Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCS-98-19

1998-01-01

Congestion Control in Multicast Transport Protocols

Rajib Ghosh and George Varghese

We discuss congestion control mechanisms in multicast transport protocols and we propose TCP-M - a TCP-friendly Multicast transport protocol. TCP-M uses IP multicast to deliver data packets and acknowledgements to provide reliability. Ack implosion at the source is prevented by fusing acknowledgements at some intermediate routers. TCP-M reacts to network congestion exactly like TCP by having the sender emulate a TCP sender.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Ghosh, Rajib and Varghese, George, "Congestion Control in Multicast Transport Protocols" Report Number: WUCS-98-19 (1998). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/471

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

Congestion Control in Multicast Transport Protocols

Rajib Ghosh
George Varghese

WUCS-98-19

June 29, 1998

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

We discuss congestion control mechanisms in multicast transport protocols and we propose TCP-M - a TCP-friendly Multicast transport protocol. TCP-M uses IP multicast to deliver data packets and acknowledgments to provide reliability. Ack implosion at the source is prevented by fusing acknowledgments at some intermediate routers. TCP-M reacts to network congestion exactly like TCP by having the sender emulate a TCP sender.

Keywords: Internet Protocol, IP Encapsulation & Tunneling, IGMP, Multicast, MBone, TCP.

This work was supported by National Science Foundation under grant NCR-9628145 and the ONR Young Investigator Award.

Congestion Control in Multicast Transport Protocols

Rajib Ghosh
rajib@ccrc.wustl.edu
+1 (314) 935-4163

George Varghese
varghese@askew.wustl.edu
+1 (314) 935-4963

Multicasting has become an important topic of research due to the increasing number of applications which require distribution of data simultaneously to multiple receivers. While multimedia applications can handle a certain degree of loss of data, other applications require reliable, sequenced and lossless delivery. An example is a web server distributing files to its mirror sites. For such applications we can have separate TCP [Ste94, Tan81] connections from the sender to each receiver. However, this is inefficient since each packet that is sent to M receivers that are reachable on a common set of links from the source is duplicated M times on the common path. If the sender uses multicast, each packet traverses a network path only once. Further, it also saves on the total transfer time taken. Thus, we need reliable multicast protocols for efficiency and performance.

The MBone [Eri94] provides us with a multicast capable network layer using IP Multicast [Dee89]. However, it is prone to packet loss and misordering of data. Multicast transport protocols [AFM92, FJM+95, LP96, RBR98, SFLT98, TA95] provide a reliable service by building a transport layer on top of the (multicast) network layer.

Congestion control has always been a principal problem for reliable multicast. Several protocols [MJV96, VRC98] respond to congestion by organizing the data in different bandwidth "layers" and dropping data layers at the congestion points. However, this approach works better for audio/video applications. Bulk data for file transfers can also be organized into layers, but this strategy becomes very complex beyond a few layers [Vic98].

Unlike a unicast connection, a multicast protocol has to adjust to congestion in multiple network paths. The important issues are Ack[1] (acknowledgment) implosion, interpretation of receiver's window for many receivers, and an accurate round-trip-time (RTT) estimation for the multiple receivers. If the receivers acknowledge data packets as in unicast protocols, then, if there are M receivers, the source receives M acknowledgments for each packet. For large M, this leads to decrease in performance at the source due to processing overheads. Further, connection parameters such as window of data sent and round-trip time have to be determined too.

In this paper, we analyze congestion control in multicast transport protocols. We propose a simple reliable multicast transport protocol which does congestion control like TCP and scales very well to large number of receivers. Through a novel design approach, it can be deployed incrementally and

[1] Throughout this paper we will use *Ack* and *Nack* as abbreviations for acknowledgment and negative acknowledgment respectively.

easily. Our protocol is appropriate for reliable file transfers, e.g., distributing a file from a web server to its multiple cache proxies. We implemented our multicast protocol (Section 3) in `Ns` [UCB98] and used it to evaluate our scheme. We present our results in Section 6. We start by discussing related work in this area.

1. Related Work

There have been several multicast transport protocols proposed in the literature so far [AFM92, FJM+95, LP96, RBR98, SFLT98, TA95]. RMTP [LP96] groups receivers into local regions and uses a *Designated Router* (DR) in each region so that the responsibilities of processing acknowledgments and retransmissions is distributed among the sender and the several DRs. Receivers in each local region send their Acks to the DR who is responsible for caching packets from the sender and retransmitting packets lost in the region. RMTP is receiver-driven, receivers periodically transmit status messages to the sender.

SRM [FJM+95] is intended for applications like whiteboard conferencing. It assumes the IP multicast delivery model and provides end-to-end reliability by having the receivers request missing data (through multicast). Any host who has a copy of the requested data can answer the request. Sources also transmit periodic *session* messages reporting their sequence number states. Receivers use these to detect losses and to determine proximity to the sources.

MTP [AFM92] uses Nacks (negative acknowledgments) and a *master process* to guarantee reliable data delivery. The sender responds to the Nack by retransmitting requested data. A source can transmit data only when it receives a *token* from the master process.

SCE [TA95] uses a *Single Connection Emulation* (SCE) layer between a unicast transport layer (e.g. TCP) and the multicast network layer. This added layer interfaces between the single destination transport layer and the multicast network layer and provides the necessary multicast functionality. SCE uses IP Multicast to transmit the data packet and on receiving acknowledgments from all the receivers passes a single one up to the transport layer. Thus, reliability is taken care of by the unicast transport layer. One obvious disadvantage of SCE is Ack implosion.

PGM [SFLT98], recently proposed by Cisco, is receiver-oriented and Nack based. To avoid Nack implosion, PGM uses state at *Network Elements* (routers) to suppress Nacks. Thus, the data source receives only one copy of the Nack for a lost data packet. To prevent retransmission of data to uninterested receivers, each network element maintains retransmission state on each interface. The data source also transmits *Source Path Messages* periodically to establish source path states in network elements. PGM assumes the Nacks to be routed through the exact reverse of the paths used by the data packets.

MTCP [RBR98] uses a multi-level tree where the root is the sender and all other nodes are receivers. Internal tree nodes called *Sender's agents* (SAs), much like DRs in RMTP [LP96], take care of handling Acks and Nacks from receivers and retransmitting lost packets. Each SA independently monitors the congestion level of its children and sends an Ack back to its parent summarizing it. A summary is essentially the receiver window size. The sender uses this summary to regulate its data flow.

In summary, research on congestion control in multicast applications has mostly focussed on audio/video data. Schemes like [MJV96] and [VRC98] react to congestion by dropping data layers at the receivers. While multimedia applications can deal with a poorer quality of video or sound, this approach does not work well for file transfers. As argued in [Vic98] finding a suitable data organization for a file becomes very complex as the number of layers increases.

RMTP [LP96] uses retransmission requests from receivers to handle network congestion. The sender reduces its transmission rate when it receives too many such requests. SRM [FJM+95], MTP [AFM92] and PGM [SFLT98] do not deal with congestion control. In MTCP [RBR98], a sender controls its sending rate based on the congestion summaries it receives from its immediate SAs. SCE [TA95] relies on the unicast transport protocol for handling congestion but has an Ack implosion problem.

Existing transport protocols (e.g. RMTP, PGM, SRM) rely on the indirection through the IP group name to avoid having the source know all the receivers. This approach works fine for conferencing applications where the additional loss recovery provided by SRM, PGM, etc. effectively improves the transient congestion and errors. But it provides no indication of crashed receivers or which receivers have received the data correctly. For example, if a receiver crashes in the middle of data transfer, the source does not know in SRM or PGM. However, for applications like file transfer, such reliability semantics are not enough. Applications must know which receivers received the data successfully. Therefore, we require positive feedback from receivers in the form of acknowledgments.

It has been argued [Bro97, RBR98, VHC, VRC98] that a multicast transport protocol should react to congestion like TCP does. Congestion control in TCP has been studied for a long time and it has improved over the years. Therefore, we should use a tried and tested method rather than inventing a new one. Further, if congestion control in multicast is *TCP-like*, then, multicast and unicast applications can co-exist fairly. Most unicast applications use TCP as the transport protocol. In other words, multicast applications should be *TCP friendly.*

2. Our Design Goals

In this section we briefly summarize the goals which motivated us to come up with our design of TCP-M (*TCP-friendly Multicast*).

- **Multicast Transport Protocol:** We wish to design a transport protocols for multicast file transfer applications from one sender to multiple receivers.
- **Reliable:** The data transfer should be reliable. Further, the sender should be able to know which receivers received the data successfully and which did not. Or in other words, the protocol should guarantee reliable data delivery at the sender — like the unicast *File Transfer Protocol.*
- **TCP-friendly:** Our multicast protocol should react to network congestion like TCP. This is very important because about 80% of internet traffic is generated by TCP [Bra96] and therefore a multicast connection should be able to co-exist fairly with TCP connections.
- **Scalable:** It should be scalable to large networks and a large number of receivers.
- **Incrementally Deployable:** Since our scheme requires modifications at intermediate routers, it should allow enough flexibility so that it can be deployed easily and incrementally over the Internet.

3. A Scalable TCP-friendly Congestion Control Scheme for Reliable Multicast

While the source in SCE essentially behaves like a TCP source and hence reacts to congestion like TCP, SCE has an Ack implosion problem. In this paper, we propose a new reliable multicast

protocol called TCP-M (TCP-friendly Multicast) in which the sender reacts to congestion like TCP (or SCE). However, we also allow some routers to have the ability to coalesce acknowledgments to avoid Ack implosion. Since we do not require all routers to implement Ack fusion, our protocol can be deployed incrementally. Further, the overhead of coalescing acknowledgments can be limited to a few routes (for example on a LAN that has many receivers).

Thus, the major difference between our protocol and SCE is the use of an incrementally deployable form of Ack fusion. This requires the addition of several mechanisms not found in SCE. Further, the SCE paper [TA95] focussed on error recovery and efficiency and not on congestion control. Since our focus is on congestion control, we report several experiments to show that TCP-M provides adequate congestion control in a multicast environment.

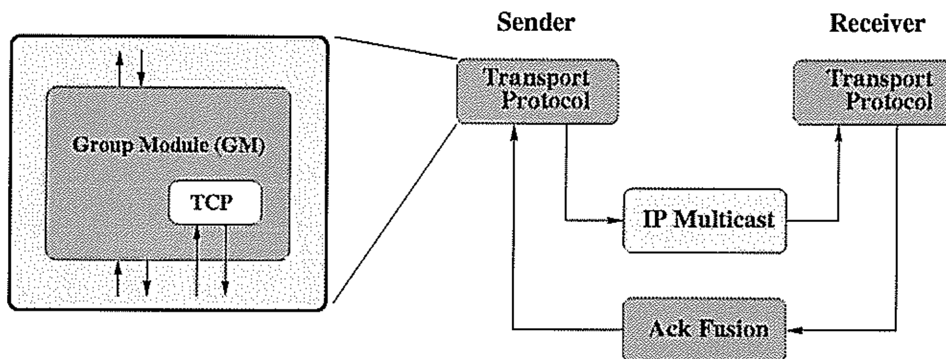


Figure 1: Overview of TCP-M

Figure 1 gives a high-level overview of the protocol structuring in TCP-M. We propose to have a unicast transport protocol (preferably TCP with very little modifications) at the sender as well as at the receivers. The source transport protocol sends the data packet to multiple receivers with the help of IP Multicast [Dee89]. A *Group Module* (GM) built on top of the network layer provides transport services to the application layer above. It also interfaces between the application and the unicast transport protocol (TCP).

The transport protocol at the receivers receive the data packets as usual and sends acknowledgments back to the source. The receiver may also use delayed acknowledgments [Ste94]. Intermediate routers and the source group module layer (we call them *Fusion Nodes* (FNs)) collect acknowledgments from all *downstream* links, *fuse* them, and forward them to the source. The source TCP thus receives only one acknowledgment as it would expect.

Figure 2 depicts an example scenario of TCP-M in execution. There are three receivers $D1$ through $D3$. Routers $R1$ and $R2$ are capable of doing Ack fusion (fusion nodes) and are shaded solid in the figure. The source S is also a fusion node. Through a novel design approach we have made TCP-M incrementally deployable. TCP-M works even if not all intermediate routers are fusion nodes. In the extreme case, TCP-M works even if only the source is a fusion node, in which case it is very similar to SCE [TA95]. In the above example, acknowledgments from $D1$ and $D2$ are fused at $R1$ and not at $R3$. However, for large networks, the performance of TCP-M is best if many intermediate routers are fusion nodes.

In Figure 2 we illustrate the receivers acknowledging sequence number 4. The Acks from $D1$ and $D2$ are not fused at $R3$ because $R3$ is not an FN. On the other hand, the acknowledgment from $D3$ is altered at $R2$ because $R2$ is an FN. All the Acks are then fused at $R1$ and a single Ack is sent to the source S. Then, the source sends data with sequence number 5.

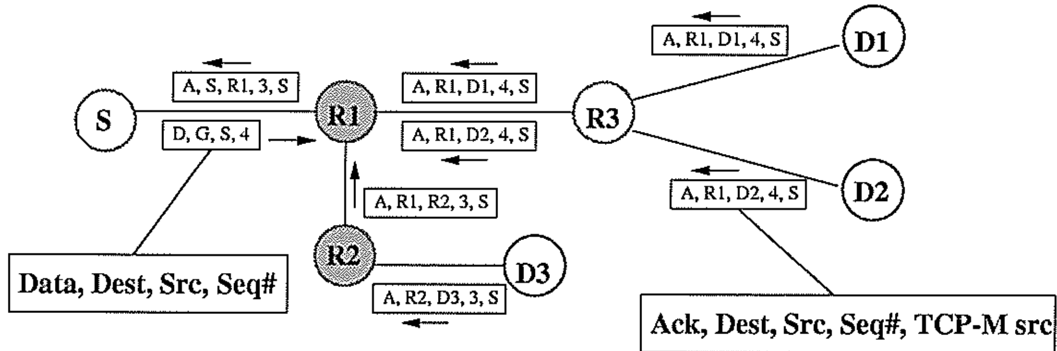


Figure 2: Ack fusion in TCP-M

An obvious consequence of fusing acknowledgments is that the sender is slowed down to the slowest of all receivers. However, this should not be a problem for file transfer applications. Some of the possible applications of TCP-M are distributing a file from a web server to its cache proxies, a software company distributing upgrades to its customers, synchronizing files between different machines and sending e-mail to mailing lists.

The source transport protocol is essentially unicast since it transmits packets to a single address (a multicast address) and receives packets from a single address (due to Ack fusion). Thus, we can potentially have any existing unicast transport protocol at the sender. Specifically, we use TCP as the transport protocol. We can also use other schemes such as DECbit [RJ88] and RPCC [GV97].

4. Description of TCP-M

TCP-M is scalable and easily deployable. It handles congestion in the network exactly like TCP. In the following subsections we describe how TCP-M establishes and terminates a connection, how the data is transferred and Acks are fused and how network failure and other error conditions are handled. Some of the protocol details are specific to the embedded unicast transport protocol and in such cases we have described it specifically for TCP.

4.1. Connection Establishment

We assume that the application program is provided with the list of receivers and the multicast group address to send to. The application passes this information to the Group Module in TCP-M. The GM then sends unicast *JOIN_REQUEST* messages to all the receivers requesting them to join the above multicast group. We assume there is a TCP-M server running at a pre-determined port on the receivers. The receiver, on receiving the *JOIN_REQUEST* message, attempts to join the multicast group using IGMP [Dee89] and sends an acknowledgment (*JOIN_ACK*) back to the source.

The GM waits a certain amount of time (*JOIN_TMOUT*) for *JOIN_ACKS* from all receivers. If the timer expires before all the *JOIN_ACK*s have been received, it informs the application of the failed receivers. As an enhancement, *JOIN_REQUEST* messages can be retransmitted a few times to allow *slow* receivers to catch up.

After all the *JOIN_ACK*s have been received or the retransmission timer has expired sufficient number of times, the GM sends an RSVP-style [ZDE+93] *PATH* message to the multicast address.

The *PATH* message contains a *parent* field and the GM sets it to its own address. "Address" referred to hereafter in this section consists of IP [Pos81] address and port number. All intermediate *Fusion Nodes* (FNs) record the *parent* address from the *PATH* messages they receive and replace it with their own addresses. All receivers record the *parent* address from *PATH* messages they receive.

After a receiver receives a *PATH* message, it sends a *REVERSE_PATH* message to its *parent*. Note that the *parent* for a receiver or an FN is the closest router upstream which can fuse acknowledgments. Thus, an FN which receives a *REVERSE_PATH* message records the sender (of the message) as a host to expect acknowledgments from. To fuse acknowledgments an FN must know all the receivers to expect acknowledgments from. *REVERSE_PATH* messages contain the address of the TCP-M source and the address of the receiver who originated it. The TCP-M source's address is required because the state set up at an FN is per source. The address of the receiver is required by the TCP-M source as described later. The FN also rewrites the source and destination address on the message and forwards it to its own *parent*.

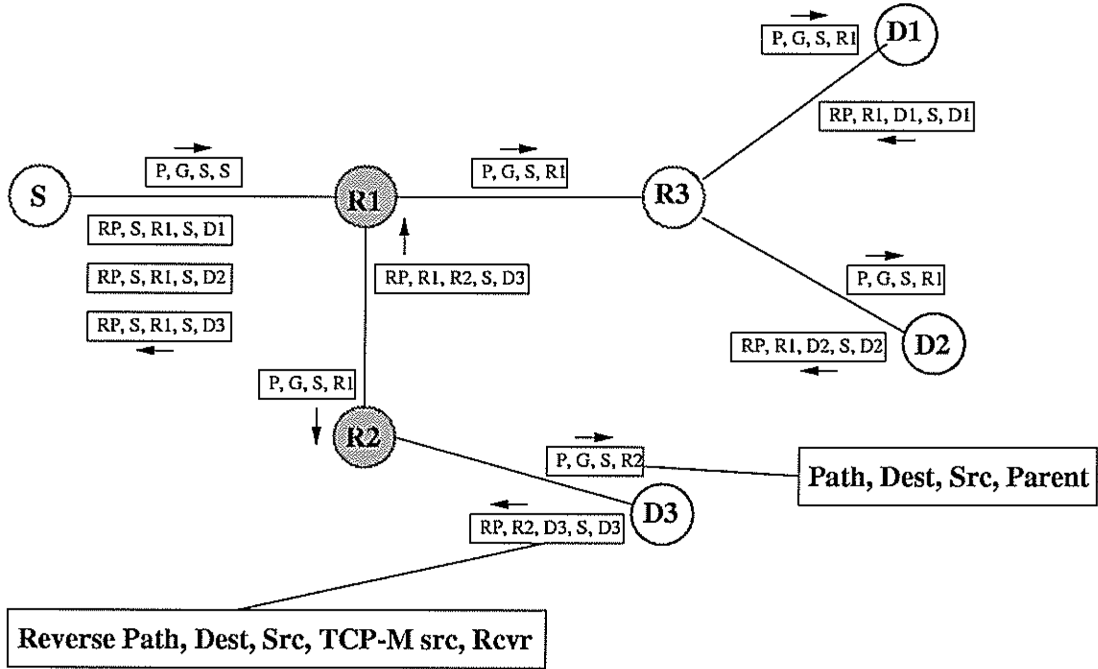


Figure 3: TCP-M: Connection establishment and state setup

As an example, consider the scenario in Figure 3. The 4-tuple messages shown next to the links are the *PATH* messages while the larger ones are the corresponding *REVERSE_PATH* messages. Only the fusion nodes ($R1$, $R2$ and S) replace the *parent* addresses in the *PATH* messages. In Figure 3 only $R1$ and $R2$ modify the *PATH* message as it passes through them. The *REVERSE_PATH* messages from a receiver or an FN to its *parent* build the state required for doing Ack fusion at the FNs. For example, the *REVERSE_PATH* messages from $D1$ and $D2$ to $R1$ tell $R1$ to expect acknowledgments from them. Similarly, the messages from $R1$ to S tell S to expect Acks from $R1$ only. The source S receives as many *REVERSE_PATH*s as the number of receivers.

Note that receiving a *REVERSE_PATH* message from a receiver implies it has already subscribed to the multicast group (since it received the corresponding *PATH* message) and the required

state has been set up to fuse acknowledgments all along the path to the source. The source waits a certain amount of time ($PATH_TMOUT$) to receive $REVERSE_PATH$s from all receivers (who acknowledged with a $JOIN_ACK$ earlier). If all the receivers do not reply within the stipulated time, the GM retransmits the $PATH$ message a few times. After a certain number of retransmissions, the GM informs the application of the failed receivers and proceeds with data transfer for the remaining receivers.

The $JOIN$, $JOIN_ACK$, $PATH$ and $REVERSE_PATH$ messages could be UDP [Pos80] packets each with a different protocol number (for the *Protocol* field in UDP packets) and a payload size. We can choose protocol numbers unassigned in [RP94].

4.2. TCP-M Data Transfer

Before the source can start the data transfer, the embedded transport protocol has to agree with the receivers on the various connection parameters. Specifically, TCP has to agree on the initial sequence numbers, window sizes, maximum segment size (MSS), etc. [Ste94, Tan81]. Further, TCP requires a three-way handshake to eliminate errors from old or duplicate packets in the network.

TCP's SYN is multicast to the group. All the receivers reply with a SYN_ACK. The GM accumulates all the SYN_ACKs. On receiving all of them, it sends a single SYN_ACK to TCP with the largest sequence number among those received. The window size is the minimum of all the received window sizes and the MSS is the minimum of MSSs received. The TCP sender acknowledges this SYN_ACK which is then multicast to the group address. This completes the TCP three-way handshake. The TCP receiver has to be patched to accept this multicast acknowledgment (because of possible skips in choosing a single sequence number for all the receivers).

All subsequent data packets are multicast to the group address. Receivers acknowledge them as usual. Fusion nodes accumulate the acknowledgments and fuse them into one by taking the minimum of the sequence numbers received so far. We present the Ack fusion algorithm in the next subsection. FNs also take care of duplicate acknowledgments from receivers. Thus, the TCP sender receives a single acknowledgment as it would expect.

Figure 2 illustrates this mechanism with an example. Acknowledgment packets at the receiver are encapsulated [Sim95] and sent to the *parent* address. Encapsulation is required because the FNs must know the address of the TCP-M source in order to perform Ack fusion. To ease lookups, we can choose to use a number unassigned in [RP94] for the *Protocol* field in encapsulated IP packets — e.g., 105 and call it *TCP-M_ACK*. The encapsulated acknowledgments are sent *hop-by-hop* to the FNs, who fuse them and pass them upstream towards the source. In Section 4.3 below we discuss in detail how Ack fusion is done at FNs.

TCP estimates the round trip time from the acknowledgments and uses it to set retransmission timers. Essentially, it uses the largest round-trip time to any receiver as the round-trip delay. The congestion control and avoidance mechanism is exactly the same as in unicast TCP. If a receiver delays its acknowledgment, eventually the TCP sender times out and retransmits the data as usual. Thus, the behavior of the transport protocol is exactly like unicast TCP. There is a single congestion window for all the receivers and it grows and shrinks in response to the round-trip times and packet losses experienced by the source.

We should also note that TCP-M does not allow receivers to be added to an already existing connection. We wish to use TCP-M for reliable file transfers and receiving an incomplete data file is futile.

```
Seq:  an array of integers indexed by host/router addresses,
   initially all -1
duplicate:  an integer, initially 0
last_acked:  an integer, initially -1
parent:  address of parent fusion node

On receipt of an acknowledgment a:
   register_ack(a) (* update ack-table as required *)
   send_ack() (* if possible *)

procedure register_ack(a)
   If (a.seq > Seq[a.src]) then
      Seq[a.src] = a.seq
      duplicate = 0
   else
      increment duplicate

procedure send_ack()
   temp = min(Seq[s]) ∀ hosts s
   If (temp > last_acked or 0 < duplicate < MAX_DUPLICATE)
   (* fresh or duplicate Ack *)
      last_acked = temp
      send acknowledgment with sequence number = last_acked,
         source = my address, and
         destination = parent
```

Figure 4: Algorithm at a fusion node for fusing Acks

4.3. Algorithm for Fusing Acknowledgments at Routers

Every fusion node maintains an *ack-table* in its memory for every multicast connection that it supports. The ack-table contains the information necessary for fusing Acks from its *children* in the multicast tree. Figure 4 gives the the algorithm at the multicast routers for fusing Acks. The ack-table keeps a record of the Acks received for each *downstream* link. The end routers, whose children are the receivers of the multicast group, maintain the Ack information for each receiver application (destination:port pair) instead of just each receiver (there might be multiple receivers on a single machine).

On receiving an acknowledgment, a router first updates the ack-table with the new information. The ack-table is an array of sequence numbers for each downstream link. For an acknowledgment received, the source field is the address of the router which sent it. For acknowledgments from the multicast receivers, it is the host's address. If it acknowledges previously unacknowledged data (the comparison in procedure *register_ack*() checks this) the new sequence number is recorded.

The algorithm then checks if the ack-table allows an Ack to be sent to the data source upstream. The minimum of the sequence numbers in the array *Seq* is the sequence number that has been acknowledged by *all* the receivers. The router then sends an acknowledgment with that sequence number (if it has not been sent already; this checking is done to avoid unnecessary duplicate Acks). The source must be its own address, since its upstream router would be expecting an Ack with *that* address.

Congestion control and fast error recovery mechanisms in TCP depend on receiving duplicate acknowledgments. Therefore, we had to design Ack fusion in FNs so that it recognizes duplicate acknowledgments from receivers and forwards them towards the source. We did this by introducing an integer *duplicate* which is positive if the last acknowledgment received is a duplicate. An acknowledgment is a duplicate if it is not a *fresh* one for that particular receiver. Thus, the FN also propagates duplicate acknowledgments.

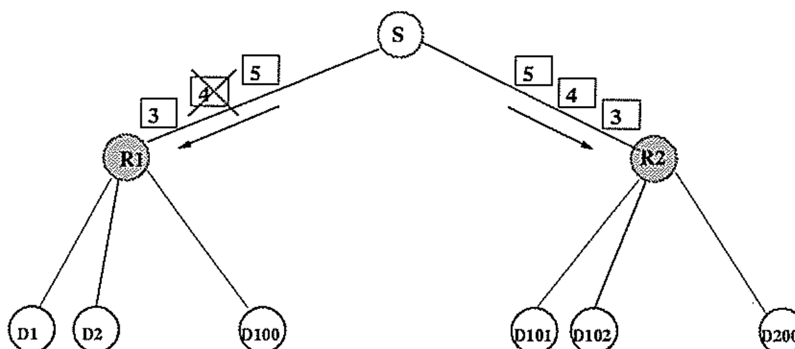


Figure 5: Duplicate acknowledgments and Ack implosion

However, propagating all duplicate acknowledgments would lead to an Ack implosion problem. For example in Figure 5, if sequence number 4 is lost and 5 is received by all the receivers $D1$ through $D100$, then they would send duplicate acknowledgments to $R1$ requesting sequence number 4. $R1$ would forward all the acknowledgments to S and thus S would receive 100 duplicate acknowledgments for sequence number 3. To avoid this problem, fusion nodes maintain a duplicate count and forwards upto $MAX_DUPLICATE$ duplicate acknowledgments. $MAX_DUPLICATE$ should be at least three since TCP responds with fast retransmit on receiving three duplicate acknowledgments[2].

4.4. Connection Termination

After the last data packet has been sent and its acknowledgment received, the TCP-M source proceeds to terminate the connection as follows. The embedded TCP sends a FIN packet to the multicast address telling all the receivers to terminate the connection. On receiving the FIN packet the receivers leave the multicast group using IGMP messages. Then they send a FIN_ACK and a *TCP-M_CLOSE* message back to the source.

The FIN_ACK proceeds towards the source just like an acknowledgment to a data packet, it is fused on its way with other FIN_ACKs. After a FN receives all the FIN_ACKS from its downstream neighbors it forwards a single FIN_ACK towards the source and deletes all state associated with that particular multicast connection. However, the *TCP-M_CLOSE* messages are not fused.

Thus, when the source receives a FIN_ACK it terminates too. Loss of FINs or FIN_ACKs are treated identical to loss of data packets — they are retransmitted if required. From all the *TCP-M_CLOSE* messages the GM also knows all the receivers who successfully received all packets. This information is passed upto the application.

[2] We used a value of five just to be safe - in case of lost Acks.

4.5. Handling Network Failures

Every protocol in the Internet must be capable of handling failures and exceptional conditions. In this subsection we discuss how TCP-M deals with host and network failures. If a receiver fails during connection establishment it is eventually timed out after a few tries and the protocol proceeds without it (described above in Section 4.1). Meanwhile the application is also informed of the failed receiver.

Next, we describe how network failure is handled during the data transfer. A receiver retransmits its *REVERSE_PATH* message periodically every *RCVR_TMOUT* interval of time. The *REVERSE_PATH* messages are forwarded as before by the FNs. Every time an FN receives a *REVERSE_PATH* message from its downstream neighbor, it starts a timer with a value *TCP-M_TMOUT* (Figure 6).

```
Seq:  an array of integers indexed by host/router addresses,
   initially all -1
Alive:  an array of boolean values, initially all TRUE
duplicate:  an integer, initially 0
last_acked:  an integer, initially -1
parent:  address of parent fusion node

On receipt of a REVERSE_PATH from a host r:
   If the first time
      set up state for r
   If timer pending for r
      cancel timer
   start timer for r with value TCP-M_TMOUT
   forward it to parent

On timer expiry for host r:
   Alive[r] = FALSE
   duplicate = 0
   send_ack()

procedure send_ack()
   temp = min(Seq[s]) ∀ hosts s such that Alive[s] = TRUE
   If (temp > last_acked or 0 < duplicate < MAX_DUPLICATE)
   (* fresh or duplicate Ack *)
      last_acked = temp
      send acknowledgment with sequence number = last_acked,
         source = my address, and
         destination = parent
```

Figure 6: TCP-M: Determining failed hosts

When a timer expires for a certain host, it is noted as "dead". Also, the Ack table is checked to see if it allows a new acknowledgment to be sent. Thus, after a brief intermission data transfer resumes. When forwarding acknowledgments, only the hosts that are "alive" are considered. The modified *send_ack*() procedure is outlined in Figure 6. It also requires some more state at FNs.

TCP-M also handles routing topology changes automatically. *REVERSE_PATH* and normal

acknowledgments to data packets are sent hop-by-hop through the fusion nodes. Thus, the protocol works even if the network topology changes and packets are re-routed as a result.

From the description of connection termination above in Section 4.4, the application also gets to know which receivers successfully received all packets sent in the connection.

5. Properties of TCP-M

In this section we discuss some of the important characteristics of TCP-M. In the following Section 6 we verify some of these properties with simulation experiments.

5.1. TCP-like Congestion Control

Our most important goal in designing TCP-M was to come up with a multicast protocol which reacts to congestion in the network like a unicast connection. Unicast connections form a bulk of the network connections in the Internet. Further, according to [Bra96], about 80% of all data flowing through the Internet is generated by TCP. Thus, our goal was to design a protocol which reacts to congestion like TCP. We discussed earlier in Section 3 why a multicast protocol should do so. We achieve this as a direct consequence of our design of TCP-M.

The embedded TCP in the TCP-M source transport protocol is completely transparent to the fact that the data packets are multicast to several receivers. TCP receives acknowledgments from a single address (due to Ack fusion). Thus, it calculates round-trip times and timeout values as usual. It also reduces or increases the sender congestion window based on the feedback it receives from the acknowledgments.

In the next section, by a simulation experiment, we demonstrate how the behaviors of TCP-M and unicast TCP are alike in this respect.

5.2. Bottleneck Bandwidth

As mentioned earlier, one obvious consequence of fusing acknowledgments is that the TCP-M source would be slowed down to the speed of the slowest receiver. Thus, all the receivers would experience the same data transfer rate irrespective of their bandwidth to the source. While this may be considered a disadvantage — the important point to note here is that in the time required to transfer a file to the slowest receiver, we have also transferred it to all the other receivers. This saves us a lot of network resources. Besides, TCP-M is intended for file transfer applications which are not time-critical, but where reliability is important.

5.3. Incrementally Deployable

TCP-M requires the intermediate routers to be capable of doing Ack fusion. However, we designed TCP-M such that it would work even if only the source host implements Ack fusion. *PATH* and *REVERSE_PATH* messages (described in Section 4.1) set up the state required at fusion nodes. Figure 3 illustrates by an example how this is done. In Figure 2 we see how TCP-M works even if not all the routers are fusion nodes. If only the source implements Ack fusion, TCP-M becomes similar to SCE [TA95] since all the acknowledgments are fused only at the source.

5.4. Complexity at Fusion Nodes: Memory and Computation

Each fusion node requires $O(3n+3) \equiv O(n)$ (from Figure 4 and Figure 6) memory for every multicast connection (source:group pair), where n is the number of downstream fusion nodes. If all the routers are fusion nodes, then n is the out-degree of a router in the corresponding multicast tree minus one. Thus, it is independent of the size of the multicast tree or the number of receivers. This makes TCP-M very scalable. However, in the extreme case, if only the source is a fusion node, n is the total number of receivers.

Refer to procedure *register_ack*() in Figure 4. Assuming hash tables or content addressable memory (CAM), this would take $O(1)$ time. Sending Acks (Figure 6) takes $O(2n+1) \equiv O(n)$ time to determine if an acknowledgment needs to be sent, and to find the appropriate sequence number. Again, if all the routers are fusion nodes, n is relatively small and therefore this is very scalable.

5.5. Scalability

From Section 5.4 above, the memory and computation required at a router to do Ack fusion is very reasonable and is independent of the total number of receivers in the corresponding multicast tree[3]. This makes TCP-M very scalable to large multicast groups. One potential disadvantage of TCP-M is that a single packet loss anywhere in the network results in a retransmission on the whole multicast tree. As a possible future extension to TCP-M we could add local retransmissions. However, with links becoming more and more reliable, packet losses due to data corruption should be rare. Most losses occur due to congestion at the routers, and with improvements over the years, TCP can take care of congestion very well. Using TCP as the transport protocol for TCP-M would thus be very conducive for the performance of TCP-M.

In our simulation experiments described below (Section 6) we tested TCP-M on networks upto 250 nodes and the performance was very encouraging.

6. Simulating and Testing TCP-M

We implemented TCP-M in `Ns` [UCB98] and conducted simulation experiments to evaluate TCP-M and verify its correctness. In this section we present the results of our experiments.

Figure 7 is the topology we used to test TCP-M. It is a ten node topology with one source and 5 end-nodes. For all our experiments we started data sources from S. The receivers were at $D1$ through $D5$. All the nodes were capable of fusing acknowledgments. The multicast routing protocol used for our experiments was DVMRP [WPD88]. The embedded unicast transport protocol at the source was TCP Vegas [BOP94] and the transport protocol at the receivers were modified TCP sinks which were capable of interpreting $PATH$ messages from TCP-M sources and sending $REVERSE_PATH$ messages periodically. The links in the topology had bandwidth and delay parameters as shown.

We used the above topology to verify the correctness of the protocol. Using it we tested how TCP-M behaves in co-existence with unicast protocols and in the the event of network failures. Link $R4 - D4$ provides an alternate path from S to $R4$ which we used to see how TCP-M responds to routing topology changes. $R4 - D4$ is not used in the multicast tree rooted at S until $R2$ failed. The multicast tree is shown in bold lines.

We also used some much larger topologies to test how TCP-M works for large networks. We present the results later.

[3] If all the intermediate routers are fusion nodes.

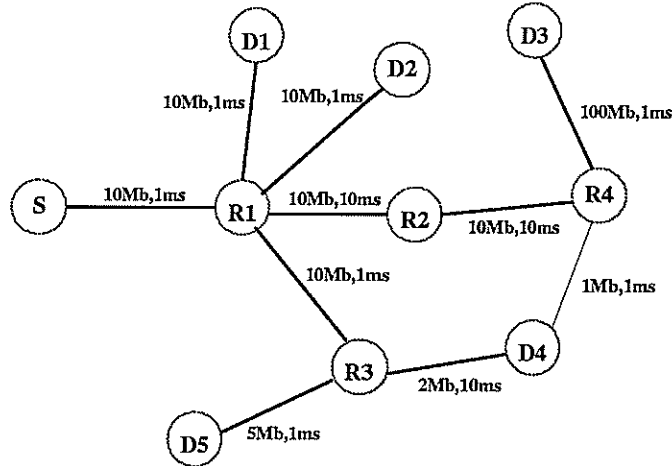


Figure 7: Topology used to test TCP-M

6.1. Single Multicast Connection

We studied a single multicast connection from an FTP source at S to $D1$, $D2$, $D3$, $D4$ and $D5$. Figure 8 shows the packet sequence number plot for the sender TCP. The multicast tree in Figure 7 had links of different bandwidths and delays, but the overall throughput is a shade less than 2Mbps, the bandwidth of the bottleneck link $R3 - D4$. TCP packets are 1000 bytes each in this and all following experiments.

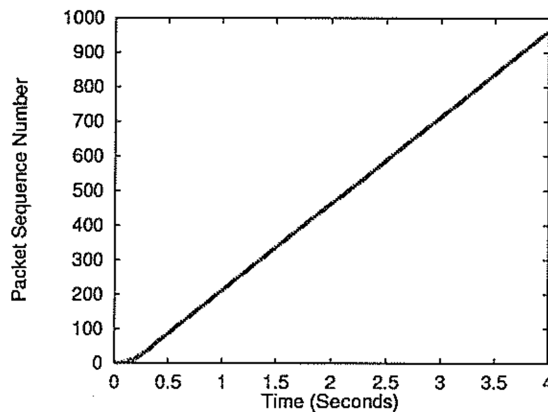
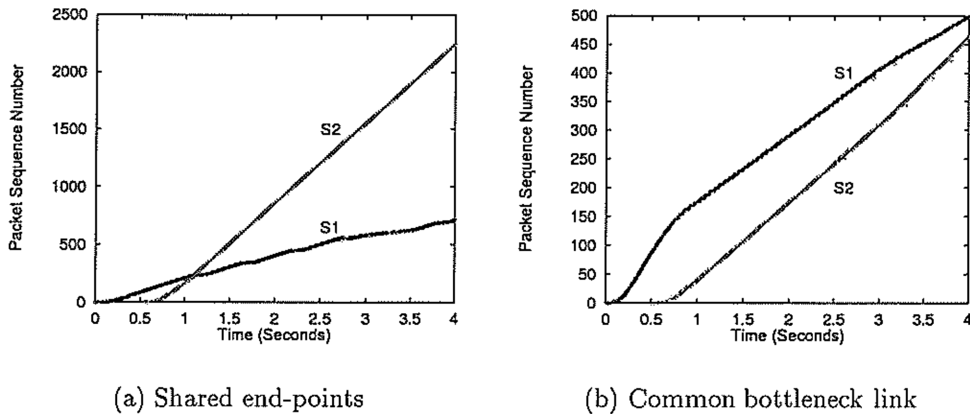


Figure 8: TCP-M for a single connection

6.2. Multiple Connection Scenarios

Here we simulated two simultaneous multicast connections on the same network from the same source S. For the experiment in Figure 9(a), $S1$'s receivers were $D1$, $D2$, $D3$ and $D4$. $S2$'s receivers were $D1$, $D2$, $D3$ and $D5$. Figure 9(a) depicts the sequence numbers transmitted by TCP-M. The bottleneck link for $S1$ is link $R3 - D4$ and while that for $S2$ is $R3 - D5$. The TCP sources obtain the bandwidth allowed by their bottleneck links. Shared links have enough bandwidth to allow both connections. $S2$ was started 0.5 seconds after $S1$.



(a) Shared end-points

(b) Common bottleneck link

Figure 9: Two simultaneous multicast connections

In the next experiment (Figure 9(b)), the multicast connections shared the same bottleneck link. Both $S1$ and $S2$ had the same set of receivers: $D1$, $D2$, $D3$, $D4$ and $D5$. Link $R3 - D4$, shared by both the connections, has a bandwidth of 2 Mbps. However, after a while, both the multicast connections obtain a bandwidth of about 1 Mbps. $S2$ was started a little after $S1$ so that the plots would be easily distinguishable.

These experiments demonstrate that multiple TCP-M connections react to each other like unicast TCP connections and share bandwidth among each other fairly.

6.3. Co-existence with Unicast Traffic

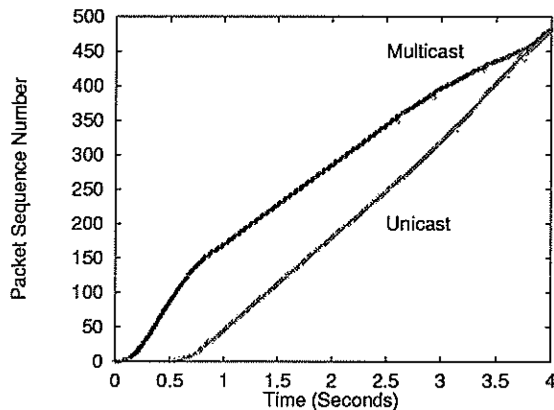


Figure 10: TCP-M co-existing with a unicast TCP connection

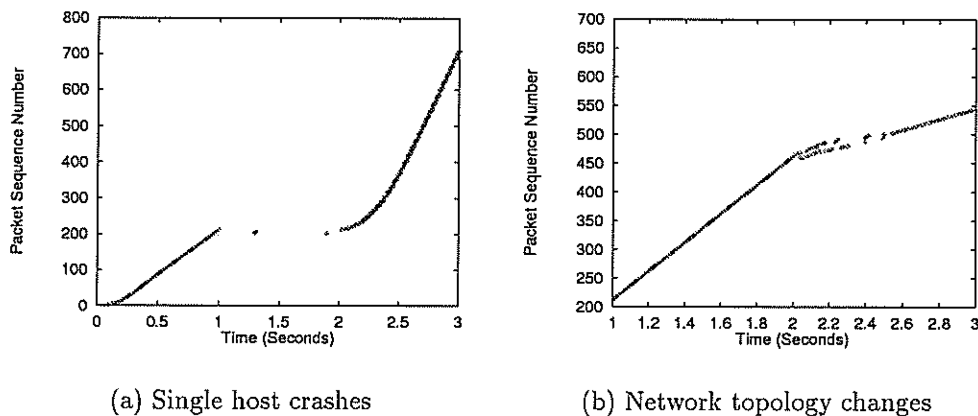
In this experiment, we studied a multicast connection in co-existence with a unicast connection. Consider the topology in Figure 7. For this experiment, we started a unicast FTP from S to $D4$ and a multicast FTP from S to $D1$, $D2$, $D3$, $D4$ and $D5$. The sequence number plot is shown in Figure 10. The multicast connection and the unicast connection shared 2 Mbps among themselves (links $R3 - D4$ is the bottleneck). Each obtained about 1 Mbps. The sources were started at staggered times so that the plots would be distinguishable.

An interesting argument at this point would be that multicast connections should receive a share of throughput proportional to the number of downstream receivers[4]. Thus, the multicast connection above receives the same throughput as the unicast connection since there is only one downstream receiver ($D4$) for router $R3$. However, in another experiment, $S - R1$ was the bottleneck neck, but still the two connections shared bandwidth equally among themselves. In order to have multicast connections receive a share proportional to the number of receivers, a router should know the number of receivers for each multicast connection passing through it. This is clearly a lot of overhead and leads us to the classical problem of fair queuing.

This experiment proves that TCP-M satisfies our most important design goal — TCP-like behavior and TCP-friendly when reacting to network congestion.

6.4. Network Failures and Routing Changes

The following two experiments were performed to verify how TCP-M reacts to network failures and routing topology changes. In Figure 11(a), the multicast source is transferring a file to all the five receivers in Figure 7. At 1.0 seconds, represented by the vertical line, we deliberately crashed router $R3$. For the purpose of this simulation, the value of *RCVR_TMOUT* was 1 second and *TCP-M_TMOUT* was 2 seconds (see Section 4.5). The plot in Figure 11(a) depicts how TCP-M reacts to crashing of $R3$. TCP-M, or rather the embedded TCP, retransmits its last window of packets at 1.3 and 1.9 seconds. $R1$ received the last *REVERSE_PATH* message from $R3$ at 0.0235 seconds. Therefore, its timer expires at 2.0235 seconds and thus data transfer resumes. Approximately one second is lost due to the failing of $R3$. This, response to failure can be hastened by decreasing *RCVR_TMOUT* and *TCP-M_TMOUT*.



(a) Single host crashes

(b) Network topology changes

Figure 11: Network failures and TCP-M

The next experiment is to demonstrate that TCP-M is transparent to routing topology changes. In the topology in Figure 7 $R4$ has an alternate route (through $D4$ and $R3$) to get to $R1$. However, this route is not used because it is sub-optimal. At time 2.0 seconds in the simulation, represented by the vertical line, we crashed $R3$ deliberately. The plot in Figure 11(b) shows that TCP-M transfer continues. The plot is zoomed to an x-axis of 1.0 to 3.0 seconds. There are a few retransmissions because of the sudden increase in the round-trip time when $R3$ goes down. The data transfer rate drops because a sub-optimal route is being used and it is the bottleneck. As stated in Section 4.5

[4] We thank Christos Papadopoulos for bringing it up.

this is because $REVERSE_PATH$ and data-acknowledgments are sent hop-by-hop through fusion nodes. In this experiment $R3$ was not a fusion node (so that we could crash it and still recover receivers reachable through $R3$ via an alternate path).

6.5. Fast Retransmit and Fast Recovery in TCP-M

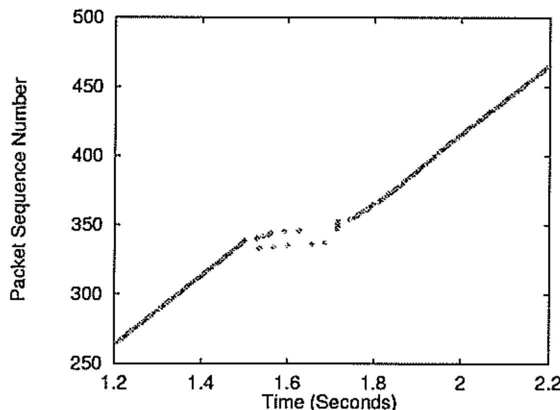


Figure 12: Fast retransmit and fast recovery in TCP-M

The following experiment is to demonstrate that we have preserved the fast retransmit and fast recovery mechanisms of TCP in our design of TCP-M. In Figure 12 we show a single multicast connection from S (in Figure 7) to all the receivers. We make receiver $D1$ deliberately lose packets numbered 333 through 337 (at about 1.5 seconds) and let TCP-M recover from it. In Figure 12 we can see that TCP-M retransmits packets 333 through 337 and after a brief period the data transfer continues as before. By studying link traces we also observed the duplicate Acks flowing back to S. The behavior here is however highly dependent on the particular network and which receiver or receivers lose packets. If the duplicate Acks are received by an FN before the previous packets have been acknowledged, they are wasted and the sender ultimately times out.

6.6. Bottleneck Bandwidth in Large Network

We simulated TCP-M over some large networks upto 250 nodes. We used the GT-ITM network topology generator [CZ97] from Georgia Tech to generate random networks. The performance of TCP-M was as expected — TCP-M sends data at the bottleneck speed as defined by the particular network. Figure 14 shows the sequence number plot for a multicast connection in a random 100 node topology. The parameters file used to generate the topology is given in Figure 13.

```
# random topology of 100 nodes
# <method keyword> <number of graphs> [<initial seed>]
# <n> <scale> <edgemethod> <alpha>
geo 1 23
100 50 3 0.03
```

Figure 13: Parameters used to generate a random 100 node topology

We simulated a multicast connection from node 0 to all 54 reachable nodes. The bandwidth of the links were 100 Kbps each. The link delays varied from 2 to 57 ms. The simulation time was 1 minute. Since TCP packets were 1000 bytes each the total throughput observed by the multicast connection was about 94.3 Kbps. It takes longer for TCP-M to reach the steady state in these larger networks because of higher link delays, but eventually it reaches the throughput allowed by the link bandwidths. When we changed the throughput of one link in the network to 50 Kbps, the total throughput obtained by TCP-M was about 49.2 Kbps.

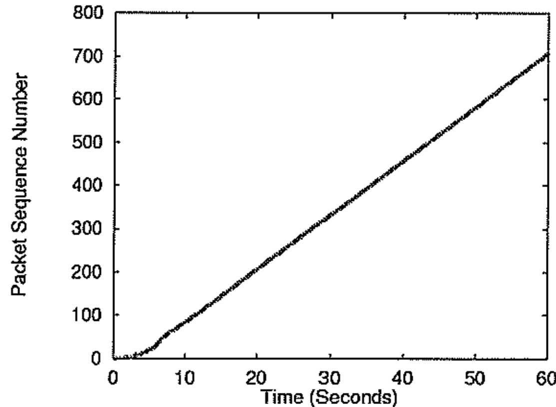


Figure 14: TCP-M on a random 100 node topology

6.7. Benefits of Ack Fusion

In the following experiment, we studied the effect of reducing the number of hosts capable of Ack fusion. We generated binary trees with random link bandwidths and delays[5]. Then we simulated a single multicast source from the root to all the other nodes. The link bandwidths were 500 Kbps each and the delays varied from 0 to 10 ms. We ran our experiment on networks of size 63 and 127 nodes[6]. All the nodes, but the root, in the tree were receivers. The simulation time was 1 minute.

We wish to study the effect of Ack implosion as the number of fusion nodes are reduced. However, for our simulator packet processing time at a node is zero. Therefore, we had to change our simulation parameters so that acknowledgments would create congestion in the network as their number increases. Specifically, the TCP packet size was changed to 40 bytes[7] for this experiment — so as to increase overhead due to acknowledgments.

Table 1: Throughput (in Kbps) for binary tree topologies as the number of fusion nodes is reduced

Number of nodes	Number of fusion nodes						Unicast connection
	127	63	31	15	7	only source	
127	142.9[8]	142.9	138.9	13.8	—	0.47[9]	145.2
63	—	173.2[8]	173.2	164.5	18.74	1.59	174.3

[5] We used a tree topology because it could easily lead to the problem of Ack implosion as the number of FNs are reduced - which we intend to observe here.

[6] We could not use a larger network because of inherent limitations in our simulator — ns.

[7] Same as acknowledgment packet size.

In Table 1 we see that the throughput reduces, sometimes considerably, as the number of FNs is reduced in the network. The fusion nodes were the topmost nodes in the tree (including the root node). For the 127 node topology, the throughput drops by about 37% when the number of FNs is reduced from 63 to 31. The throughput for 50% and 100% FNs are the same because adding Ack fusion capability to the leaf nodes in a binary tree topology do not make any difference — no acknowledgments are fused at leaf nodes. For the case where the source is the only fusion node, congestion due to Ack implosion is so bad that several hosts were timed out because their *REVERSE_PATH* messages were dropped. In fact, for the 127 node topology, TCP-M aborted the connection midway dut to high network delays. Thus, we see a very poor throughput. The decrease is not very consistent because the results depend on the particular topology. However, the general trend in Table 1 is that the throughput and efficiency decreases as the number of fusion nodes are reduced.

We also show the throughput of a unicast TCP Vegas connection from the root to one of the leaf nodes. As we can see the throughput obtained by TCP-M is quite close to the throughput obtained by the unicast connection — the small difference can be attributed to the TCP-M protocol messages.

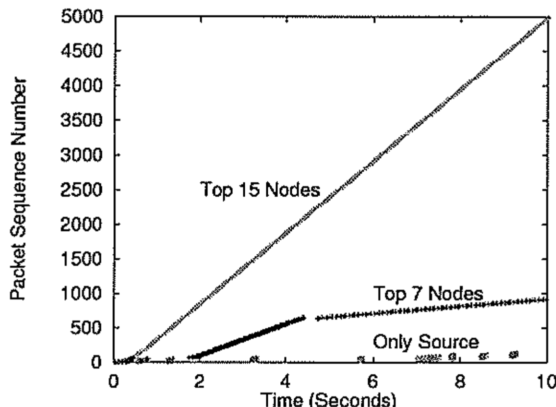


Figure 15: TCP-M on a binary tree topology of 63 nodes as the number of FNs are varied

In Figure 15 we show the sequence number plots for three cases for the 63 node topology. We only show the first 10 seconds of the simulation for clearer understanding. For the case where the source is the only FN, it leads to a large number of packet drops and consequently the very low throughput as shown in Table 1. having only 7 fusion nodes also result in packet drops, but TCP-M (or, rather the embedded TCP) recovers from it quite well and is successful in obtaining a steady state. Clearly the performance is better for 15 fusion nodes.

7. Applicability of TCP-M

We intend to use TCP-M for applications which require one to many reliable file transfers. For example, a web server distributing files to its cache proxies or mirror sites, a software vendor distributing updates to its clients or synchronizing software versions among different machines in a network. Another possible application is sending electronic mail to multiple receivers. Currently, all these use separate unicast connections to the different machines. Replacing it by TCP-M would reduce the overall time to a fraction. Further, it would reduce the overall network bandwidth used

[8] The rest of the bandwidth being used up by the Acks and TCP-M protocol messages.

[9] Connection aborted midway due to network delays.

(multicast reduces the total number of packets). A file transfer with TCP-M would take as long as the longest unicast file transfer, but in the process, all the other machines would have received the file too.

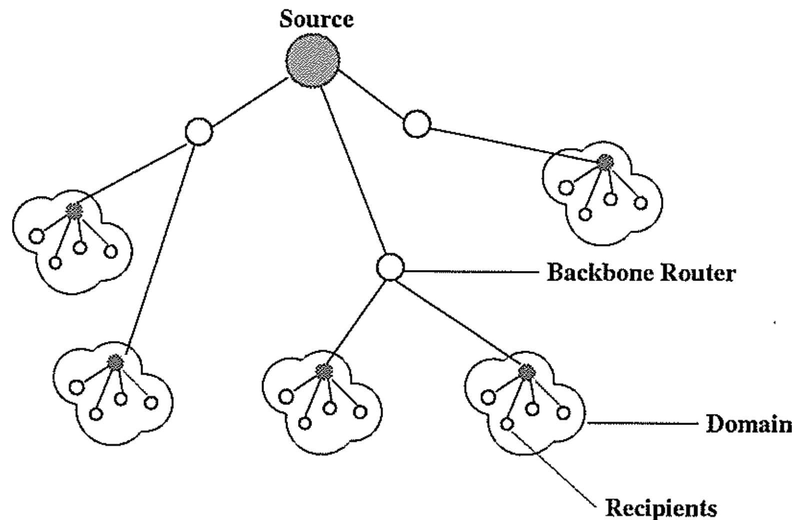


Figure 16: Ack fusion can be restricted to intra-domain routers

The design of TCP-M provides us with a lot of flexibility in choosing fusion nodes in order to improve throughput and efficiency. For example, for a transfer across the Internet, we can avoid modifying backbone routers to do Ack fusion. Even if we could change backbone routers, we would not want to because backbone routers are often overloaded and the extra effort of doing Ack fusion would further slow them down. Only the intra-domain routers and the source would fuse acknowledgments. See Figure 16 for an illustration. Since the number of backbone routers are in hundreds while the number of recipients could be in the order of tens of thousands or more, the intra-domain routers and the source would have to fuse about hundreds of Acks.

As another possibility and an extension to TCP-M, we could have special hosts (not necessarily routers) who could fuse acknowledgments. Thus, we could relieve the routers of the overhead of Ack fusion.

8. Summary

In this paper, we proposed TCP-M, a reliable multicast transport protocol which can use TCP as the upper level transport layer protocol to do multicast. The multicasting is done by IP Multicast and the reliability is assured by acknowledgments and retransmissions. Ack implosion is prevented by fusing them at the routers. The important advantages of TCP-M are that it is very scalable and TCP-friendly. The congestion control in TCP-M is exactly like TCP: multicast and unicast connections thus behave similarly in response to congestion. This is very important because TCP mechanisms have been perfected over the years and we know it works. Further, it would allow unicast and multicast connections to co-exist fairly. TCP-M is very easily deployable — it would work even if only the source is capable of Ack fusion. We simulated TCP-M in Ns and performed some experiments which support our arguments.

References

[AFM92] S. Armstrong, A. Freier, and K. Marzull. Multicast Transport Protocol. *Network Working Group, RFC 1301*, February 1992.

[BOP94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM'94 Conference, London*, pages 24–35, August 1994.

[Bra96] L. S. Brakmo. *End-To-End Congestion Detection And Avoidance in Wide Area Networks.* PhD Dissertation, Department of Computer Science, University of Arizona, 1996.

[Bro97] Frank Brockners. Notes on FEC supported Congestion Control for One to Many Reliable Multicast. *Working draft*, September 1997.

[CZ97] Ken Calvert and Ellen Zegura. GT Internetwork Topology Models (GT-ITM). *http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html*, May 1997.

[Dee89] S. Deering. Host Extensions for IP Multicasting. *RFC 1112*, August 1989.

[Eri94] H. Eriksson. MBone: The Multicast Backbone. In *Communications of the ACM*, volume 8, pages 54–60, August 1994.

[FJM+95] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of ACM SIGCOMM '95*, pages 342–356, August 1995.

[GV97] R. Ghosh and G. Varghese. Symmetrical Routes and Reverse Path Congestion Control. Technical Report TR-97-37, Department of Computer Science, Washington University, St. Louis, September 1997.

[LP96] John C. Lin and Sanjoy Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of IEEE INFOCOM '96*, pages 1414–1442, March 1996.

[MJV96] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proceedings of the ACM SIGCOMM '96 Conferencs*, pages 117–130, August 1996.

[Pos80] J. Postel. User Datagram Protocol. *RFC 768*, August 1980.

[Pos81] J. B. Postel. Internet Protocol. *RFC 791*, September 1981.

[RBR98] I. Rhee, N. Balaguru, and G. N. Rouskas. MTCP: Scalable TCP-like Congestion Control for Reliable Multicast. Technical Report TR-98-01, Department of Computer Science, NCSU, January 1998.

[RJ88] K. K. Ramakrishnan and Raj Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer. In *Proceedings of the ACM SIGCOMM '88 Symposium*, pages 303–313, August 1988. Also available as DEC-TR-508, Digital Equipment Corporation.

[RP94] J. Reynolds and J. B. Postel. ASSIGNED NUMBERS. *RFC 1700*, October 1994.

[SFLT98] Tony Speakman, Dino Farinacci, Steven Lin, and Alex Tweedly. PGM Reliable Transport Protocol. *Internet Draft: draft-speakman-pgm-spec-01.txt*, January 1998. Expires July 1998.

[Sim95] W. Simpson. IP in IP Tunneling. *RFC 1853*, October 1995.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, The Protocols.* Addison-Wesley Professional Computing Series, 1994.

[TA95] R. Talpade and M. H. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proceedings of the 15th IEEE Intl Conf on Distributed Computing Systems, Vancouver*, June 1995.

[Tan81] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, Englewood Cliffs, N. J., 1981.

[UCB98] UCB/LBNL/VINT. *Network Simulator - ns version 2.* http://www-mash.cs.berkeley.edu/ns/, 1998.

[VHC] L. Vicisano, M. Handley, and J. Crowcroft. B-MART, Bulk-data (non-real-time) Multiparty Adaptive Reliable Transfer Protocol. *Department of Computer Science, University College London.*

[Vic98] L. Vicisano. *Notes on a cumulative layered organization of data packets accross multiple streams with different rates.* http://www.cs.ucl.ac.uk/staff/L.Vicisano/layer.ps, March 1998.

[VRC98] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of the IEEE INFOCOM '98 Conference*, March 1998.

[WPD88] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. *Network Research Group, RFC 1075*, November 1988.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, September 1993.