Bulk Multicast Transport Protocol

Robert Morris
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138 USA

Abstract

BMTP offers rate controlled multicast with reliability, high throughput, and support for large numbers of receivers. A multicast sender needs feedback from receivers to recover from errors and to choose an appropriate send rate, but must avoid being overwhelmed as the number of receivers grows. BMTP does this by keeping the rate at which each receiver sends feedback inversely proportional to a running estimate of the number of receivers. BMTP bases its send rate on the minimum of the receive rates observed by the receivers, causing the sender to slow down in the face of packet loss or competing traffic, and to speed up when there is spare network capacity. BMTP's NAK-based retransmission rarely sends any data more than twice, a substantial improvement over iterated unicast. Rabin's Information Dispersal Algorithm can reduce this re-send rate as close as desired to the underlying loss rate of the network. Simulations with 1000 receivers substantiate these claims.

1. Introduction

Lack of a suitable multicast transport protocol has prevented bulk data transfer applications from using the Internet multicast facilities. Netnews, mailing lists, and mass software updates, for instance, all distribute large amounts of identical information to many receivers. Currently, however, these applications use many one-to-one (or unicast) transfers over the Internet, sending the same data over the network many times. The idea of multicasting the data just once is old [11], but no existing multicast transport protocol provides the reliability and scalability required by these applications. BMTP addresses this need.

BMTP solves only the reliable high-throughput multicast problem; it does not cater to all multicast applications. These are its design goals:

- Allow a sender to transmit a long stream of data to many receivers over the Internet.
- Assure reliability by re-sending lost data.
- Send as fast as the network allows without overloading it.
- Limit the sender's complexity and the load placed on it, as it could be a bottleneck.
- Allow receivers to come and go at any time.
- Adapt to widely varying network capacities along the paths to different receivers.
- Do not require changes to the underlying Internet.
- Scale to thousands of receivers over a wide area.

BMTP's goals do not include low delay, coordination among multiple senders, and consistent message ordering. Interactive applications might work better with multicast transport protocols that provide low delay [2,5]. Protocols that guarantee consistency among multiple participants [1] are better suited than BMTP to distributed computations. Applications with a limited number of receivers or limited network complexity may be more efficient with protocols [3,10,12] that take advantage of such limits. BMTP's intended applications fall outside these classes, and BMTP takes advantage of the their tolerance of delay to provide scaling and adaptability to a wide range of network conditions.

The next section describes BMTP's relationship to existing work. Section 3 presents BMTP's design, and sections 4 and 5 describe further details. Section 6 uses simulation results to show how well BMTP works across a range of error rates and numbers of receivers. Section 7 presents the improvements in error recovery made possible by the Information Dispersal Algorithm.

2. Related Work

BMTP relies on IP Multicast [4] to deliver packets to receivers. An IP Multicast application can send packets to one of a number of special multicast IP addresses, or ask to receive all packets sent to such an address. IP Multicast routers keep track of all the receivers listening to each address; the receivers for each address are called a group.

IP Multicast delivers packets along a tree rooted at the sender with a leaf at each group member. Figure 1 shows

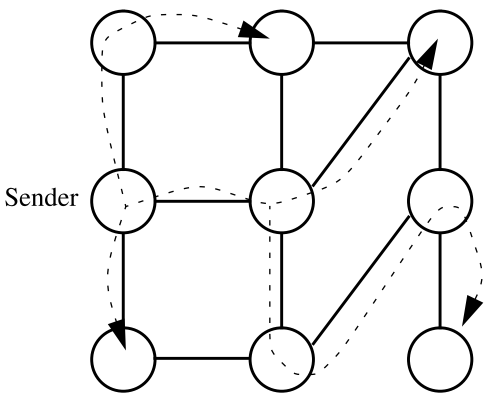

Figure 1: IP Multicast Distribution Tree Example

an example of a multicast delivery tree layered on top of an internet. This tree provides higher efficiency than separate unicasts to each receiver. No one entity, not even the sender, needs to keep track of the entire group membership; this allows IP Multicast groups to scale to thousands of members. Like the Internet, IP Multicast provides unreliable best-effort delivery. The distribution tree often causes a single packet loss near the sender to affect more receivers than a loss near a leaf.

A number of previous systems have addressed reliable multicast. One approach is to extend ideas in reliable unicast protocols such as TCP [8]. For instance, HPBFTP [3] and SCE [10] require that all receivers acknowledge the data as they are sent. The sender knows the identities of the receivers, so it can re-send any data not acknowledged by all receivers within one round-trip time of the initial transmission. Further, the acknowledgments (ACKs) allow the sender to use sliding window flow control to avoid overloading the network or receivers. While these are important properties, such a scheme cannot scale well: the sender cannot easily track a large and changing list of receivers, nor avoid being swamped by a large number of acknowledgments for each packet sent.

MTP [1] fixes some of these problems, though it focuses on ordering messages from many senders rather than bulk transfer. An MTP receiver does not acknowledge every data packet. Instead, it returns a negative acknowledgment (NAK) packet to the sender when it notices that it has failed to receive a data packet. Thus the sender receives NAK packets in proportion to the number of receivers times the loss rate, not times the send rate. However, if multiple receivers miss the same data packet, they may flood the sender with NAKs. This might happen constantly in a large network. In addition, because MTP does not use ACKs, it cannot use a sliding window for flow control. Instead the sender must pace itself with rate control. However, the protocol specification does not mention a general method for adapting this rate to changing conditions in a large network.

SRM [5] provides a solution to the problem of multiple NAKs for the same lost packet. A receiver delays transmission of a NAK packet in a manner designed to ensure that the receiver closest to the point where the packet was lost sends a NAK first. The receiver multicasts the NAK to all the other receivers as well as the sender, so that any participant with a copy of the lost data can re-send it. Again, participants that receive a NAK delay re-sending the data in a way that makes it likely that only the closest participant will re-send. Thus, in many situations, a lost data packet will result in only one NAK and re-sent data packet.

SRM is well suited to interactive traffic but has some drawbacks for bulk transfers. If retransmission is to be efficient, all the receivers must remember all recently transmitted data. This may not be reasonable for large amounts of data. SRM also includes no adaptive flow control mechanism to avoid overloading the network. Since the number of NAKs generated is only loosely related to the number of packet loss events, it might be hard for the sender to gather enough information to adjust its rate. For instance, suppose there are 1000 receivers, and the sender sees one NAK for every two data packets it sends. This could mean that all the receivers are seeing 50% packet loss, or that 999 of them are receiving perfectly and just one has a problem. The sender might wish to drastically reduce its rate in the first case, and keep it the same in the second, but has no way in SRM to distinguish the two.

The IVS video multicast system [2], while not reliable, does have adaptive rate control. Each receiver keeps track of the fraction of packets it failed to receive (presumably due to network overload). The sender decides what fraction of receivers are overloaded by probing a small subset of them, chosen randomly. The sender decreases its send rate by half if more than 1.4% of the receivers are losing packets. It increases its send rate slowly if none of the receivers report loss. BMTP, as well as providing reliability, allows more accurate flow control by giving the sender more specific information about receive rates.

The RMTP system [7] uses ACKs for reliability and a window for flow control. It avoids flooding the sender with ACKs by combining them as they flow back up the multicast tree. The combining nodes may re-send missing data if they have it stored locally. RMTP requires changes to multicast routers to allow such retransmissions to be sent only to the sub-tree below the combining node.

RMTP lacks a mechanism to adapt the number and placement of combining nodes to changes in the number and location of receivers. Finally, RMTP's ACK combining scheme discards information that the sender needs to set its window size, and thus its transmission rate. An RMTP sender decreases its window when ACKs indicate that too many packets are being lost. Combining nodes that buffer and re-send lost data will not report losses to the sender unless they run out of buffer space. Combining nodes that merely combine ACKs will tend to report every packet as lost if there are many receivers. In either case the information available to the sender may be misleading.

3. BMTP Architecture

BMTP bases its error and flow control on status packets sent from receivers to the sender. Each status packet contains the rate at which the receiver recently received data. This receive rate is the number of bytes per time successfully delivered to the receiver by the network; a receiver will report a rate lower than the send rate if the network is slow, if the network drops packets, or if the receiver cannot keep up with the sender. A status packet may optionally contain a NAK as well: the sequence number of a missing packet. Receivers unicast status packets to the sender at times described below.

The BMTP sender arranges to receive only one status packet per data packet it sends as follows. It maintains a number N, an estimate of the number of receivers. The sender includes N in each data packet it sends. Whenever a receiver gets a data packet, it sends back a status packets with probability 1/N. The sender measures the ratio of status packets it receives to data packets it has sent. If this ratio is higher than one, the estimate N is too low; and if lower, N is too high. The sender periodically divides N by the ratio to adapt to any changes in the number of receivers. Thus, no matter how many receivers miss a given data packet, the sender will not have to deal with a high rate of incoming status packets.

The sender maintains an estimate of a reasonable transmit rate, and the minimum receive rate recently reported in a status packet. When it receives a status packet which indicates a receive rate substantially less than the previous minimum, it immediately decreases the transmit rate to the indicated receive rate. The sender periodically increases the transmit rate to be slightly higher than the minimum receive rate recently reported. This algorithm will increase the send rate to just over the capacity of the slowest part of the system, whether that is a slow network link or a slow CPU. It knows when it has reached the bottleneck capacity because reported receive rates stop increasing. If network capacity decreases, or if packets are lost, one or more receivers will observe a decreased receive rate. These receivers will notify the sender of the lower rate, and the sender will send at that rate but no slower. The algorithm intentionally overloads the network slightly in order to detect spare capacity. As described below, BMTP can recover from the resulting packet losses with little loss in efficiency. This may seem a greedy policy, but it is analogous to TCP's congestion window increase algorithm [6].

Receivers detect missing data packets by looking for gaps in the packet sequence numbers. Each time the receiver sends a status packet, it can ask the sender to re-send one data packet. The receiver will only ask for a packet that has been missing for a while, in case the network re-orders packets and to allow the sender some flexibility in the order in which it sends data. The receiver will not ask for the same packet twice in quick succession, to give the sender time to respond to the earlier request.

The sender may need to make decisions about which packets to re-send; even though it will receive at most one NAK per data packet it sends, it may wish to limit the bandwidth used by re-transmissions. Note that the transmit rate described above applies to all packets the sender sends, including re-transmissions. The sender could record all NAK requests and service them in some order. However, this is not necessary. The sender is willing to use some fraction of its transmit rate for re-transmissions. As each NAK arrives, the sender responds to it if recent re-sends have used less than that fraction, and discards the NAK otherwise. If there are many NAKs, the sender will use up the allowed fraction with responses, and remembering NAKs won't cause it to re-send any faster. If there are only a few NAKs, the sender will be willing to respond to all of them. Receivers re-send NAKs periodically in case the sender ignores them or the network drops them.

As described, this scheme has trouble with very non-uniform packet loss. Suppose one receiver misses many packets, but 999 others have perfect reception. The one receiver can request only one re-transmission for every 1000 packets sent, which is only enough to recover from a tiny loss rate of 0.1 percent. Few of the status packets the sender gets will have a NAK in them; the others will be wasted from the point of view of error correction. BMTP handles non-uniform loss by weighting the receiver counting mechanism described above by the number of packets each receiver is missing. Call this number M for a particular receiver. Receivers cap M at a global constant K to avoid one receiver hogging all the NAK bandwidth. The receiver places MAX(MIN(M, K), 1) in status packets. The sender sends out its estimate N of the sum of these numbers in data packets. The receiver returns a status packet with probability MAX(MIN(M, K), 1) / N in response to each data packet. This scheme

Data Packet Field	Bytes
UDP/IP Header	28
# of Receivers	4
Measurement Interval	4
Packet Seq #	8
Data Seq #	8
Data Payload	512

Status Packet Field	Bytes
UDP/IP Header	28
Measurement Interval	4
Packets Received	8
Packets Lost	8
NAK Data Seq #	8

Figure 2: Packet Formats

allows less reliable receivers to send more status packets and thus more NAKs. Note that this means the bandwidth samples in status packets are dominated by the receivers with high loss rates, which is appropriate since they will tend to have the lowest receive rates.

4. Timing Details

The discussion above omitted the details of timing intervals. BMTP defines a constant control interval, which must be at least the maximum possible round trip time including queuing delay. The experimental implementation uses ten seconds. Control is exerted once per control interval. This includes the sender increasing the transmit rate and the sender changing the estimated number of receivers that it announces. Receivers wait for a control interval before sending a NAK for a missing packet, and between NAKs for the same packet. The idea is that any of these actions takes at least one round trip time before its effects will be noticed by the actor.

BMTP defines a measurement interval to be twice the control interval. All measurements are averaged over the measurement interval. For instance, the receivers measure the receive rate and the sender estimates the ratio of status packets to data packets over a measurement interval.

These intervals are conservative constants because it is difficult to estimate the round trip time adaptively. An accurate estimate would speed BMTP's response to lost data and increases in available network bandwidth (recall that BMTP immediately slows down in response to reports of decreased bandwidth). Low delay re-transmission is not critical for bulk transfers. The sender must retain data to re-send it, and the receiver must remember out of order arrivals, but they can do this on disk rather than in expensive main memory.

One danger of a short measurement interval is that the sender will not hear status packets from all receivers in each interval. The send rate in packets per second times the measurement interval yields the number of status packets the sender will hear. If reliability is critical, it may be necessary to increase the measurement interval to reflect the number of receivers. Another possibility is to allow more than one status packet per data packet. In practice it is enough that status packets give the sender a sample of the receive rates observed by the receivers, from which the sender need only select the minimum. As long as there are no tiny minorities among the receivers, such a sample will likely include a rate close to the minimum.

The experimental version of BMTP sends 5 percent faster than the minimum reported receive rate. The 5 percent figure is a trade-off between the speed at which BMTP can ramp up its rate to make use of spare bandwidth and the rate at which it forces the network to drop packets.

5. Practical Details

This description of BMTP assumes an infinite stream of bytes. Typical applications will want to transfer a stream of finite-size files. The intent is that an upper protocol layer place file identifiers and delimiters in the payload of BMTP data packets. The BMTP sender needs to be able to ask the file layer for a block of data, giving it a byte offset in the BMTP stream. The sender may ask for any given block of data more than once if it must re-send it. The BMTP receiver will deliver blocks to the file layer along with a byte offset. The receiver may deliver blocks out of order, expecting the file layer to store them until it accumulates a full file.

BMTP cannot guarantee complete reliability. A receiver may be powered down or disconnected from the net indefinitely. The sender may wish to place a limit on the age of data it re-sends, to avoid having to buffer it forever. The sender may also wish to enforce a minimum send rate, knowing that receivers on very slow links might not be able to keep up. These unlucky receivers will wish to generate many NAKs. For this reason, BMTP places the limit K on the weight allowed each receiver in the receiver count estimation algorithm. The sender could also ignore NAKs in status messages that report receive rates lower than the enforced minimum send rate.

If BMTP comes to the end of the data to be sent, it keeps sending empty data packets at a low rate, which allows receivers to continue to report lost packets.

Figure 2 shows the BMTP packet formats. The sender places the measurement interval in milliseconds in each data packet it multicasts so the receivers need not compile this number into their code; the measurement interval does not change over the life of a session. The sender includes two sequence numbers in each data packet. The packet sequence number uniquely identifies packets; a retransmission will have a different packet sequence number than the original transmission. Receivers use the packet sequence numbers to decide how many packets they have missed. The data sequence number indicates where the payload lies in the stream of bytes to transmit. Receivers use data sequence numbers to ask for specific data to be re-sent. Each status packet includes the total number of packets the receiver has recently received and missed. To allow receivers some flexibility, status packets include the interval over which those counts were taken. The status packet also includes one optional data sequence number which the receiver is missing and would like the sender to retransmit.

6. Experimental Results

A test version of BMTP exists, running as a user application on UNIX workstations. A single sender on a 10-megabit Ethernet can sustain about 5 megabits per second to half a dozen receivers. The system also works efficiently when one receiver is connected by way of a modem: the sender keeps the modem busy, losing only the expected 5% of packets.

Simulation with a wide variety of error rates and numbers of receivers illustrates BMTP's strengths and limitations. The simulator used for the following discussion models the network path from the sender to each receiver independently, as if the multicast tree consisted of a separate link from the sender to each receiver. For BMTP this topology is a worst case. Such a flat tree tends to produce different sets of lost packets at different receivers. Receivers in more tree-like topologies would tend to see similar sets of losses due to packets being dropped near the root of the tree. Thus the number of distinct lost packets over all receivers will tend to be higher in the simulated topology than in a tree-like topology. Since BMTP places a limit on the rate at which receivers can produce NAKs, it is at a disadvantage when many different packets are missing.

Each simulated path involves a queue which can hold at most 50 512-byte packets. Packets leave each queue at a rate specified in the description of each simulation below. If a packet arrives from the sender and the queue is full, the new arrival is dropped. Packets may also be dropped with a uniform probability on the way into each queue; this is meant to simulate transmission line errors. The receivers share a single channel on which to send status packets to the sender. This reverse channel runs at a rate of 500 packets per second, has a queue with maximum length 50, and suffers the same transmission line loss rate as the forward channels. The faster reverse bandwidth is reasonable because status packets are much shorter than data packets. Each link has a one-way propagation delay of 50 milliseconds.

Figure 3 depicts BMTP recovering from an initial

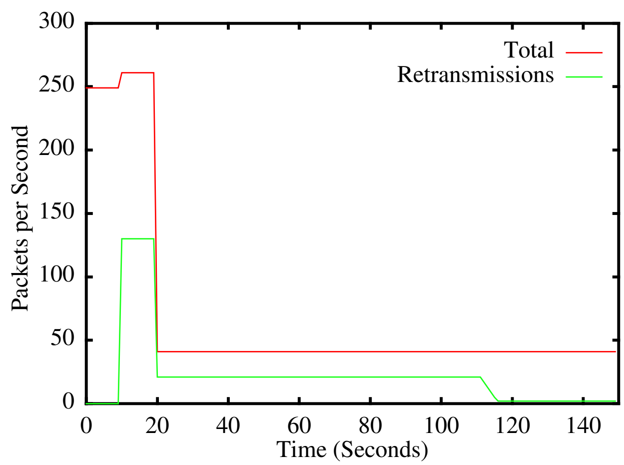

Figure 3: BMTP Start-up

overestimate of the network bandwidth. Ordinarily BMTP would start at a low rate; the high initial rate in this example is for illustration only. In this simulation there are 100 receivers. The slowest receiver can receive at 40 packets per second, and the rest receive at 100 packets per second. The graph's x axis is time in seconds, and the y axis indicates packets sent per second. The upper line indicates the total number of packets the sender sent in a second. The lower line indicates how many of these were retransmissions. The sender starts sending five times too fast, since it knows nothing about network conditions initially; it keeps up this rate for one measurement interval of twenty seconds. In ordinary operation, some receiver would tell the transmitter to slow down fairly quickly; however, the receivers do not report their receive rates until they have observed them for at least one measurement interval of 20 seconds. Thus BMTP's rate control mechanism begins to take effect at time 21, at which point the sender immediately decreases its rate. From then on the sender maintains a rate of about

42 packets per second, just over the capacity of the slowest receiver. Receivers start reporting losses at time 11; recall that they delay such reports for at least one control interval in case the packet is late. Until time 115 the sender uses half the bandwidth for re-transmissions, since over 3000 packets were lost when it was sending too fast. The server is willing to devote no more than half its bandwidth to re-transmissions. After time 120 the sender only needs to re-send the 5% of packets lost because it sends 5% too fast.

Figure 4 shows the effect of transmission line errors

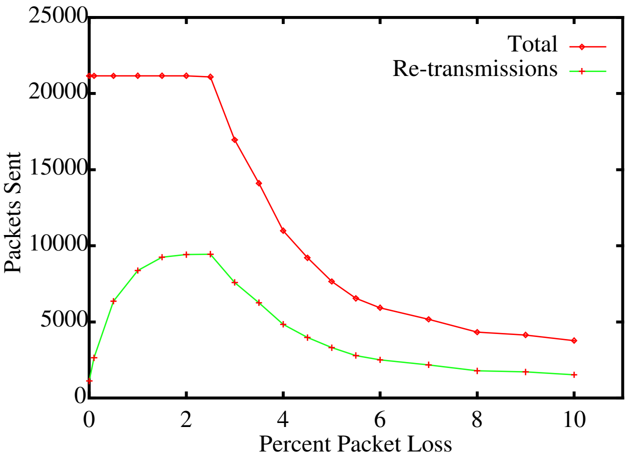

Figure 4: Effect of Transmission Errors

on BMTP performance. Such errors might include telephone line noise disrupting modem traffic. The x axis indicates the percent of packets the simulator throws away to simulate transmission line errors. Packets lost due to queue overruns are not included in the x value, so even at the origin of the graph, about five percent of packets are being lost due to overruns. The y axis shows total packets sent, and of those how many were re-transmissions. Each point represents one 500 second simulation run. The sender starts out at a send rate slightly higher than 40 packets per second; thus the expected number of packets sent in each simulated run is roughly 21000. The simulation configuration is otherwise as described in the previous paragraph.

The leftmost part of Figure 4 is not surprising. Even with no transmission errors, the sender must re-send the 5% of packets lost due to queue overruns. Thus 1113 of the 21000 packets were re-transmissions even when there were no transmission errors; that is just over 5%. As the error rate climbs towards 2%, the sender must re-send correspondingly more packets. The re-send rate flattens out briefly after 2% because the probability that two receivers lose the same packet increases. Such double losses are fixed by single re-transmissions. However, just before 3% loss the sender's overall transmission rate abruptly decreases. The receivers reflect losses not just with NAKs, but also in the receive rates they send back. Whenever the sender sees a reported receive rate more than 5% less than its current rate, it decreases the rate. So any transmission error rate approaching 5% will cause the sender to decrease its rate. If the losses were caused by congestion, they would soon go away, if only because BMTP decreases its offered load. However, transmission errors don't decrease when the load decreases. Thus a high enough error rate causes the sender to gradually decrease its send rate, even though it would be acceptable for it to send faster. In practice, few IP networks have transmission errors even as high as one percent. They may experience more congestion loss than that, but BMTP's rate reduction algorithm reacts correctly to congestion loss.

BMTP still works with 1000 users. Figure 5 shows

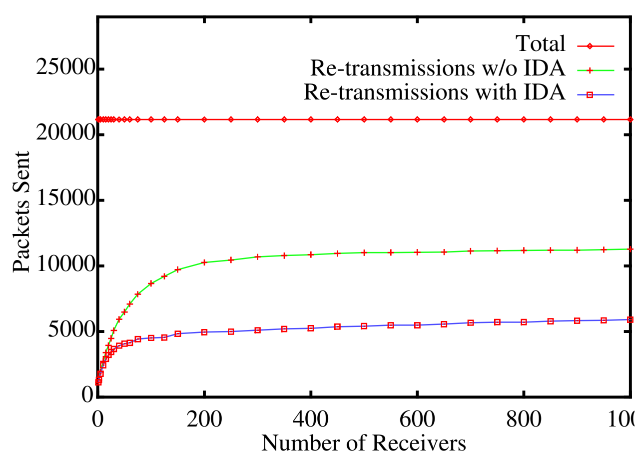

Figure 5: Effect of Many Receivers

the effect of increasing numbers of receivers. Each point in the top line represents the total number of packets sent, and each point in the middle line the number which were re-sends, during a 500 second simulation with the indicated number of receivers. In each simulation, the network bandwidth to one receiver was 40 packets per second, and to all the others 100 packets per second. Each receiver independently loses 1% of data packets because of transmission errors. Just for this simulation the sender always gives preference to retransmissions over sending new data, and places no limit on the fraction of bandwidth devoted to retransmission. From the fact that the total is always just over 20000 packets, or 40*500, we can conclude that the sender sustains the correct transmit rate. The danger, which BMTP avoids, is that status packets from the slowest receiver might not arrive at the sender often enough, allowing the sender to increase its bandwidth to the 100 packets/second available to all the other receivers.

Note also that the sender never needs to devote more than about half the bandwidth to re-transmissions. A typical packet is lost by 1% of the receivers. Thus, for

1000 receivers, there is a danger that up to 10 receivers might NAK each packet. Since the sender does not keep track of which packets it has recently re-sent, it risks using 90% of its bandwidth for re-transmissions. BMTP avoids this problem because the receivers return status packets one by one, not all at once. The expected interval between times when different receivers NAK the same lost packet is N/(40*N*1%), where N is the number of receivers, 40 is the rate in packets per second, and 1% is the packet loss rate. This comes out to 2.5 seconds. The round trip time, including the maximum queueing time and propagation delay of 50 milliseconds, is just over a second. Thus the re-send triggered by one receiver's NAK typically arrives back at all the receivers before a second receiver gets a chance to NAK the same packet.

All the simulations described above have asymmetric receiver bandwidths: there is some slowest receiver. BMTP would work poorly if all the receivers had identical bandwidths and queue lengths. In such a case all receivers would experience an independent 5% loss rate caused by the sender transmitting 5% too fast. At least one receiver would miss every packet, causing the sender to devote half its bandwidth to retransmission. On the other hand, if there is some slowest receiver, only that receiver misses 5% of packets. Note that the weighting mechanism allows that receiver to repair the misses. It's reasonable to expect that real networks (as opposed to simulators) will exhibit such asymmetry.

7. Effects of IDA

Rabin's Information Dispersal Algorithm (IDA) [9] can be added to BMTP in a straightforward manner. IDA allows correction of losses within a group of m packets with a number of retransmissions equal to the maximum number of packets lost from the group by any one receiver. For instance, if each receiver loses a different two packets out of a particular group, the sender can repair all the losses with just two IDA retransmissions. This makes IDA particularly appropriate for multicast. The ideas in the ensuing discussion all spring from the original IDA paper.

BMTP with IDA treats successive groups of m packets' worth of input data separately. Suppose a packet holds 512 bytes. Instead of sending each packet's worth of data in a packet, BMTP with IDA sends a packet-sized *piece* computed as follows:

$$\begin{bmatrix} p_1 & \cdots & p_{512} \end{bmatrix} = \begin{bmatrix} v_1 & \cdots & v_m \end{bmatrix} \cdot \begin{bmatrix} b_1 & \cdots & b_{511m+1} \\ \cdots & \cdots & \cdots \\ b_m & \cdots & b_{512m} \end{bmatrix}$$

b_1 through b_{512m} are one group of input data. v_1 through v_m are random numbers, chosen separately for each piece sent. The sender places p_1 through p_{512} and v_1 through v_m in a packet and multicasts it to all the receivers. Ordinarily the sender does this m times for each group of data, sending m packets. Call the v and p row-vectors the sender transmits V_i and P_i for i from 1 to m. The sender is essentially calculating the following, one P and V at a time:

$$\begin{bmatrix} P_1 \\ \cdots \\ P_m \end{bmatrix} = \begin{bmatrix} V_1 \\ \cdots \\ V_m \end{bmatrix} \cdot \begin{bmatrix} b_1 & \cdots & b_{511m+1} \\ \cdots & \cdots & \cdots \\ b_m & \cdots & b_{512m} \end{bmatrix}$$

When a receiver has received m pieces for a group, it reconstructs the original data thus:

$$\begin{bmatrix} b_1 & \cdots & b_{511m+1} \\ \cdots & \cdots & \cdots \\ b_m & \cdots & b_{512m} \end{bmatrix} = \begin{bmatrix} V_1 \\ \cdots \\ V_m \end{bmatrix}^{-1} \cdot \begin{bmatrix} P_1 \\ \cdots \\ P_m \end{bmatrix}$$

If no packets are lost, this costs some extra cpu time and the extra bandwidth required to send the V vectors.

Suppose a receiver fails to receive a packet from a group, and thus is missing a V/P pair. In BMTP with IDA, the receiver will ask the sender to send an additional piece from the group. The sender calculates the additional piece as described above. Since the new V vector is randomly chosen, the new piece is unlikely to be the same as any previous piece from the same group. However, the equations above hold for any m pieces, so the receiver will be able to reconstruct the group from any combination of m pieces. A receiver will keep requesting new pieces for each group until it has m; it follows the rules previously described governing when it is allowed to send status messages.

Now suppose two receivers received only m-1 pieces from a group, but missed different pieces. An ordinary BMTP sender would have to re-send two packets to fix this. IDA allows both losses to be fixed with one re-send: the equations above allow the receivers to use any distinct m pieces to reconstruct a group.

IDA does not help when fixing losses in different groups, so a group should contain as many bytes as possible. One could increase the piece size beyond 512 bytes to achieve this goal, but still send each piece in a packet. However, the Internet fragments large packets into small ones, and loss of any one fragment causes the remainder of the packet to be useless. Thus it seems most reasonable to avoid fragmentation by keeping pieces small. Another approach to making groups large is to make m large. The matrix multiplies needed to create pieces and reconstruct the original data require O(m)

operations per byte. Thus large values of m will consume large amounts of CPU time. For purposes of experimentation, BMTP with IDA uses groups of m=5 pieces of 512 bytes each.

There is a chance that the matrix of V vectors will fail to invert. In this case, the receiver discards one randomly chosen piece and ask the sender for a new one. The sender could avoid this by choosing V vectors that are linearly independent. The per-group state required in the sender does not seem worth the added efficiency.

All of the arithmetic described above is carried out in the field of integers mod 257. There are one too many field members to encode in a byte. BMTP with IDA uses an escaping mechanism to mark the extra value, which increases the packet size slightly.

An implementation of BMTP with IDA on 120 mHz Pentiums can send at four megabits bits per second, but receive at only one megabit per second. Essentially all of the time is in the matrix multiplies. The algorithms involved could certainly be improved.

The simulation of Figure 5, when run with IDA and m=5, results in about 28% re-sends with 1000 receivers, rather than the 50% without IDA. One would expect every group of five original packets to require roughly one extra piece to fix losses: a re-send rate of 16%. For a typical group, 50 receivers will need this extra piece. At a one percent loss rate, roughly 60% of these extra pieces will themselves be missed by one of the 50 receivers. This implies a total re-send rate of 26%, which agrees with the simulation result.

8. Further Work

Some of BMTP's mechanisms could be improved. It intentionally overloads the network by 5% in order to detect spare capacity. While this is arguably in the tradition of TCP, the impact could be decreased by a periodic probing algorithm. BMTP can recover from the 5% loss due to that overload, as well as a loss rate of a few percent due to transmission errors. Its tolerance to transmission errors could be increased if the sender noted any loss rate that persists despite substantial rate reduction. BMTP uses fixed measurement and control intervals, which slow its response to changing conditions; these could be based on dynamic measurements of round trip time. BMTP allows a fixed maximum weight for the probability with which a receiver sends a status packet; this maximum should be governed by the total number of receivers.

More experience is needed with the interaction between multiple BMTP sessions, and between BMTP and other protocols.

BMTP leaves some decisions up to the application, but lacks a well-defined interface with which an application can declare its policies. An application may wish to limit the maximum age of data BMTP should re-send, or to ignore very slow receivers, or to ensure complete reliability to a known set of receivers. BMTP could enforce these policies if the application could ask it to do so.

9. Conclusions

BMTP will allow applications like Netnews, mailing lists, and mass software distribution to take advantage of the Internet's multicast system. It multicasts bulk data with a reliability and capacity to scale not previously available. It avoids overloading the network with data or the sender with status packets. It sustains high throughput even with a large number of receivers or a modest transmission error rate. It usually avoids re-sending any one lost packet more than once even if multiple receivers missed it. BMTP achieves these effects by counting the receivers, using this count to limit the rate at which receivers can send status packets, and feeding rate information back to the sender from the receivers.

References

[1] ARMSTRONG, S., FREIER, A., AND MARZULLO, K. Multicast Transport Protocol, RFC1301, 1992.

[2] BOLOT, J-C, TURLETTI, T., AND WAKEMAN, I. Scalable Feedback Control for Multicast Video Distribution in the Internet. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM 1994).*

[3] DAKA, J., AND WATERS, A., A High Performance Broadcast File Transfer Protocol. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM 1988).*

[4] DEERING, S., Multicast Routing in a Datagram Internetwork, Ph.D. thesis, Stanford University, December 1991.

[5] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C., AND ZHANG, L. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM 1995).*

[6] JACOBSON, V., Congestion Avoidance and Control. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM 1988).*

[7] LIN, J., AND PAUL, S. RMTP: A Reliable Multicast Transport Protocol. *IEEE INFOCOM 1996, March 1996.*

[8] POSTEL, J., Transmission Control Protocol, RFC793, 1981.

[9] RABIN, M., Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance, *Journal of the ACM, Volume 36, Number 2, Pages 335 to 348. ACM Press, New York, 1989.*

[10] TALPADE, R., AND AMMAR, M. H., Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service, *Georgia Institute of Technology College of Computing Tech Report GIT-CC- 94-47, October 1994.*

[11] WEINSTEIN, L., Broadcasting of Netnews and Network Mail

via Satellite, *USENIX Conference Proceedings, 1984*.

[12] XTP Protocol Definition, Revision 3.6. Protocol Engines Incorporated, Mountain View, CA, 1992.