RMCM: Reliable Multicasts for Core-based Multicast Trees

Yuan Gao, Ye Ge, and Jennifer C. Hou

Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210
{*gaoy,gey,jhou*}*@ee.eng.ohio-state.edu*

Abstract

Reliable multicast is one of the important and challenging problems that must be effectively resolved before multicast applications can be fully deployed on the Internet. Most existing reliable multicast schemes focus on source-based *multicast trees. They either cannot be directly deployed in* core-based *multicast trees or may render sub-optimal performance when directly deployed. The major difficulty lies in that an on-tree router on a core-based tree does not know, due to lack of the per-source information, how to direct a retransmission request (NAK) toward the sender or some designated replier router/host.*

In this paper, we design and evaluate a reliable multicast scheme for core-based *multicast trees, called* RMCM. *RMCM closely emulates the optimal recovery scenario achieved in the turning point approach [14]. Specifically, we define new IP options to carry path information in both data packets and NAKs and design a simple, yet effective scheme to facilitate routers on a core-based tree to properly direct NAKs/replies, even in the existence of clouds (of reliable-multicast-incapable routers). To conduct local recovery in the case of group membership and topology changes, we devise a mechanism that selects designated hosts (called* repliers*) which are most likely to have the requested packet and to which NAKs will be forwarded. We also devise a delayed ACK approach so that both sources and repliers can purge data packets that have been received by all the group members. The event-driven simulation study indicates that RMCM achieves better performance than most existing repair-based reliable multicast schemes, in terms of NAK implosion control, recovery latency, recovery isolation, and capability to deal with clouds. Finally, our experiences with implementing RMCM on FreeBSD 2.2.8 show that the add-on overheads to routers are reasonably small.*

1 Introduction

Reliable multicast is an important and challenging problem that differs in many aspects from reliable unicast in point-to-point packet-switched networks. Reliable transport in unicasts is achieved by having each receiver send an ACK or NAK (in the case of detecting data loss) to the sender which then retransmits lost packet(s). This error control method can not be directly deployed in multicasts because of the following reasons: first, simultaneous NAKs from a large number of receivers can lead to sender and network overload, causing the famous NAK implosion problem [5, 10, 14]. Second, if only a small part of receivers experience data loss, it is inefficient to have a sender multicast the lost data to the entire multicast group. Instead, recovery should be isolated to members that experience data loss. To this end, it has been proposed that the task of processing NAKs and retransmitting lost packets be shared among group members. Several mechanisms are devised to suppress duplicate NAKs and replies and to designate certain group members to handle NAKs, so as to mitigate the NAK implosion problem and to reduce recovery latency (measured as the time from the instant a loss is detected until a reply is received by the group member that suffers from data loss). Moreover, all the objectives stated above should still be fulfilled when group members join/leave dynamically. The main intent of this paper is to design and evaluate a repair-based reliable multicast scheme for *core-based* multicast trees that avoids NAK implosion and duplicate replies, reduces recovery latency, achieves recovery isolation, and is adaptive to dynamic membership changes.

Existing reliable multicast schemes can be roughly classified into two categories: (i) reactive repair-based approaches and (ii) proactive FEC-based approaches. In repair-based approaches, the sender or designated router/host sends lost data packets reactively in response to NAKs. They can be further classified into those which are non-TCP-based, such as SRM [5], PGM [18], hierarchical schemes (best represented by LBRRM [8], TMTP [22], and RMTP [12]), *Turning Point* (TP) [14], *Search Party* [4], and ARM [10], and those which extend TCP to accommodate multicasts, such as IRMA [9] and MTCP [16]. In a radical departure from repair-based approaches, FEC-based approaches, such as those reported in [13, 17, 2], combine forward error correction (FEC) with automatic repeat request (ARQ), and send repair packets proactively before they are required. Specifically, data is transmitted in blocks, each of which consists of k data packets and $n-k$ repair packets. Each block is so encoded (by, for example, Reed-Solomon code) that as long as a receiver receives k out of n packets in that block, it will be able to recover all n packets. Initially the sender transmits k data packets and additional i repair packets, where $0 \leq i \leq n-k$. Only when a receiver fails to decode the entire block will it unicast a NAK (requesting additional repair packets) to the sender. The sender responds to the NAKs by sending an appropriate number, ℓ, of repair packets, where ℓ is determined so as to satisfy all the receivers' requests.

FEC-based approaches do not require router support and reduce the delay in data delivery under certain circumstances. However, these approaches incur bandwidth overhead of transmitting i additional packets per block. Determination of the optimal value of i is not trivial, as it depends (among other things) on the data error rate, and is a tradeoff between the level of proactivity desired, the bandwidth overhead incurred, the number of NAKs received at the sender, and the recovery delay incurred.

The work reported in this paper was supported in part by DARPA under Contract No. N66001-97-C-8526 and by NSF under Grant No. ANI-9903160.

Also, as NAKs are unicast to the sender, the NAK implosion problem may become a concern in the case of bursty packet losses. In case that repair packets (sent in response to NAKs) are multicast to the entire group, recovery may not be isolated only to the area that incurs errors.

All the contemporary work in the category of repair-based approaches focuses on *source-based* multicast trees. What has not been extensively addressed (except partially in [14]) is reliable data transport on *core-based* multicast trees constructed by, for example, CBT [1], simple multicast (SM) [15], or bi-directional PIM-SM [7][1]. In a source-based tree, the root is the source and upstream/downstream routers are defined with respect to the source. Data packets originate from the source and are sent *downstream* to all the destinations via the links of the tree. An on-tree router knows on which (upstream) interface data packets come. In contrast, in a core-based tree one router for each group is selected as the core [1, 15] (or termed in [20, 7] as a rendezvous point), and a tree rooted at the core is constructed to span all the group members. The state associated with a router is with respect to a group, rather than to a pair of (group, source)[2], and the notion of upstream/downstream is defined with respect to the core. Data packets flow from a source to its parent and children, except the interface on which the data packets come. As a result, a data packet may arrive on an upstream or downstream interface. As a router does not usually record the incoming interface on which data packets are received, it does not know on which interface the source node of a lost packet can be reached when it receives the corresponding NAK.

The lack of per-source information makes PGM, ARM, IRMA, and MTCP not directly applicable to core-based trees. The hierarchical schemes and SRM do not use the per-source information to forward NAKs, and can be directly applied to core-based trees. However, as will be discussed in Section 4, all of them have their shortcomings. The TP approach (and its variation, *Search Party*) can be applied to a core-based tree by treating the tree as a source-based tree rooted at the core and having the core unicast the request to the source whenever it receives a NAK. However, as will be discussed in Section 2, this cannot totally eliminate the drawbacks that result from lack of the per-source information (e.g., certain degree of loss of recovery isolation).

In this paper, we design a repair-based reliable multicast scheme for core-based multicast trees that closely emulates the optimal recovery scenario achieved in the TP approach [14]. We define new IP options to carry path information in both data packets and NAKs and design a simple, yet effective scheme to facilitate routers on a core-based tree to properly direct NAKs/replies, even in the existence of clouds (of reliable-multicast-incapable routers). To conduct local recovery in the case of group membership and topology changes, we devise a mechanism that selects designated hosts (called *repliers*) which are most likely to have the requested packet and to which NAKs will be forwarded. We also devise a delayed ACK approach so that both sources and repliers can purge data packets that have been received by all the group members. Finally, we implement RMCM both in a Java-based network simulation environment, *JavaSim*, and on the *FreeBSD* 2.2.8 kernel, and conduct a performance study to validate the proposed design.

The rest of the paper is organized as follows. In Section 2, we describe the network model under consideration, and summarize the TP approach. In particular, we give an example that demonstrates why the TP approach renders sub-optimal performance in core-based trees (e.g., loss of recovery isolation). In Section 3, we present RMCM and elaborate on each of the components. We evaluate RMCM both empirically and via simulation in Section 5. We summarize the related work in Section 4. The paper concludes with Section 6.

2 Background Materials

2.1 Network Model

The network model under consideration is a packet-switched network that provides unreliable, best-effort services. Packets may be delayed, lost, or duplicated. Data packets of a multicast session are numbered in sequence. An end host detects data loss by detecting gaps in the sequence numbers of the data packets received. When data loss is detected, an end host requests retransmission of the lost data packet by sending a NAK.

We consider the most general many-to-many multicast paradigm in which each member in a multicast group can be a source as well as a destination. A core-based multicast routing protocol, e.g., CBT, SM, or bin-directional PIM-SM is used to construct such a core-based distribution tree that spans all the group members, while the Internet group membership protocol (IGMP) [3] is used between a local router and the end hosts on its directly attached subnet for member join/leave.

2.2 The Turning Point Approach

The TP approach [14] uses three strategies (selection of repliers, defining turning points, and directed multicast for retransmission) to closely approximate an optimal recovery scenario. In the optimal recovery scenario, the router directly below a data loss sends a request to the router immediately above the loss and the router immediately above the loss multicasts the lost packet to the affected branch. In the TP approach, each router selects one of the downstream links as a replier link.[3] A receiver that detects a loss sends a request to the local router. Routers forward all incoming requests to the replier link, except the request that arrives on the replier link, which is forwarded upstream. If the router has no downstream replier links, the request is also forwarded upstream. If a request arrives on a downstream link other than the replier link, the router becomes the *turning point*. The router inserts into the request its address and the identifier for the link on which the request arrived. The request is then forwarded downstream toward the replier. A replier with the requested data extracts from the request the address of the turning point, and unicasts a reply directly to the turning point which then performs a directed multicast to the subtree rooted at the turning point.

Example 1: Consider the multicast tree in Fig. 1. Suppose data loss occurs on the link between R_2 and R_5 (marked with an X). Hosts E_0 through E_6 detect the loss and send requests. As shown in Fig. 1, the requests sent by E_1, E_3, and E_5 are

[1] In the original PIM sparse mode [20], there are two possible trees for traffic from a particular source for a group: the shared tree and the source tree. To prevent loops, the shared tree is designed to be unidirectional, i.e., a source has to encapsulate its data packets and unicasts them to the core. As a result, existing source-based reliable multicast schemes suffice to provide reliable data transport.

[2] This is one dominating factor that makes core-based multicast routing more scalable.

[3] The only exceptions are (i) routers with only one downstream link, which selects the upstream link as the replier link; and (ii) routers adjacent to the source, which select the source link as its replier link.

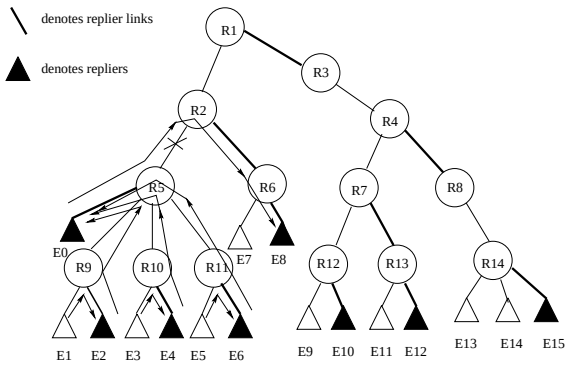


Figure 1: A core-based multicast tree that employs the turning point reliable multicast approach.

forwarded to E_2, E_4, and E_6, respectively, while those by E_2, E_4, and E_6 are forwarded upstream until they reach R_5 which then forwards them to E_0. The request from E_0 is forwarded upstream by R_5 to R_2 which then forwards it downstream to the replier, E_8. (Note that R_2 is the turning point that forwards a request downstream to a replier link.) As E_8 has the requested data, it unicasts the reply directly to R_2 which then multicasts the requested data along the left link to the affected area. □

Now the question is whether or not the TP approach can be applied to core-based multicast trees. Note that lack of the per-source information in the routers in a core-based tree implies that routers can no longer distinguish upstream and downstream links with respect to a source, and thus may not direct requests appropriately. To bypass the above problem, Papadopoulos *et. al* [14] proposed to treat a core-based multicast tree as one that is rooted at the core, and apply the three strategies to such a tree. If a request message reaches the core, the core encapsulates the request and unicasts it to the source. The source in turn performs a directed multicast to the turning point as before. Although this extension is simple and straightforward, as demonstrated in the following example, it is subject to loss of recovery isolation (that results from lack of the per-source information).

Example 2: Consider the core-based multicast tree in Fig. 2. Suppose the TP approach is used to recover the data loss that occurs on the link between R_5 and R_8. The request from the affected subtree (i.e., the tree rooted at R_8) is forwarded upstream until it reaches the core which then unicasts it to the source R_{12}. As the source has the requested data, it unicasts a reply to the core which then performs a directed multicast on the tree rooted at the core R_1. As a result, several routers (i.e., R_2, R_4, R_5, R_9, R_{12}, R_{13} and R_{14}) that do not suffer from data loss receive the reply. That is, recovery is not isolated only to the affected area. □

3 RMCM

3.1 Overview

In RMCM, each router selects, for *each* of its interfaces ℓ, the *most reliable*[4] interface among all the interfaces except ℓ as the replier link ℓ_r. A replier link ℓ_r of link ℓ will be used as the default link to forward a NAK *if the lost data packet originally arrived on link* ℓ. Note that the selection of a replier link is *not* source-specific — all the sources that can be reached on an interface ℓ share the same replier link, ℓ_r — and is realized

[4] To be defined in Section 3.3.

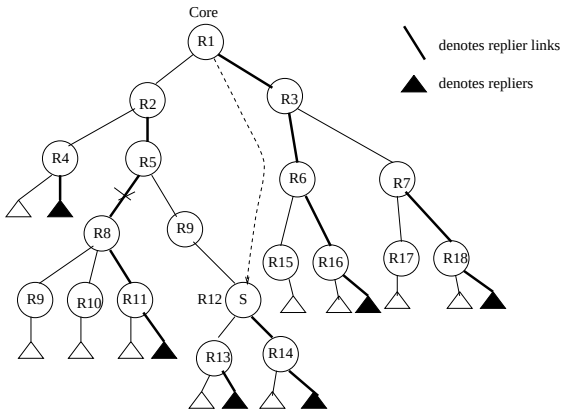


Figure 2: A core-based multicast tree that employs the turning point reliable multicast approach.

with modest modification of host membership report messages in IGMP and soft state refresh messages in the core-based multicast routing protocol.

An IP option, called *path_info*, is defined to carry the path information. A source host includes the *path_info* option in the IP header of a data packet periodically or upon some receiver's request. Upon receiving a data packet with the *path_info* IP option, an on-tree router includes in the option the identifier of the interface on which the data packet arrives. When a data packet with a *path_info* option arrives at a receiver, the receiver stores *path_info* for the (source, multicast group) pair.

When a receiver detects a data loss, it sends to its local router a NAK that contains the sequence number(s) of lost packet(s) and the path information, *path_info*, for the (source, multicast group) pair. If no *path_info* is available at this time, the receiver may unicast a special request to the sender, asking it to include *path_info* in the next data packet sent. When a router R receives a NAK, it is able to identify the interface on which the lost data originally arrived by retrieving information from *path_info*. (The case in which router R cannot identify the correct interface because a NAK is erroneously forwarded to router R as a result of route/topology change will be discussed in Section 3.4.) Then, router R directs the NAK using a mechanism similar to TP. In particular, if router R decides that it becomes the turning point (with respect to the source), it inserts into NAK the necessary information so that the replier knows where to forward the reply. A replier with the requested data unicasts a reply to the turning point which then multicasts the requested data only to the affected subtree.

As an auxiliary (and independent) function, RMCM also includes a mechanism to properly notify the sender and the repliers of when data packets can be purged (as they have been received by all group members). Succinctly, receivers send *delayed ACKs* periodically. Similar to NAKs, ACKs include a *path_info* IP option, are intercepted by an on-tree router, and are appropriately aggregated: the on-tree router uses the *minimum* of all the received sequence numbers from its downstream routers (with respect to a source) as the new sequence number. The on-tree router then forwards an ACK that contains the new sequence number upstream toward the source. The source determines the minimum message sequence number prior to which messages can be safely purged and piggybacks the information in the *path_info* IP option.

Another feature of RMCM is its ability to handle "clouds"

I. When a receiver host, H, receives a data packet with the *path_info* option for the (S, G) multicast group:

Step 1. Host H stores the *path_info* for the (S, G) group.

II. Upon receipt of a *path_info_request* message or timeout for the (S, G) group, a sender host S

Step 1. Includes a *path_info* option in the next data packet to be sent.

Step 2. Resets the *path_info_update* timer for the (S, G) group.

III. When an on-tree router, R, receives a data packet with the *path_info* option, router R

Step 1. Inserts the ID of the incoming interface on which the data packet arrives, and updates the *option length* sub-field in *path_info*.

Figure 3: Procedure that handles the path information. (S, G) denotes the (source, multicast address) pair.

of reliable-multicast-incapable routers that separate two capable routers. Inside a cloud, the newly defined IP options cannot be recognized and are simply ignored, and ACKs/NAKs are simply forwarded upstream (with respect to the core). We devise a method to enable a capable router to detect whether or not a data packet/NAK has traversed a cloud and to properly direct NAKs/replies when clouds exist on the path either to the source or a replier.

3.2 Distribution of Path Information

We deal with the problem of identifying upstream and downstream interfaces with respect to a source by defining a *path_info* IP option. There are four sub-fields in the option: a one-byte field that identifies the option (*path_info*), a one-byte *option length* field that gives the size (in bytes) of the *path_info* option, a *minimum sequence number* field (for use in delayed ACK, the details of which will be given in Section 3.5), and a list (of variable length) of incoming interface identifiers.

The procedure that handles the path information is outlined in Fig. 3. Example 3 gives an example that shows how the path information is distributed. Note that if a router numbers all of its interfaces by integers, the identifier of an interface can be as small as a byte. Note that while changes to the codes of the current IP routers seem unavoidable, we have attempted to keep them as compatible with existing IP codes as possible. A new protocol number is assigned and used in all RMCM-related unicast control messages (i.e., the *path_info_update* message, the *cloud_detection* message, and the reply to the turning point) so that they can be forwarded to the RMCM modules. An IP router that does not support the *path_info* option may simply ignore it. We will discuss in Section 3.6 how we deal with clouds.

Example 3: Consider the core-based multicast tree in Fig. 4. If a sender E_{S_2} includes the *path_info* option in a data packet, the path information received by each group member is

For receivers	Intermediate routers traversed	incoming interfaces inserted
$E_{R_1}, E_{R_2}, E_{S_1}$	R_6, R_5, R_2, R_1	3, 2, 3, 1
E_{R_3}–E_{R_5}	R_6, R_5, R_2, R_3, R_4	3, 2, 3, 1, 1
E_{R_6}, E_{R_7}	R_6, R_7, R_8	3, 1, 1
E_{R_8}, E_{E_9}	R_6, R_5, R_{10}	3, 2, 1

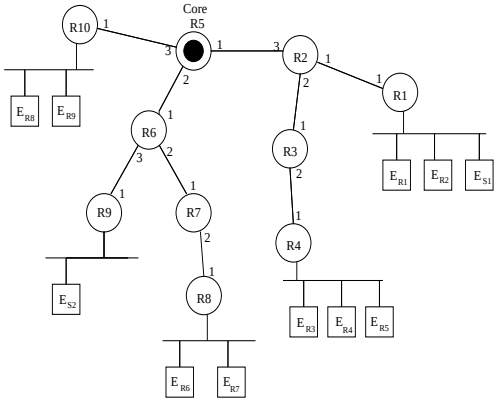


Figure 4: An example that shows how the path information is distributed.

3.3 Selection of Replier Interfaces

If a group member is willing to act as a replier, it notifies its local router when it joins the multicast group and/or when it refreshes its membership (in the IGMP host membership report [3]). A local router notifies the other on-tree routers of the fact that a host on its directly attached subnet is willing to be a replier in the periodic "tree maintenance" activities. That is, when an on-tree router sends a "keep-alive" message to its upstream router (with respect to the core), it piggybacks the replier information. Upon receiving a keep-alive message over a valid downstream interface, an on-tree router prepares a reply message with the replier information (in the upstream direction) piggybacked in the reply. As such, the mechanism of locating replier interface(s) integrates well with the underlying group membership and tree maintenance activities. Each on-tree router knows, at the expense of little overhead (a few bytes in the IGMP host membership reports and in the keep-alive/reply messages of tree maintenance activities), the interfaces on which volunteering repliers may be reached. As the replier information is kept as soft state at each on-tree router and refreshed whenever the group membership or the tree structure changes, the scheme is adaptive to membership and topology changes.

Now an on-tree router, u, selects, for each of its interfaces ℓ, a replier link ℓ_r among all its interfaces on which repliers can be reached. Let L denote the set of router u's interfaces. We consider three cases: (C1) If there does not exist any interface in $L \setminus \{\ell\}$ with volunteering repliers, router u does not select any replier link for ℓ. (C2) If there exists only one such interface in $L \setminus \{\ell\}$, router u selects that interface as the replier link. (C3) If there exist more than one such interface in $L \setminus \{\ell\}$, router u selects the *most reliable* interface as the replier link. By the *most reliable* interface, we mean the interface that receives NAKs least frequently.

To facilitate selection of such an interface in case **(C3)**, we propose to on-line collect error statistics and use a simple Bayesian parameter estimation method to estimate the NAK arrival rate on an interface. Each on-tree router keeps track of the probability, $p(j)$, that j NAKs were received on an interface in the past observing window of W time units, $0 \leq j \leq N_{max}$, where N_{max} is set to an integer large enough such that $\sum_{j=0}^{N_{max}} p(j) = 1$ holds. The parameters are then updated as

$$E(p(j)) = \frac{\alpha_j + y_j}{\sum_{k=0}^{N_{max}} (\alpha_k + y_k)}, \tag{3.1}$$

I. When a receiver host, H, detects a gap, $[n, m]$, in the sequence number(s) of data packet(s) received in the (S, G) multicast group, host H
Step 1. Prepares and sends a NAK:

Step 1.1. Retrieves the *path_info* for the (S, G) group. Includes both the *path_info* and the sequence gap $[n, m]$ in the NAK.

Step 1.2. Sends the NAK to the source S with the *Router Alert* option.

Step 2. Sets the *retransmission_request* timer.

II. Upon timeout of the *retransmission_request* timer for the (S, G) multicast group, a receiver host H

Step 1. Unicasts a *path_info_update* request to the sender host S.

III. When a group member, H, receives a NAK,

Step 1. If H has the requested data,

Step 1.1. Retrieves from the NAK (i) the IP address of the turning point and (ii) the identifier of the link to be recovered.

Step 1.2. Composes a reply message that contains (i) the requested data and (ii) the identifier of the link to be recovered.

Step 1.3. Unicasts the reply message to the turning point.

Step 2. Discards the NAK.

IV. When an on-tree router, R, receives a NAK on interface ℓ_{req},

Step 1. Router R checks if the *pass_turning_point* bit in the NAK has been set. If not,

Step 1.1. Updates the statistics on-line for calculating $E(N_r)$.
Step 1.2. Retrieves (and deletes) the interface identifier ℓ_i from the list of incoming interfaces in the NAK.
Step 1.3. If there exists a replier link, ℓ_r, corresponding to ℓ_i,

Step 1.3.1. Inserts into the *turning_point* field in the NAK its IP address and the identifier of the interface, ℓ_{req}, on which the NAK is received.

Step 1.3.2. Sets the *pass_turning_point* bit.

Step 1.3.3. Forwards the NAK on the replier interface ℓ_r.

Step 1.4. Otherwise, router R forwards the NAK on interface ℓ_i to its parent router.

Step 2. If the *pass_turning_point* bit has been set, router u forwards the NAK unchanged to its replier link corresponding to interface ℓ_{req}.

Figure 5: Procedure that handles NAKs.

where y_j is the number of times j NAKs were received in an observing window in the past $(\sum_{k=0}^{N_{max}} y_k)$ time windows, and the parameters α_i, $0 \leq \alpha_i \leq N_{max}$, can be assigned a value of 1 if the non-information distribution is used initially (i.e., the number of NAKs received in an observing window is equally likely to be in the range of $[0, N_{max}]$) or assigned accordingly if some prior distribution can be obtained. The expected number of NAKs $E(N_r)$ can then be expressed as

$$E(N_r) = E(\sum_{j=0}^{N_{max}} jp(j)) = \sum_{j=0}^{N_{max}} \frac{j(\alpha_j + y_j)}{\sum_{k=0}^{N_{max}} (\alpha_k + y_k)}. \quad (3.2)$$

Interested readers are referred to [6] for a summary of the theoretical base for Eq. (3.1). Now in the case of **(C3)**, each router designates, for each of its interfaces ℓ, a replier link $\ell_r \in L \setminus \{\ell\}$ that has the smallest value of $E(\lambda)$ or $E(N_r)$.

3.4 Handling of NAKs

The procedure that handles NAKs is outlined in Fig. 5. When a receiver perceives a gap in the sequence number(s) of data packet(s) received, it prepares a NAK that contains (i) the sequence number(s) of lost data packet(s), (ii) *path_info* corresponding to the (source, multicast group) pair with lost packet(s), and (iii) a new *TP_info* IP option. There are five subfields in the *TP_info* option: a one-byte field that identifies the option (*TP_info*), a one-byte *option length* field that gives the size (in bytes) of the *TP_info* option, a one-byte field the least significant bit of which is designated as the *pass_turning_point* bit, a one-byte *interface_info* field that identifies the interface to be recovered, and a 4-byte *turning_point* field. Use of these fields will be explained below.

When a host initially joins a multicast group, it does not have the *path_info*. The new member may either delay sending of any NAK until it receives a data packet with the *path_info* option. Or, it may unicast a *path_info_request* message to the source.

After the receiver composes a NAK, it sends the NAK to the source with the *Router Alert* option. The receiver also sets up a *retransmission_request* timer. Upon timer expiration, if the reply has not returned, the receiver assumes that the NAK is incorrectly forwarded (perhaps due to route/topology change), and will send a *path_info_update* message to the source asking it to include a *path_info* option in the next data packet sent.

When an on-tree router R receives a NAK on interface ℓ_{req}, it inspects whether or not the *pass_turning_point* bit in the NAK has been set. If not, the NAK is still being forwarded upstream with respect to the source of the lost data packet(s). Under this case, router R first updates the statistics to be used to estimate $E(N_r)$ for interface ℓ_{req}. It then retrieves (and deletes) from the list of incoming interfaces in the NAK the identifier of the interface, ℓ_i, on which the source can be reached. If router R has a replier link, ℓ_r, corresponding to ℓ_i, it inserts (i) its IP address into the *turning_point* field request and (ii) the identifier of the interface, ℓ_{req}, into the *interface_info* field, sets the *pass_turning_point* bit, and forwards the request to the replier link ℓ_r. Otherwise, router R forwards the NAK on ℓ_i to its parent router (with respect to the source of the lost packet). On the other hand, if the *pass_turning_point* bit has been set, router R forwards the NAK unchanged to its replier link corresponding to interface ℓ_{req}.

In the case of route or topology changes, a NAK with an outdated *path_info* may be incorrectly forwarded to a router. The consequence is either the router may not have interface with identifier ℓ_i or the router forwards the NAK on an incorrect interface. In the former case the router simply discards the NAK, while in the latter case the NAK will be eventually discarded by some other routers or repliers.

When a group member receives a NAK, if it has the requested data, it unicasts to the turning point the data, along with the identifier of the link to be recovered. (The reply message uses the new protocol number assigned to RMCM.) Both the IP address of the turning point and the identifier of the link to be recovered can be retrieved from the NAK. On the other hand, if the group member does not have the requested data (in which case it is likely that itself has also sent a retransmission request), it simply discards the request.

3.5 ACK Aggregation

A pure NAK-based scheme requires an infinite buffer at the source and/or repliers to cache data packets. This is essentially because the source/repliers cannot purge data packets un-

less they ensure that data packets have been received by all the group members. To deal with this problem, we propose that receivers send *delayed ACKs*. An ACK contains the sequence number, k, that acknowledges the receipt of data packets with sequence number smaller than k and a *path_info* option that carries the path information. The *path_info* option contained in ACKs serves the same purpose as it does in NAKs (Section 3.4).

Delayed ACKs are sent periodically (on a time scale much larger than that usually used in TCP delayed ACKs). Upon intercepting ACKs over all the downstream interfaces (with respect to a source), an on-tree router uses the minimum among all sequence numbers received as the sequence number for the subtree rooted at that router (which we term as *subtree sequence number*, SSN). In case that an on-tree router does not receive an ACK on an interface for a time period that is a multiple, m, of the delayed ACK period, it may exclude that interface from the SSN update process. This allows loss of up to m ACKs on an interface before it is excluded. When the router receives an ACK on an previously-excluded interface, it includes the interface again in the update process if the sequence number carried in the ACK is greater than or equal to the current SSN.

When the source receives from each of its downstream interfaces an ACK that acknowledges k, it purges data packets with sequence numbers smaller than k. Moreover, it includes this minimum sequence number in the *minimum sequence number* field of a data packet's *path_info* IP option to notify all repliers of the fact that data packets with sequence numbers less than k have been received by all the receivers. In the rare case that an interface ℓ is excluded from, and later included in, the SSN update process, some data packets may not be recovered for receivers reached on that interface, if the sequence number, k, carried in the ACK received on ℓ is less than SSN.

One drawback of the proposed ACK aggregation approach is that an on-tree router has to keep for each source the largest sequence number received on each of the downstream interfaces. We are currently investigating how to resolve this issue.

3.6 Handling of Reliable Multicast Incapable Clouds

To make RMCM operate correctly even in the existence of a cloud of incapable routers, we have to consider the following three problems: (i) how a capable router detects whether or not a packet has traversed a cloud; (ii) how recovery from data loss is achieved when clouds of incapable routers exist on the path; and (iii) how ACKs are aggregated in the existence of clouds. The first two problems are self-explanatory. The third problem arises because ACKs are not aggregated inside a cloud and are simply forwarded upstream. Under this case, a capable router immediately above a cloud may receive multiple ACKs on an interface. For example, under the error-free case, router R_1 in Fig. 6 should receive ACKs from 3 downstream routers (R_{15}, R_{16} and R_7). However, R_1 does not know (under the IP multicast model) exactly how many downstream routers it has in the existence of clouds and hence does not know when it can forward an ACK upstream that acknowledges k.

How a capable router detects whether or not a packet has traversed a cloud: A reliable multicast capable router inserts into the *path_info* option the identifier of the incoming interface (on which the data packet comes) *and* the IP address of the outgoing interface. When a capable router receives a data packet with the *path_info* option, it first retrieves (and deletes) from the option the IP address of the outgoing interface of the previous capable router, and checks whether or not it is on the same subnet of the previous capable router [5]. If not, it implies that the data packet has traversed a cloud. The capable router then (i) inserts the true identifier of the incoming interface followed by -1 (which is never used as an interface identifier) in the list of incoming interfaces in the *path_info* option and (ii) inserts the retrieved IP address of the outgoing interface of the upstream router into a new IP option, *upstream_router_info*.[6] Otherwise, the data packet has not traversed a cloud and the capable router proceeds as usual. When a group member receives data packets with the *path_info* option, it stores both the list of incoming interfaces (in the *path_info* option) and the list of IP addresses of upstream capable routers (in the *upstream_router_info* option) if any.

How data recovery is achieved in the existence of clouds: When a group member detects a data loss and sends a NAK, it attaches both options in the IP header of its NAK. When a capable router receives a NAK, it handles it as usual except when the NAK is still being forwarded upward (*Req.pass_turning_point* = 0). Under this case, the router has to distinguish whether or not (i) the immediate upstream router from which lost data packet(s) were sent is incapable; and (ii) the NAK has traversed a cloud at the immediate downstream. Item (i) is identified if the router retrieves (and deletes) a "-1" from the list of incoming interfaces in *path_info*. The true interface on which the source can be reached and the address of the immediate upstream capable router can then be retrieved (and deleted) from *path_info* and *upstream_router_info*, respectively.

To identify item (ii), we define a capable router immediately downstream of a cloud as a *pseudo turning point* and introduce a new *pass_pseudo_turning_point* bit in (the third one-byte field of) the *TP_info* option. When a capable router forwards a NAK upstream (with respect to the source) through a cloud, it fills in the fields of the *TP_info* option and sets the *pass_pseudo_responsible_router* bit in the *TP_info* option. Now we consider four cases when a capable router, R, receives a NAK from downstream (with respect to the source):

(P1) The NAK has traversed an immediate downstream cloud and the immediate upstream router of router R is also incapable: if router R determines to forward the NAK downward toward a replier (i.e., router R becomes the turning point), it sets the *pass_turning_point* bit, but <u>does not</u> fill in the *TP_info* option. This is because if router R did fill in the *TP_info* option, and later when a reply were multicast on the subtree rooted at router R, routers inside the downstream cloud and their children that do not suffer from data loss would have received the reply, leading to loss of recovery isolation. This scenario is depicted in the following example. On the other hand, if router R determines to forward the NAK upward, it forwards the NAK to the immediate upstream capable router.

Example 4: Consider the multicast tree in Fig. 6. Suppose data loss occurs on the link between R_3 and R_7. The NAK from the affected subtree (i.e., the tree rooted at R_7) is forwarded upstream (across the cloud) until it reaches R_1 which in turn forwards the NAK downstream to the replier on the subnet of R_{11}. If R_1 were recorded in the *TP_info* option as the turning point

[5] by comparing the IP address on its own incoming interface against the retrieved IP address.

[6] The *upstream_router_info* option is first created if it is not included in the data packet.

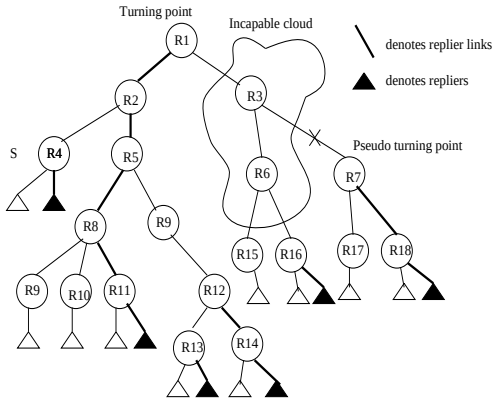


Figure 6: A scenario in which the pseudo turning point, rather than the turning point, should be the router that performs a directed multicast.

and the reply were multicast at the subtree rooted at router R_1, several routers (i.e., R_3, R_6, R_{15}, and R_{16}) that do not suffer from data loss would have received the reply. If router R_7, instead of router R_1, records itself as the (pseudo) turning point, the replier will unicast the reply to R_7 which then performs a directed multicast only on the affected subtree. □

(P2) The NAK has traversed an immediate downstream cloud but the immediate upstream router of router R is capable: if router R determines to forward the NAK downward toward a replier (i.e., router R becomes the turning point), due to the same reason as in **(P1)**, it sets the *pass_turning_point* bit, but does not fill in the *TP_info* option. Otherwise, it resets the *pass_pseudo_turning_point* bit to reflect the fact that the immediate upstream capable router does not have an immediate downstream cloud.

(P3) The NAK is forwarded by a capable router immediately downstream but the immediate upstream router of router R is incapable: if router R determines to forward the NAK downward toward a replier (i.e., router R becomes the turning point), then router R proceeds as usual. Otherwise, before router R forwards the NAK to its upstream capable router, it fills in the *TP_info* option and sets the *pass_pseudo_turning_point* bit. In particular, router R fills in the incoming interface, ℓ_i, on which the source can be reached as the interface to be recovered.

(P4) The NAK is forwarded by a capable router immediately downstream and the immediate upstream router of router R is also capable: this is the "normal" case and router R proceeds as usual.

How ACKs are aggregated in the existence of clouds: After a capable router R_u forwards the first data packet with the *path_info* IP option, it sets up a timer. If a downstream capable router R_d detects a cloud immediately upstream between itself and R_u, it unicasts an *cloud_detection* message to R_u. If R_u does not receive any such unicast upon timer expiration, it assumes that there is no downstream cloud and it will receive one aggregate ACK per interface. Otherwise, it acknowledges to R_d receipt of such an *cloud_detection* message, and will henceforth forward a new ACK that acknowledges k upstream only when it receives from each of the downstream capable routers such an ACK.

To make the scheme robust to *cloud_detection* message losses, R_d initiates a three-way handshaking operation when it identifies itself to R_u: R_d unicasts an *cloud_detection* message to R_u and sets up a timer. R_u acknowledges receipt of such a message. If R_d does not receive any acknowledgment upon its timer expiration, it unicasts again the *cloud_detection* message to R_u. The value of the timer at R_u should be set one or two orders of magnitude larger than that at R_d to accommodate transient network loads and occasional unicast packet loss. Note that the cloud detection operation needs to be performed only once when the first data packet with the *path_info* option traverses.

4 Related Work

As mentioned in Section 1, existing reliable multicast schemes can be roughly classified into two categories: (i) reactive repair-based approaches and (ii) proactive FEC-based approaches. FEC-based approaches combine FEC with ARQ to send repair packets proactively before they are required. In the category of repair-based approaches, several solution approaches have been proposed, among which SRM [5], LBRRM [8], TMTP [22], RMTP [12], PGM [18], TP [14], Search Party [4], ARM [10], IRMA [9], and MTCP [16] have received the most attention.

The pioneer work SRM [5] performs well in suppressing NAKs, but suffers from higher recovery latency (because of its timer-based request/reply suppression mechanism). To effectively suppress duplicate requests/replies, SRM requires that each host accurately estimates the round-trip delay between two hosts and fine tunes its timers accordingly. This may not be easily achievable when group membership or topology changes frequently. Its mechanism for controlling the recovery scope (i.e., by use of the TTL field in the IP header) may still allow replies to reach areas that do not incur data loss.

Pretty Good Multicast (PGM) [18] is a NAK-based protocol (with NAK confirmation (NCF) to ensure reliable transmission of NAK). Packet retransmission is accomplished by the source, receivers, or designated local retransmitter (DLR). It uses Source Path Messages (SPMs) to keep (as soft state) the per-source path information in routers and receivers so that a NAK may be forwarded along the reverse path of the distribution tree to a source. It also specifies procedures that permit DLRs to redirect NAKs to themselves rather than to the source. Random back-off is used to suppress both NAKs and retransmissions. The major drawbacks of PGM are the need to keep per-source information in routers and the potentially long recovery latency due to the random back-off mechanism.

Several hierarchical schemes, e.g. LBRRM [8], TMTP [22] and RMTP [12], have been proposed to control NAK implosion and duplicate replies by organizing, either statically or dynamically, receivers into a recovery tree. In particular, request implosion is controlled by allowing requests from children to their parents only, and duplicate repliers are reduced by unicast (or multicast after certain threshold of requests is exceeded) from parents to children. While simple, the static hierarchical schemes (e.g., LBRRM and TMTP) do not adapt to membership or topology changes well. The dynamic hierarchical scheme (RTMP), on the other hand, relies on an expanding ring search method (with the use of the TTL field) to locate parent routers in the tree hierarchy and may incur increased recovery latency and certain degree of loss of recovery isolation if the parent chosen by a receiver is located downstream with respect to a sender.

Schemes that require modifications to IP routers have been

recently proposed: TP [14] and its variation *Search Party* [4] require (i) moderate modification to IGMP and the membership refresh mechanism of the multicast routing protocol (to facilitate collection of replier information) and (ii) addition of several soft states (which are maintained at each router to keep track of upstream links, downstream links, and replier links). *Search Party* [4] differs from the TP approach simply in that a replier link is randomly chosen to eliminate the need to keep soft states. ARM [10], on the other hand, employs active routers that can perform data caching and customized computation based on multicast data packet types. Each router caches replies and data packets, so that when it receives requests it can act as a replier and responds immediately. Each router also keeps soft states that contain information to suppress NAKs and to prepare for scoped multicast (i.e., multicast of replies only on interfaces on which requests were received). Although the above three schemes are effective in suppressing NAK implosion and duplicate replies, achieving recovery isolation, and reducing recovery latency, they are mainly targeted for source-based trees. They either cannot be directly applied to core-based trees (e.g., ARM) or when directly applied, give sub-optimal performance (e.g., TP and *Search Party*) due to lack of the per-source information.

With a different design objective, both IRMA [9] and MTCP [16] have been proposed to support TCP as the reliable multicast transport protocol. IRMA modifies the functionality of multicast routers that are close to end hosts[7] so as to trick end hosts with the illusion that the communication is unicast TCP, and hence the TCP congestion control and flow control mechanism can be readily reused in IRMA. To achieve this, an IRMA-multicast router has to intercept control (SYN, SYN+ACK, ACK) and data packets incurred in a TCP connection and multicast them to the entire group. In addition, an IRMA-aware multicast router (1) caches the SYN packet in order to support dynamic joins and leaves; and (2) keeps track of numerous states (e.g., for each downstream router, the sequence number, the advertised window of the last ACK packet received, the number of duplicate ACK with the last sequence number) in order to perform ACK aggregation and local repair. As local recovery is used as an enhancement, rather than a replacement, of the end-to-end TCP retransmission mechanism, it is possible that both the sender and one of the repair servers downstream retransmit repair packets in response to packet loss, leading to duplicate replies. Moreover, the repair packet from the source may be multicast to the entire group, leading to loss of recovery isolation. In MTCP [16], all receivers are organized into a multicast tree with some receivers acting as relay nodes. As a result, only code change at end hosts (but not routers) are needed to realize MTCP, and data are reliably transmitted among receivers by taking advantage of the TCP end-to-end retransmission mechanism. The drawback of MTCP is that in order to use receivers as relay nodes, the resulting tree may not be optimal.

5 Performance Evaluation

5.1 Simulation Results

We have implemented RMCM, along with the TP scheme and the ARM scheme, in a Java-integrated customized network simulation tool, *NetSim*^{Q} [19], and conducted a simulation study to validate the proposed design and compare the performance

[7] Although the authors of [9] claimed that only routers close to end hosts need to be IRMA-aware, we believe every multicast router has to be IRMA-aware in order for the scheme to perform ACK aggregation correctly.

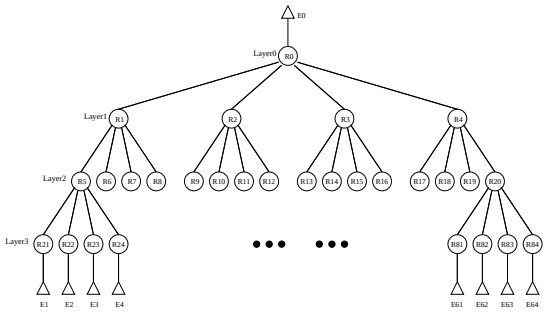


Figure 7: A complete tree topology of depth 3. The source is E_1, while receivers are on the directly attached subnets of routers R_{21} through R_{85}.

of the three schemes. The reasons for selecting TP and ARM for comparison are four-fold: (1) TP has been reported by Papadopoulos *et al.* in [14] to outperform SRM, TMTP, RMTP, and LBRRM in terms of recovery isolation, recovery latency, NAK/reply suppression; (2) ARM has been reported in [10] to give the best performance in terms of recovery latency, due to the fact that every active router in ARM caches data packets and replies (hence, we will use ARM as a baseline scheme for comparison with respect to recovery latency); (3) IRMA and MTCP were designed with a (different) objective of supporting TCP as a multicast transport protocol and hence they fulfilled only a partial set of the objectives outlined in Section 1; and (4) FEC-based approaches are fundamentally different from repair-based approaches, and hence a comparison between them should be made at the abstract model level, but not restricted to specific approaches.

The performance metrics used are:

Sender load: is defined as the ratio of the number of NAKs received by a source to the total number of NAKs generated in a simulation run.

Maximum replier load: The replier load is defined as the ratio of the number of NAKs received by a replier to the total number of NAKs generated in a simulation run. The maximum of the replier loads among all the repliers is taken as the maximum replier load.

Recovery latency: is defined as the time interval between the time when a receiver detects a packet loss and sends a NAK to the time when it receives the corresponding repair packet.

Nuisance: is defined as the ratio of the total number of *unsolicited* replies received by a receiver to the total number of replies received by the same receiver.

Both the sender load and the maximum replier load serve as indices of how well NAKs are distributed among group members. Nuisance is an index of how well data recovery is isolated only to the area with data loss.

Two types of network topology are used: the complete tree topology in which group members are attached to leaf routers and the root router (one example is depicted in Fig. 7) and the network topology randomly generated using NetSim^Q (one example is depicted in Fig. 8). We assume in both types of topology that (i) packet loss is equally likely to occur on any link, and the delay on each link is normally distributed with mean equal to 1 unit of time (i.e., the time unit is normalized to the mean link delay); (ii) the probability of packet loss on directly attached subnets is zero, and the delay between a local router

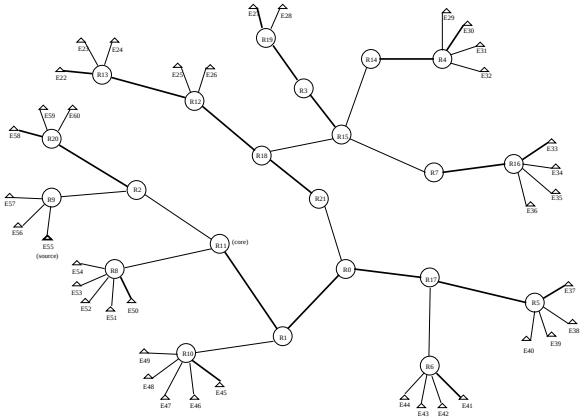


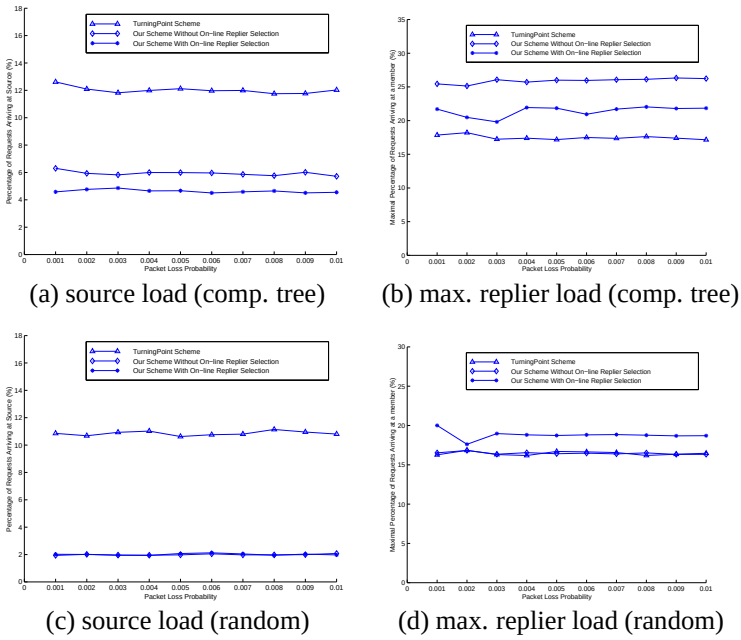
Figure 8: The random topology used in the simulation (with replier link boldfaced).

and the sender/receiver on its directly attached subnet is negligible. Both NAKs and unicast messages from repliers to turning points are assumed to be reliably delivered, but both data packets and repair packets (in directed multicast) may be lost. In spite of numerous system parameters involved, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a representative set of parameter values (reported below) is valid over a wide range of parameter values.

Comparison with the TP Approach: To evaluate the capability of controlling NAK implosion at the source, we compare both schemes in terms of the sender load and the maximum replier load in a complete tree topology of depth 3 and degree 3 (in which the leftmost router is the source, the leaf routers are the receivers, the root is the core, and each non-leaf router has 3 children routers) and a randomly generated topology (Fig. 8, in which router E_{29} is the source). In the randomly generated topology, router R_0, R_{21}, R_{18}, R_{15}, and R_{14} are selected as the core, respectively, in the various simulation runs. The probability of packet loss is 0.005 over all the links.

As shown in Fig. 9, the sender load incurred in RMCM is much smaller than that in TP. The performance improvement is even more pronounced when the replier links are on-line selected via Bayesian analysis. On the other hand, RMCM has a higher maximum replier load than TP. This is due to the fact that some of the requests that are directed to the source in TP are now distributed among repliers in RMCM. Note that the increase in the maximum replier load is smaller than the decrease in the sender load in RMCM. This implies that the sender load is distributed among more than one replier. This demonstrates the capability of NAK implosion control in RMCM.

Fig. 10 gives the performance comparison in terms of recovery latency between TP and RMCM. RMCM with the use of on-line replier selection mechanism achieves better performance than TP in the case of complete tree topology (Fig. 10 (a)). This is because all intermediate routers select (under the assumption of equal link error probabilities) the shortest downstream path as the replier path, thus leading to better performance. In the case of random topology, the performance depend on the relative positions of the source, the core, the receiver that suffers from data loss, and the replier. For example, given the random topology in Fig. 8, host E_{50} experiences a larger recovery latency in RMCM than in TP (Fig. 10 (b)), while host E_{58} experiences the reversed situation (Fig. 10 (c)). This is because a NAK issued by E_{50}

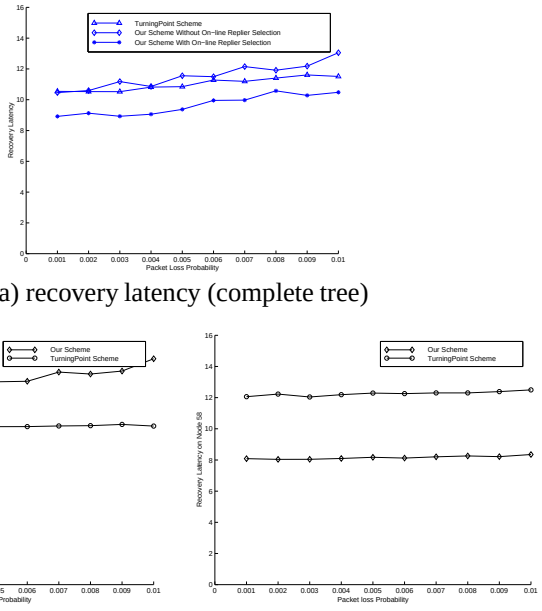


(a) source load (comp. tree) (b) max. replier load (comp. tree)

(c) source load (random) (d) max. replier load (random)

Figure 9: Performance comparison (in terms of source load and maximum replier load) between TP and RMCM. R_0 is selected as the core. In RMCM, if the on-line replier selection mechanism is not used, each router randomly picks up a link that leads to repliers.

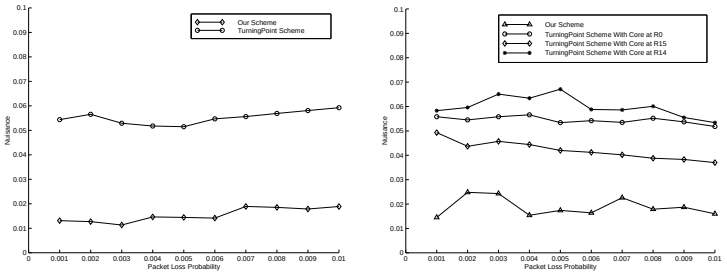
travels 7 IP hops ($E_{50} \rightarrow R_8 \rightarrow R_{11} \rightarrow R_1 \rightarrow R_0 \rightarrow R_{17} \rightarrow R_5 \rightarrow E_{37}$) before it reaches the replier in RMCM, while in TP after the NAK arrives at the core (R_{11}) it is unicast to the source E_{55}. On the other hand, a NAK issued by E_{58} travels directly to the source in RMCM, while it is sent all the way to the core before being unicast to the source in TP.



(a) recovery latency (complete tree)

(b) recovery latency for E_{50} (c) recovery latency for E_{58}

Figure 10: Performance comparison (in terms of recovery latency) between TP and RMCM.



(a) complete tree topology (b) random topology

Figure 11: Performance comparison (in terms of nuisance) between TP and RMCM.

Fig. 11 gives the simulation results in terms of nuisance between the two schemes. The performance of TP is sensitive to the placement of the core and the relative position of the source, the receiver that suffers from data loss, and the replier, and is always inferior to that of RMCM. This verifies our assertion that lack of the per-source information at each router makes the TP scheme lose certain degree of recovery isolation in a core-based tree, while RMCM does not suffer from this drawback.

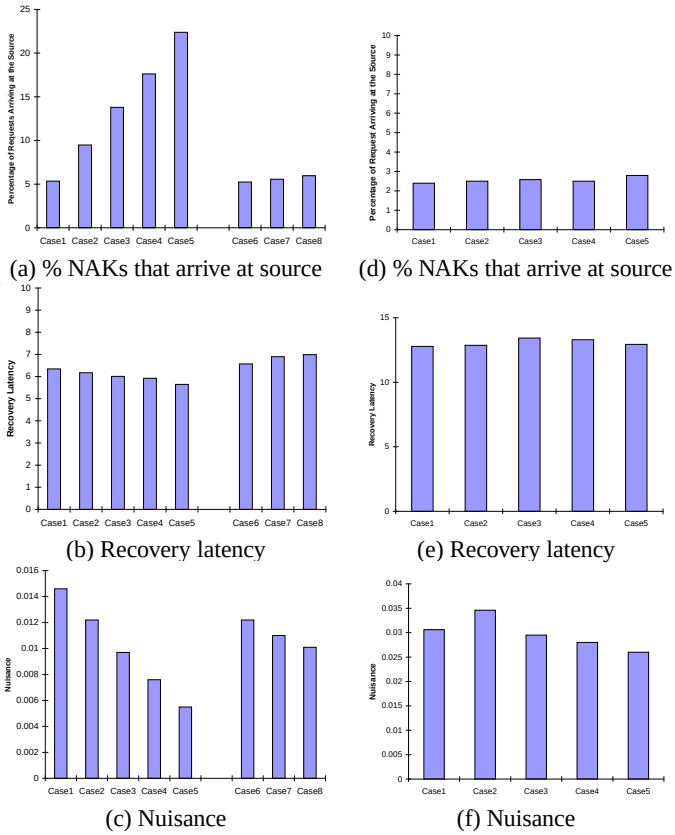
Handling of Clouds of Incapable Routers: We now demonstrate the effectiveness of RMCM in handling clouds of incapable routers. We show that not only clouds are detected, NAKs and replies are properly directed, but also recovery isolation is retained to the maximal possible extent. Note that the cloud handling mechanism can be readily applied to the TP scheme.

We consider a complete tree topology (Fig. 7) of depth 3 and degree 4 and a randomly generated topology (Fig. 8) in which E_{29} is the source and R_{21} is the core. The probability of packet loss is set to 0.005. In the complete tree topology, we consider the following eight cases. Note that in Cases 1–5 the number of incapable routers at tree level 1 increases, while in Cases 6–8 the number of incapable routers at level 2 increases.

Case	Incapable Routers
1	None
2	R1
3	R1, R2
4	R1, R2, R3
5	R1, R2, R3, R4
6	R8, R12, R16, R20
7	R7, R8, R11, R12, R15, R16, R19, R20
8	R6, R7, R8, R10, R11, R12, R14, R15, R16, R18, R19, R20

Fig. 12 (a)–(c) give the performance (in terms of sender load, recovery latency, and nuisance) of RMCM in the complete tree topology. Nuisance decreases as the number of incapable routers in layer 1 increases (Fig. 12, Cases 1–5). This is because with the use of the pseudo turning point (which in this case is a layer 2 router), directed multicast is only performed at the area with data loss (i.e., the subtree rooted at the pseudo turning point). The sender load, on the other hand, increases as the number of incapable routers in layer 1 increases (Fig. 12 (a), Cases 1-5). This is because in order to bypass the clouds more requests are sent by layer 2 routers to the source. As the number of incapable routers in layer 2 increases (Fig. 12, Cases 6–8), nuisance is still quite small (i.e., recovery is still isolated to the area that suffers from data loss). Moreover, the sender load does not increase with the number of incapable routers in layer 2 (Fig. 12, Cases 6-8). This is because layer 1 routers properly direct retransmission requests that traverse a layer-2 cloud to replier links.

In the random topology, we consider the following five cases.



(a) % NAKs that arrive at source (d) % NAKs that arrive at source

(b) Recovery latency (e) Recovery latency

(c) Nuisance (f) Nuisance

Figure 12: Impact of incapable clouds on the performance of RMCM. (a)-(c) are results in the complete tree topology and (d)-(f) are results in the random topology.

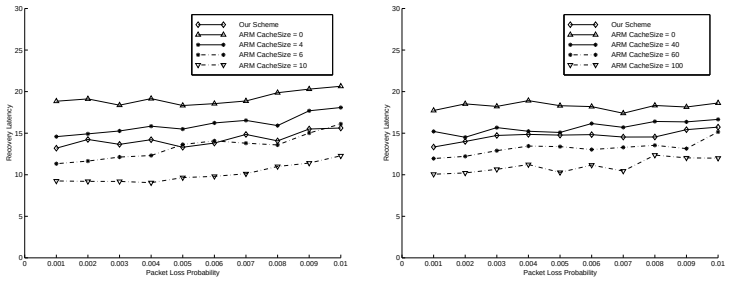
Case	Non-active Nodes
1	None
2	R2
3	R1, R2
4	R1, R2, R7
5	R1, R2, R7, R18

Again as shown in Fig. 12 (d)–(f), nuisance remains small (recovery isolation is achieved) and the sender load remains low (NAKs are properly re-directed to repliers).

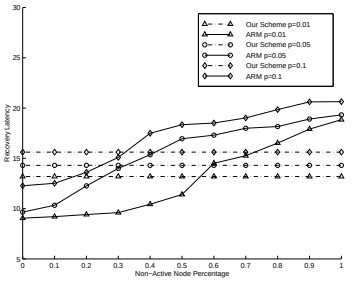
Comparison with ARM: As reported in [10], the performance of ARM with respect o recovery latency is prominent. This results from the fact that every active router caches repair packets and data packets so that NAKs can be quickly handled by an intermediate router with the requested data. The price which ARM has to pay is, however, the increase in network caches, the time overhead in caching repair/data packets, and the complexity of router design.

As ARM cannot be directly applied to core-based trees, we use a source-based tree in the simulation runs. We conduct two sets of simulation runs. In the first set of simulation runs we vary the cache size under the condition that all the routers are active. In the second set of simulation runs we vary the percentage of active/capable routers, while keeping the cache size fixed at certain value. In each set of simulation runs, we generate multicasts that send data packets at the rate of 1 and 10 packets per unit time, respectively. The other parameters relevant to the ARM performance are fine tuned so as to achieve the best performance of ARM.

Fig. 13 compares RMCM against ARM with respect to recovery latency. Several observations are in order. First, as



(a) packet generation rate = 1 (b) packet generation rate = 10



(c) packet generation rate = 1, cache size = 10

Figure 13: Performance comparison (in terms of recovery latency) between ARM and RMCM. (a) and (b) show the impact of cache sizes on the performance while (c) shows that of the percentage of active/capable routers on the performance.

shown in Fig. 13 (a)–(b), in the case that all routers are active, the performance of RMCM is comparable to that of ARM with cache size 6 packets (50 packets) when the packet generation rate is 1 per unit time (10 per unit time). As the percentage of non-active/incapable routers increases, the performance of ARM degrades (Fig. 13 (c)), while that of RMCM remains approximately the same. When the probability of packet loss is 0.01, RMCM outperforms ARM when there are more than 60% non-active/incapable routers. When the probability of packet loss falls between 0.05 and 0.1, RMCM outperforms ARM when there are more than 30% non-active/incapable routers. Second, the performance of ARM is very sensitive to (i) the percentage of active routers in the network and (ii) the cache size at each router. For a given percentage of active routers and cache size, it is then greatly affected by the other parameters such as the data packet generation rate and the life time during which a packet is cached. In contrast, RMCM (and hence TP) are rather insensitive to the above parameters.

5.2 Implementation Note

We have also implemented RMCM on *FreeBSD 2.2.8* and evaluated its add-on overheads on routers. The implementation effort involves two parts: (a) router modification and (b) a user-level RMCM daemon at end hosts. Although the current implementation is on the *FreeBSD 2.2.8* kernel, the software can be easily ported to other Unix-based platforms, as the modification is generic to most Unix-based platforms.

On the end host, the user-level daemon sends and receives data/repair packets, *ACK*s/*NAK*s, and *path_info*, all through *raw sockets*. The reason for using raw sockets is because they allow the applications to easily generate *IP* packets with new IP options. On the other hand, to minimize the context switch overhead, all the functions to be supported at a router are implemented inside the kernel. The structure of a *RMCM* router is given in Fig. 14. *RMCM* consists of three modules, namely the *option handling* module, the *state management* module and the *link selection* module. In the Internet Layer, when *ip_input* gets packets from the network interface, it filters out packets with those newly-defined options and sends them to the option handling module in *RMCM* (through pointers to avoid data copy). The *option handling* module will process these packets according to the rules described in Section 3. After that, packets with *path_info* will be forwarded to *ip_forward* where they will be multicasted. Repair packets generated by RMCM will be passed to the *link selection* module and will be forwarded to the specific interface through the Link Layer. Other types of packets will be further passed to the *state management* module.

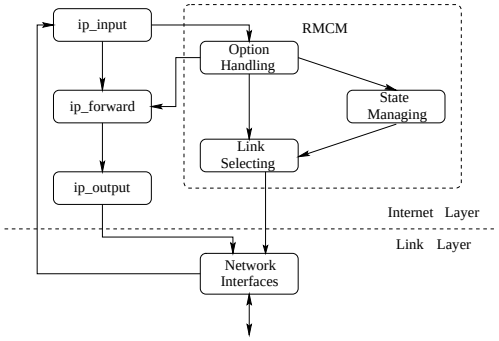


Figure 14: The structure of a *RMCM* Router (*ip_input*, *ip_forward* and *ip_output* are three main kernel functions in IP layer of *FreeBSD*.)

The *state management* module maintains the states on each interface and performs ACK aggregation. Its main functions include: (1) update of the NAK arrival rate (2) ACK aggregation and (3) selection of a replier link for each of its interfaces. To accomplish these functions, the *state management* maintains the following information for each interface:

1. *Interface Index*: an integer that uniquely identifies the interface within the kernel.
2. *Replier Flag*: a flag that specifies if there is a volunteering downstream replier reachable on this interface.
3. *Replier Link Index*: an interface index of the replier link for this interface.
4. *Last NAK Arrival Time*: the time instant at which the last NAK arrived.
5. *NAK Number*: The total number of NAKs received on this interface so far.
6. *NAK Arrival Rate*: The estimated average NAK arrival rate.
7. *ACK List*: The linked list for ACKs received on this interface. Each item in the list keeps the ACK information for each source, and includes: (i) *Source ID*: the identification of the source which consists of an IP address and a port number; and (ii) *ACK Sequence Number*: the received ACK sequence number for this source.

The information for all the interfaces is kept in a linked list and will be initialized when the IP protocol is initialized. The contents in this list will be updated when the *State Management* module receives a packet. Specifically, when a NAK is received, the NAK number will be increased by one, the estimated average NAK arrival rate will be recalculated, and the replier link for each interface will be reselected. When an ACK is received, the corresponding item in the ACK list will be updated and ACK

aggregation is performed.

The *link selection* module is responsible for determining on which interface the ACKs, NAKs or repair packets should be forwarded and then forwarding them on those interfaces. ACKs will be forwarded upstream through the Link Layer. NAKs will be either forwarded on the replier link or toward source (both through the Link Layer), based on the information *State Management* module maintains.

Overhead evaluation: The overheads introduced by *RMCM* at a router include option update/processing, IP checksum recalculation, NAK arrival rate estimation, selection of replier links, and ACK aggregation. We have measured these overheads in terms of CPU time, using a PC with Pentium II-400 and 256M RAM to emulate a router. The time resolution of our experiments is microsecond, because of the time granularity provided by FreeBSD. The result of our measurement is shown in Table. 5.2. Most operations cost less than 1 microsecond, and the dominating overhead is checksum recalculation. In particular, the overhead incurred in checksum recalculation depends on the IP header size. In our experiment, the packet size changes from 20 bytes (header only) to 60 bytes (header and option). From the experiment result, we conclude that the overhead introduced by *RMCM* is reasonably small.

Operation	Option Proc.	Checksum Cal.	NAK Rate Est
Time(μSec)	<1	1-3	<1
Operation	Replier Sel	Ack Aggregation	
Time(μSec)	<1	<1	

Table 1: Overhead Test Result

6 Conclusion

In this paper, we design, implement, and evaluate a repair-based reliable multicast scheme for core-based multicast trees that closely emulates the optimal recovery scenario achieved in the turning point scheme [14]. We address the issues of (i) identifying upstream/downstream interfaces (with respect to a source) in core-based multicast trees, (ii) selecting repliers with the use of Bayesian analysis and simple parameter estimation, (iii) employing a delayed ACK approach to notify senders/repliers of when data packets can be purged, and (iv) handling clouds of reliable-multicast-incapable routers.

Through event-driven simulation, we show that RMCM achieves better performance than TP in terms of NAK implosion control and recovery isolation under regular complete tree topologies and random topologies. RMCM also achieves better performance in terms of recovery latency under complete tree topologies. As compared to ARM (the known scheme that gives the best performance in terms of recovery latency), the performance of RMCM is comparable to that of ARM with cache of 50 packets when the source sends data packets at a rate of 10 packets per unit time (the time is normalized to the mean link delay of transporting a data packet). Moreover, RMCM is rather insensitive to the percentage of capable routers in the network, the cache size at each router, and the data packet generation rate (while the performance of ARM is contingent upon fine tuning of all the above parameters). In our efforts to implement RMCM in FreeBSD, we verify that the processing overhead incurred at a RMCM router is reasonably small.

References

[1] A. Ballardie, "Core based trees (CBT version 3) multicast routing: Protocol specification," *draft-ietf-idmr-cbt-spec-**.txt*, Internet draft, August 1998.

[2] J. Bolot, S. Fosse-Parisis, D. Towsley, "Adaptive FEC-based error control for interactive audio in the Internet," *Proc. IEEE INFOCOM'99*, New York, USA, March 1999.

[3] B. Cain, S. Deering, and A. Thyagarajan, "Internet group management protocol, version 3 (IGMPv3)," *draft-ietf-idmr-igmp-v3-.txt*, Internet draft, 1999.

[4] A. Costello, "Search party: an approach to reliable multicast with local recovery," *Proc. IEEE INFOCOM'99*, New York, USA, March 1999.

[5] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *ACM SIGCOMM'95*, pp.342 - 356, October 1995.

[6] Y. Gao, Y. Ge, and J. C. Hou, "RMCM: reliable multicasts for core-based multicast trees," Tech. report, Department of Electr. Eng., Ohio State Univ., August 2000.

[7] M. Handley, I. Kouvelas, and L. Vicisano, "Bi-directional protocol independent multicast," *draft-ietf-pim-bidir-00.txt*, Internet draft, March 2000.

[8] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," *Proc. ACM SIGCOMM'95*, pp.328 - 341, October 1995.

[9] K.-W. Lee, S. Ha, and V. Bharghavan, "IRMA: a reliable multicast architecture for the Internet," *Proc. INFOCOM'99*, New York, NY, March 1999.

[10] L.-w. Lehman, S. J. Garland, and D. L. Tennenhouse, "Active reliable multicast," *Proc. IEEE INFOCOM'98*, San Francisco, CA, March 1998.

[11] B. N. Levine and J. J. Garcia-Luna-Aceves, "Improving Internet Multicast with Routing Labels," *Proc. IEEE ICNP'97*, Atlanta, Georgia, October 1997.

[12] J. Lin, S. Paul, K. Sabnani, and S. Bhattacharyya, "A reliable multicast transport protocol(RMTP)," in *IEEE Journal on Selected Areas in Communications*, Vol. 15, April 1997.

[13] J. Nonnenmacher, E. Biersack, D. Towsley, "Parity-based loss recovery for reliable multicast transmission," *IEEE Trans. on Networking*, January 1998.

[14] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast applications," *Proc. IEEE INFOCOM'98*, San Francisco, CA, March 1998.

[15] R. Perlman, C.-Y. Lee, A. Ballardie, J. Crowcroft, Z. Wang, and T. Maufer. "Simple multicast: a design for simple, low-overhead multicast," *draft-perlman-simple-multicast-02.txt*, Internet draft, 1999.

[16] I. Rhee, N. Balaguru, and G. N. Rouskas, "MTCP: scalable TCP-like congestion control for reliable multicast," *Proc. IEEE INFOCOM'99*, New York, NY, March 1999.

[17] D. Rubenstein, S. Kasera, D. Towsley, J. Kurose, "Improving reliable multicast using active parity encoding services (APES)," *Proc. IEEE INFOCOM'99*, New York, NY, March 1999.

[18] T. Speakman *et al*, "PGM Reliable Transport Protocol," *draft-ietf-speakman-pgm-spec-04.txt*, Internet draft, April 2000.

[19] H.-Y. Tyan, B. Wang, Y. Ye, L. Su, W. Lin, and C.-J. Hou, "NetSim^Q: a Java-integrated network simulation tool for QoS control in high speed networks," http://eewww.eng.ohio-state.edu/drcl/grants/middleware97/netsimQ.html.

[20] L. Wei, D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, and P. Sharma, "Protocol independent multicast – sparse mode (PIM-SM: protocol specification," *draft-ietf-idmr-pim-v2-sm-01.txt*, Internet draft, November 1999.

[21] D. Wetherall, J. Guttag, and D. L. Tennenhouse, "ANTS: A toolkit for building and dynamically deploying network protocols," *IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[22] R. Yavatkar, J. Griffioen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," *ACM Multimedia*, 1995.