same recovery mechanisms for packets that were lost during multicast.

One way in which loss recovery of multicast transmissions might be achieved is as follows. When a receiver detected packet loss it would send a retransmission request over the existing TCP connection to the server. The retransmission request would indicate the name of the incompletely received file along with the sequence numbers of missing packets and the multicast address to which the receiver was currently joined and over which it had expected to receive the missing packets. Upon receiving this request the sender would first check if the (address, file) allocation indicated in the re-transmission request was still in existence. If it wasn't then the sender would not service the retransmission request using the multicast address indicated in the request since this might deliver it to some receivers that had not even requested the file. Instead, the contents of the missing packets of the file would be unicast to the requesting receiver over the existing TCP connection. If the (address, file) allocation did still exist then we refer to the retransmission request as being 're-multicast-eligible'. For each such re-multicast-eligible retransmission request the sender would first check a list of sequence numbers of packets that were re-multicast over the current epoch for the (address, file) allocation indicated in the re-transmission request. Action would only be taken with regard to retransmssion requests for missing packets that were not in this list in which case they would be re-multicast immediately over the indicated address and their sequence numbers would be added to the list. At each epoch boundary any such retransmission lists would be removed from the server.

An alternative to the above loss recovery procedure would be to delay handling any re-multicast-eligible requests until the next epoch boundary. Then, prior to the service allocation process for first-time requests, the server would handle all stored retransmission requests as follows. Any packet that needed to be retransmitted to more than one receiver would be sent via multicast while the contents of any packet that needed to be retransmitted to only a single receiver would be delivered over the existing TCP connection to that receiver. Yet another alternative would be to serve all retransmission requests immediately over the existing TCP connections to the requesting clients.

With all of the alternatives mentioned above a receiver would judge a packet to have been lost if it had not been received within some timeout period or before the receiver started to receive packets of another file on the same multicast address. This latter scenario would indicate that the incompletely received file had been fully transmitted by the sender but the file had lost its address allocation at a subsequent epoch boundary and the address was then being used to multicast another file.

The relative merits of each of the alternative recovery methods discussed is a subject for further study. Other areas that need further work include the receiver's determination of timeout period to use for detecting multicast packet loss.

Multicast Join-time

Another issue that we need to consider is the delay between the server assigning multicast delivery to a requested file, and all intended recipients joining the multicast tree. This delay is given by the sum of two components. First, the delay incurred in sending the HTTP 302 response back to a receiver and second the delay incurred in the receiver joining the multicast

tree in accordance with the multicast protocol in use. If the sum of these two components is greater than 0.5 epochs then the receiver may not have joined the multicast tree by the time the sender begins to multicast the requested file and so may not receive some (or all) of the file.

In theory the delay incurred by the HTTP 302 response could be minimised by setting the delay Type Of Service bit in the IPv4 packet header in order to convey the low delay requirement to intermediate nodes who might then take this into account when handling the packet. In practice however these bits are ignored by the vast majority of network nodes. Another alternative is to reserve resources in the network nodes between the server and clients along the path to be taken by the prospective HTTP 302 redirects. A suitable protocol for this would be the lightweight sender-based reservation protocol presented in [10]. Once the HTTP 302 packet passed through each node the protocol presented in [10] could be used to remove the reservation. However, like the use of the Type of Service bits, the reservation approach is dependent upon support by the intermediate network nodes.

In cases where the client fails to join the multicast group in time for all or some of the transmission of the requested file then as far as the client is concerned the outcome is no different to that experienced by packet loss due to congestion in the network and can therefore be handled in the same way.

Handling of duplicate requests from the same client

Currently, the server in our scheme stores details of the requesting client associated with each outstanding request and can therefore detect when multiple requests for the same file have arrived from the same client. The two most likely causes of such a situation are as follows.

1) The user has generated multiple requests by excessive clicking on a button, for example the browser reload button or a JavaScript button.
2) The multiple requests have been generated by a web page that references multiple copies of the same file. For example, a single page of HTML might contain multiple copies of the same gif graphic.

Potentially 1) above can lead to the server registering an artificially high request count for a file which can artificially increase its chances of being multicast. However a simple policy such as the server discarding any duplicate requests from the same client would not work since the duplicates could be valid as in 2) above. One approach would be for the server to handle all requests without considering the identity of the client in which case the client would be responsible for ensuring that invalid duplicate requests were not generated.

In the simulations of section 5 the server did not consider the identity of the requesting clients in any way. In addition the full set of GET HTTP 200 entries from the HTTP server log were presented to the server. Consequently, the bandwidth savings presented are higher than if suppression of invalid duplicates from the same client had been implemented. To assess this potential effect, we ran the simulations again with the modification that any arriving request that matched on an outstanding request from the same client was treated as being not multicast-eligible. This represents an extreme check since it would result in the valid duplicates in case 2) above being registered as not multicast-eligible also. For the case of holdback(1:1:1:1) with N=16 this resulted in a bandwidth saving of 28.6% compared to the figure of 35.4% obtained previously. The approach mentioned above in which sole responsibility for invalid duplicate request suppression was assigned to the client would therefore yield a bandwidth saving somewhere in between these two figures. In this section we have touched upon some of the issues surrounding the handling of duplicate requests from the same client but acknowledge that further investigation is required to determine the most appropriate method of handling them.

7. Summary

In this paper we have extended the basic scheme of Asynchronous Multicast Push(AMP) to accommodate service differentiation among clients with respect to the maximum holding time experienced by their requests. We have compared through simulation the performance of our

scheme, known as AMP with COS, with that of single class AMP. In environments where heterogeneity of latency requirements exists among clients, AMP with COS exhibits a superior bandwidth saving to that of single class AMP. This is because AMP with COS is able to apply a maximum holding time to each request that is limited only by the latency requirements of that particular client. By contrast with single class AMP the same maximum holding time is applied to each request and so may be unnessarily low for less-delay sensitvie requests.

We have also discussed some implementation issues and how they might be resolved, particularly with respect to reliability and multicast join times. Other areas that need to be addressed and which represent work in progress include multicast congestion control and billing. Billing is required in order to exert some backpressure on users with regard to the use of lower latency request classes. The more clients that specify a lower latency request class the lower the resultant bandwidth saving and so it is reasonable to expect the network provider to be compensated in some way for servicing lower latency classes of requests.

8. Acknowledgements

The authors would like to express their thanks to British Telecom Labs, Ipswich, England for supporting this work and for providing the web server log. In addition we would like to thank Alan O'Neill and Martin Tatham, both of BT Labs for many beneficial discussions during the course of this work.

9. References

[1] K. Almeroth, M. Ammar, Z. Fei. Scalable Delivery of Web Pages Using Cyclic Best-Effort (UDP) Multicast, Networking and Telecommunications Group Georgia Institute of Technology Atlanta, Georgia. INFOCOM ' 98.

[2] M. Arlitt and C. Williamson. Web Server Workload Characterization: The Search for Invariants, proceedings of the 1996 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, Philadelphia, PA, May 23-26, 1996.

[3] A. Chankhunthod et al. A Hierarchical Internet Object Cache. Proc. 1996 USENIX Technical Conference, San Diego, CA, Jan 1996.

[4] R. Clark, M. Ammar. Providing Scalable Web Service Using Multicast Delivery, Proceedings of 2nd International Workshop on Services in Distributed and Networked Environments (SDNE 95), June, 1995.

[5] J Nonnenmacher and E. W. Biersack. Asynchronous multicast push: Amp. In Proc. of ICCC' 97 International Conference on Computer Communications, Cannes, France, November 1997. http://www.eurecom.fr/~nonnen/

[6] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling, Proceedings of ACM SIGCOMM ' 94, London, England, pp 257-268, August 1994.

[7] P. Rodriguez, E. Biersack. Continuous Multicast distribution of web documents over the Internet, Institut Eurocom, France, Nov 1997.

[8] Comparison of reliable multicast protocols web page, http://www.tascnets.com/mist/doc/mcpCompare.html

[9] S. Herzog, S. Schenker and D. Estrin. Sharing the "Cost" of Multicast Trees: An Axiomatic Analysis. ACM SIGCOMM '95, August 1995, Cambridge, Massachusetts, USA.

[10] P. White and J. Crowcroft. A dynamic sender-initiated reservation protocol for the Internet, accepted for High Performance Networking '98 conference.

[11] T. Lee et al. Hypertext Transfer Protocol – HTTP/1.0. RFC1945, May 1996.

[12] R. Fielding et al. Hypertext Transfer Protocol – HTTP 1.1, Request for Comments, RFC2068, January 1997.