A Scalable Recovery Tree Construction Scheme Considering Spatial Locality of Packet Loss

Jinsuk Baek[1], ***Member*** **and Jehan-François Pâris**[2], ***Non-member***
[1]Department of Computer Science, Winston-Salem State University,
601 M. L. King Jr Dr, Winston-Salem, NC 27110, USA
[e-mail: baekj@wssu,edu]
[2]Department of Computer Science, University of Houston,
4800 Calhoun Rd, Houston, TX 27204, USA
[e-mail: paris@cs.uh.edu]
*Corresponding author: Jinsuk Baek

Received January 15, 2008; revised February 26, 2008; accepted March 1, 2008; published April 25, 2008

Abstract

Packet losses tend to occur during short error bursts separated by long periods of relatively error-free transmission. There is also a significant spatial correlation in loss among the receiver nodes in a multicast session. To recover packet transmission errors at the transport layer, tree-based protocols construct a logical tree for error recovery before data transmission is started. The current tree construction scheme does not scale well because it overloads the sender node. We propose a scalable recovery tree construction scheme considering these properties. Unlike the existing tree construction schemes, our scheme distributes some tasks normally handled by the sender node to specific nodes acting as repair node distributors. It also allows receiver nodes to adaptively re-select their repair node when they experience unacceptable error recovery delay. Simulation results show that our scheme constructs the logical tree with reduced message and time overhead. Our analysis also indicates that it provides fast error recovery, since it can reduce the number of additional retransmissions from its upstream repair nodes or sender node.

Keywords**:** Multicast, reliability, scalability, recovery tree, spatial locality

Digital Object Identifier 10.3837/tiis.2008.02.002

1. Introduction

Many network applications require a sender to distribute the same data to a large number of receivers. Multicast is an efficient way to support these applications. One of the most difficult issues in end-to-end multicasting is that of providing an error-free transmission mechanism. Ensuring reliability requires efficient schemes for retransmission control, flow control, congestion control and so on. This has led to numerous proposals [1][2][3][4][5][6][7][8] aiming at providing scalable reliable schemes.

Among the many protocols [1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16], most recently, overlay reliable multicast protocols [1][2][3], controlling and maintaining an efficient overlay for data transmission, have been proposed. However, there are still many issues to be considered including addressing and advertisement, path quality measurement, tree refinement, and failure recovery. In this paper, we focus on the tree-based protocol that has been known to provide high scalability and reliability. They construct a logical tree at the transport layer. This logical tree comprises three types of nodes: a *sender node*, *repair nodes*, and *receiver nodes*. The sender node is the root of the logical multicast tree. It controls the overall tree construction and is ultimately responsible for resending lost packets within the group. Each repair node acts as a local server for its downstream nodes. It stores in its buffer recently received packets and performs local error recovery for these nodes. Hence, tree-based protocols achieve scalability by distributing the server retransmission workload among the repair nodes.

There are still some open issues in tree-based protocols. One of the most important and difficult of them is how to construct the logical tree in an efficient manner. Logical tree construction schemes can be classified into top-down [12] and bottom-up schemes [13]. Both schemes suffer from their own limitations. In most cases, the top-down scheme guarantees caching of data somewhere in the tree hierarchy, and produces loop-free tree construction with fewer control messages than the bottom-up scheme. At the same time, it does not provide concurrent tree creation and takes a long time to construct a logical tree. Conversely, the bottom-up scheme forms the optimal logical tree more quickly thanks to its parallelism but requires more messages than the top-down scheme does.

One of the authors has recently proposed two efficient hybrid schemes [14] combining the advantages of the top-down and bottom-up approaches. Both schemes construct their logical tree in a semi-concurrent manner while minimizing the number of control messages. Unfortunately, these two schemes also imposed some additional overheads on the sender node.

A common objective of all tree-building schemes is constructing a logical tree that reflects best the physical structure of the network. In fact, this has often been the main criterion for evaluating the efficiency of a particular tree-building scheme [12][13][14]. Unfortunately, the physical tree-like logical trees constructed by previous schemes did not always provide a fast error recovery as they expected because they did not consider that packet transmission errors often are strongly correlated. As a result, these schemes were unable to take into account the temporal and spatial locality of packet losses especially when each receiver node selects its repair node having a shortest Time-to-Live (TTL) distance.

We propose a more efficient tree construction scheme taking advantage of this temporal and spatial locality. In most networks, transmission errors tend to happen in bursts separated by long periods of relatively error-free transmission. Unlike our previous schemes, this new scheme delegates some of the tasks previously handled by the sender node to specific nodes acting as repair node distributors. In addition, it provides a mechanism for relocating repair nodes that cannot fulfill their duties because they fail to receive the packets their receiver nodes did not receive. Our new scheme constructs the logical tree without increasing the number of control messages and tree construction time.

The remainder of this paper is organized as follows. Section 2 reviews existing logical tree construction schemes. In section 3, our new scheme is described in detail. Section 4 contains the simulation results of the proposed scheme. Finally, Section 5 contains directions for future work and our conclusions.

2. Existing Schemes

The common purpose of all tree construction schemes is to construct a tree having the sender node of the multicast session as root, the repair nodes as non-terminal nodes and the receiver nodes as leaves.

A logical tree construction includes several steps: 1) advertising the multicast session, 2) discovering a repair node for each receiver node, and 3) binding each receiver node to its repair node. In the multicast session advertisement phase, all nodes obtain the group address of the multicast, the address of the sender node, and other necessary information for tree construction. This process can be realized by using a mechanism such as a web page announcement [12][13][14]. After that, each receiver node starts to find one or more candidate repair nodes that are available in the session for its error recovery. Finally, each receiver nodes selects and binds to the repair node having a shortest TTL distance among the candidate repair nodes.

This logical tree can be constructed using a bottom-up [13], a top-down [12], or a hybrid scheme [14]. There are two options in deploying the repair nodes in tree-based protocols. First, they can be elected from all the multicast participants. Second, they can be pre-deployed dedicated nodes, which are able to act as representatives for their local groups. In this case, it should be decided whether a node will function as a repair node or not before data transmission is started. We note that the first case introduces additional overhead in electing the repair nodes among all nodes. Also, defining the repair node selection criterion is critical to performance. In this paper, we consider the second case. Even though the repair nodes are pre-determined by the multicast service provider, we assume they are not known in advance to any node in the multicast session. Hence, each node will try to locate a favorable repair node that is an active end-host in its local group. All schemes are based on the following assumptions. These are standard hypotheses made by all tree construction schemes [8][12][13][14].

- There is one sender node *(S)* and it controls overall tree constructions. $|S| = N^S = 1$.
- There are N^{RP} repair nodes and the repair nodes are pre-determined.
- RP is the set of repair nodes with $|RP| = N^{RP}$.
- There are N receiver nodes that want to join a multicast session.
- RC is the set of receiver nodes with $|RC| = N$.
- The repair node i ($RP_i \in RP$, $1 \leq i \leq N^{RP}$) can accept up to N_i receiver nodes as its child nodes.
- The control messages are reliably transmitted.

2.1 Bottom-Up Scheme

In the bottom-up scheme [13], each receiver node actively finds its repair node by multicasting a *QUERY* message. Hence, in some cases, each repair node forms its own group even before becoming attached to the logical tree. As seen in **Fig. 1**, the nodes perform the following sequence of actions:

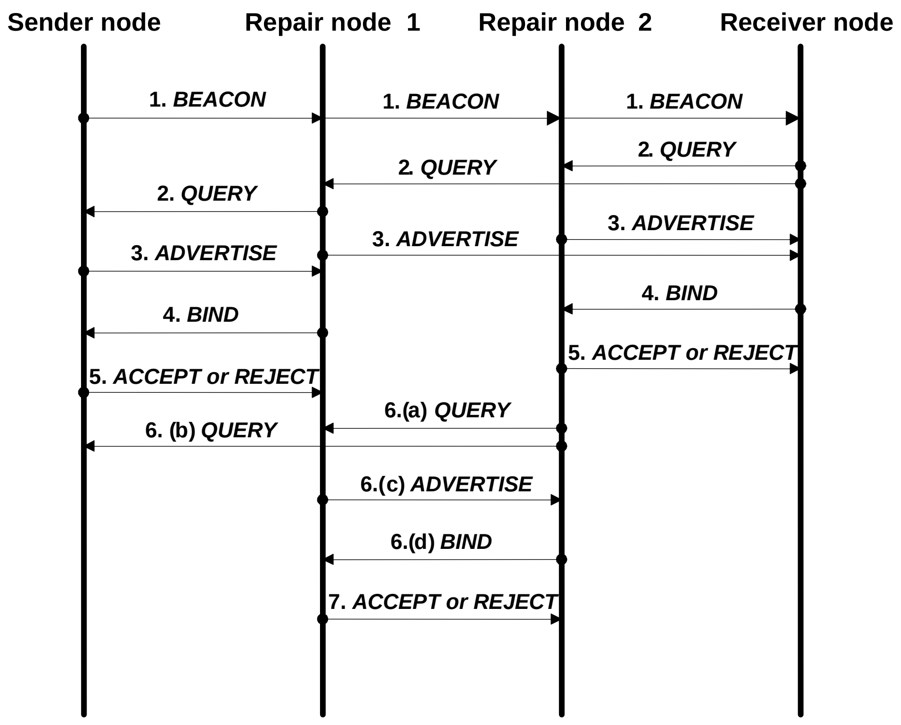


Fig. 1. Bottom-up scheme

1. The sender node first sends a session *BEACON* message to all nodes to indicate the start of a tree construction. This message includes a metric. Based on the metric, each repair node and receiver node can measure the TTL distance from the sender.
2. Each node multicasts a *QUERY* message to the group it wishes to join. The Initial TTL of this query is set to one.
3. One or more repair nodes (possibly sender node for some nodes which are closely located to the sender node) may response to the receiver node with an *ADVERTISE* message. This message contains the specific TTL distance from the repair node to the sender node. The receiver node selects the repair node with the shortest TTL distance from the sender node. In case of two or more repair nodes with the same TTL distance from the sender node, the lowest IP address among them is used as a tie-breaker. The node then selects the best-suited repair node. If there is no response from any repair node, the node goes back to STEP 2 and re-sends a *QUERY* message with an increased TTL value. This step will be repeated with ever increasing TTL values until at least one repair node answers.
4. Upon selecting the best repair node, the receiver node sends a *BIND* message to the repair node.
5. On receiving a *BIND* message, a sender node or repair node will verify the message and check its capacity. If there is an available space for a new member, they will send an *ACCEPT* message. If a *REJECT* message is received, the node goes back to STEP

4. If they fail to receive an *ACCEPT* message after a number of attempts, they try to bind to the next best repair node. If an *ACCEPT* message is received, the node is attached to the tree (Let us call this status ON_TREE).

6. If the repair node is still not attached to the tree (Let us call this status NON_TREE), it also begins the process of finding its parent node.
7. The parent node responds to the repair node with an *ACCEPT* or *REJECT* message.

This bottom-up scheme provides a simple and robust method for tree construction. It also provides concurrent tree construction by allowing each node to actively find its repair node. However, it causes message overhead, as each nodes must generate a lot of *QUERY* messages to find their repair node. We would like to argue that this property can make loop relationship among the repair nodes when two or more repair nodes simultaneously bind together.

2.2 Top-Down Scheme

In the top-down scheme [12], the tree is constructed from the sender node. That is, the sender node starts to construct the tree by accepting some repair nodes and receiver nodes that are located close to it. After becoming a member of some group, each repair node accepts a number of receiver nodes as a member. As seen in **Fig. 2**, the nodes perform the following sequence of actions:

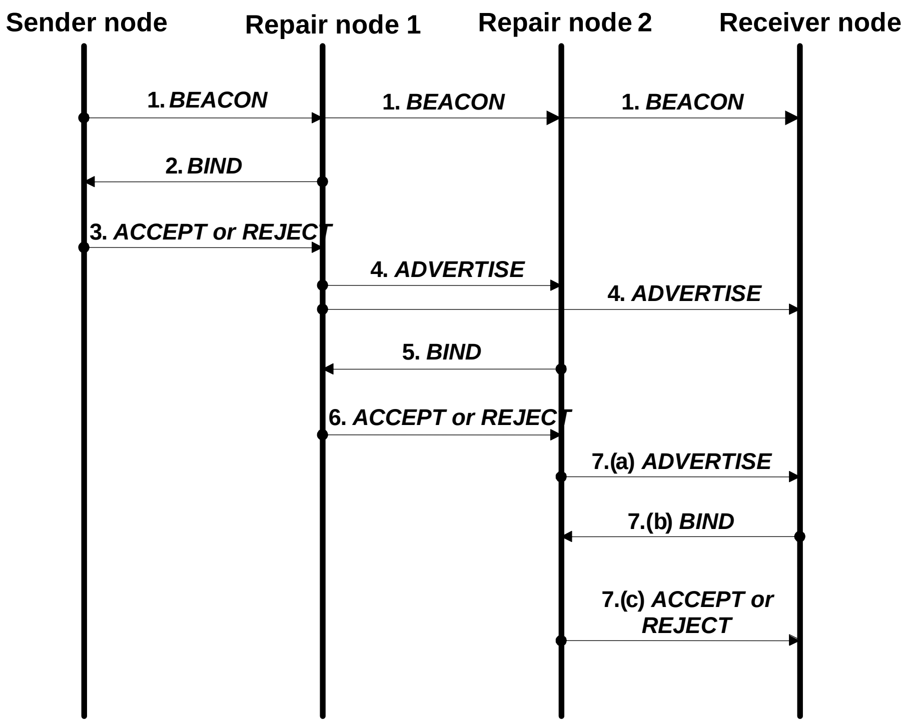


Fig. 2. Top-down scheme

1. The sender node sends a session *BEACON* message to all nodes to indicate tree construction. This message includes a metric. Based on the metric, each repair node and receiver node can measure the TTL distance from the sender.

2. Some neighboring nodes whose TTL distance is less than the threshold send a *BIND* message to the sender node indicating that they wish to join the group.
3. On receiving a *BIND* message, a sender node or repair node will verify the message and check its capacity. If there is an available space for a new member, they will send an *ACCEPT* message. If an *ACCEPT* message is received, the nodes are attached to the tree and said to be ON_TREE. Otherwise, they must go back to STEP 2.
4. ON-TREE repair nodes begin to periodically transmit an *ADVERTISE* message to invite other (NON_TREE) nodes. This message contains the specific TTL distance from the repair node to the sender node.
5. Each NON_TREE node collects information from repair nodes and maintains a list of them based on the *ADVERTISE* messages. Upon selecting the best repair node, the NON_TREE node sends a *BIND* message to the repair node. If it fails to receive an *ACCEPT* message after a number of attempts, it tries the next best repair node.
6. The repair node verifies the message sent from the NON_TREE node and responds with an *ACCEPT* or *REJECT* message after checking its capacity. If an *ACCEPT* message is received, the node is attached to the tree (ON_TREE).
7. Otherwise, it goes back to STEP 5. The newly joined ON_TREE repair nodes now perform their process at STEP 4.

The top-down scheme guarantees that its outcome will be a loop-free logical tree. It also reduces the message overhead as it eliminates the need for multicast *QUERY* message from the receiver node. However, it does not achieve concurrent tree construction, because it does not allow repair nodes to form local group before they are themselves attached to the tree. In addition, it still generates a high number of control messages, because each repair node periodically transmits an *ADVERTISE* message.

2.3 Hybrid Scheme

The hybrid scheme [14] has two major differences from the other two schemes. First, unlike the bottom-up scheme, a receiver node does not multicast a blind *QUERY* message in order to find its repair node. Second, unlike the top-down scheme, a repair node does not periodically send an *ADVERTISE* message to invite receiver nodes. These two features significantly reduce the message overhead. As seen in **Fig. 3**, the hybrid scheme requires all nodes to go though the following sequence of actions:

1. The sender node sends a session *BEACON* message to all nodes to indicate tree construction. This message includes a metric. Based on the metric, each repair node and receiver node can measure the TTL distance from the sender node.
2. When a repair node receives a *BEACON* message, it sends an *InfoRP* message containing the repair node's information, such as IP address, and repair node ID. The sender node collects information of the repair nodes based on the *InfoRP* messages it has received and maintains a list of repair nodes.
3. After receiving the *BEACON* message, each repair node and receiver node measures the TTL distance from the sender node. The repair nodes and receiver node with a favorable TTL distance to the sender node reply with a BIND message to indicate they want to join the tree.
4. The sender node verifies the message and responds with an *ACCEPT* or *REJECT* message based on its capacity. Nodes that receive an *ACCEPT* message attached themselves to the tree (ON_TREE). Otherwise, they go back to STEP 3.

5. Upon receiving the *BEACON* message, the nodes that have not joined the session yet send a *RPQuery* message to the sender node to get some candidate repair nodes.
6. The sender node selects some candidate repair nodes for the given node, based on the *InfoRP* messages it has received from the active repair nodes and responds with a *RPList* message that includes a list of available candidate repair nodes.
7. The nodes will then select the best-suited repair node among the candidates by sending a *QUERY* message to them.
8. The candidate repair nodes will reply with an *ADVERTISE* message containing their TTL values. The receiver node will use these TTL values to select the repair node to which it will bind. If there are two or more repair nodes having the same TTL value, round-trip times will be used as a tie-breaker.
9. The node sends a *BIND* message to the best-suited repair node.
10. The repair node replies to the node with a *ACCEPT* or *REJECT* message after checking its capacity. These procedures are repeated until all the nodes are assimilated into the logical tree.

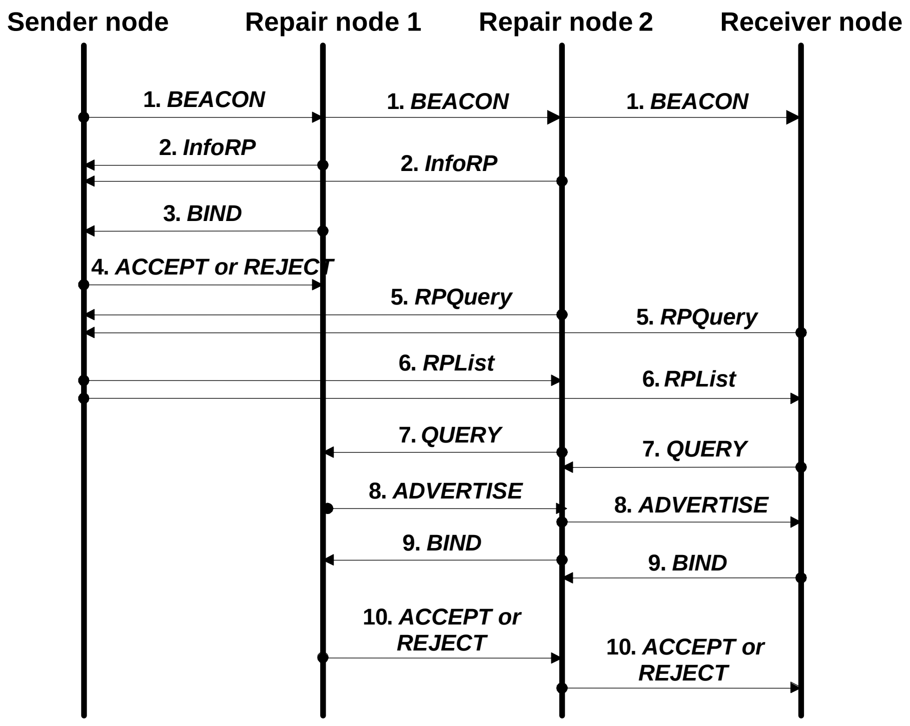


Fig. 3. Hybrid scheme

The hybrid scheme combines the advantages of both the top-down and bottom-up schemes by constructing a logical tree with a reduced message overhead within a reasonable time. In addition, it does not require receiver nodes to have any multicast capability because they do not need to multicast *QUERY* messages. The sole drawback of this scheme is that it does not scale well because it overloads the sender node. As we will see, our new hybrid scheme solves this problem by distributing some tasks normally handled by the sender node to specific nodes acting as repair node distributors. It also allows receiver nodes to adaptively re-select their repair node when they experience unacceptable error recovery delay.

3. New Hybrid Scheme

All the tree construction schemes we have reviewed so far construct logical repair trees that mimic the physical topology of the network. This logical tree is independent of the routing tree at the network layer. But, if the repair node is located at a level even lower than its receiver node in the physical tree hierarchy, the packet is not likely to be available at the repair node.

In order to avoid this, we need to construct a logical tree that reflects the physical topology. This is the natural outcome of a process in which receiver nodes always select the repair node with the shortest distance among all potential candidates. Locating repair nodes as close as possible to their receiver nodes offers the advantage that repairs nodes will answer more rapidly retransmission requests from their receiver nodes. It would be an optimal solution if we assumed that packet losses are independent events.

This is not the case for most real networks. Packet losses tend instead to happen in bursts separated by long periods of relatively error-free transmission. There is also a significant spatial correlation in losses among the receiver nodes in a multicast session. These facts are discussed in detail in [16]. Consider for instance the case of a repair node receiving its packets from the sender nodes through the same routers as its receiver nodes. Any failure of any of these routers will cause the repair node and all its receiver nodes not to receive some packets.

Our new scheme takes into account these packet loss correlations. In addition, it constructs a loop-free logical tree in a more scalable manner by decreasing sender node's workload. Our objective is to build a logical tree satisfying the following conditions.

- It should be loop-free.
- It should reflect the topology of the network while allowing receiver nodes to select a new repair node whenever the current one fails to perform its recovery tasks due to correlated packet losses.
- It should be constructed at least as fast as the fastest existing tree construction scheme with the smallest possible message overhead.

Our scheme is based on the following assumptions.

- There are N^{RPD} repair node distributors and the repair node distributors are pre-determined and the sender knows their locations and characteristics.
- RPD is the set of repair node distributor with $|RPD| = N^{RPD}$.

Like for the repair nodes, we also assume the repair node distributors are pre-determined. In our scheme, all receiver nodes will obtain the group address of the multicast, the address of sender node and the list of the repair node distributors. This process can be realized by using out-of-band mechanism such as a web page announcement.

3.1 Decentralized Sender's Workload

Our most recent scheme [14] constructs loop-free logical trees with reduced message and time overhead. At the same time, it requires the sender node to perform too many tasks. In order to reduce this sender node workload, our new scheme employs dedicated servers called *repair node distributors* established for the logical tree construction in the network. Hence, all tree nodes will contact their repair node distributor rather than the sender node to get their candidate repair node list.

3.2 Loop-free Logical Tree

Let us assume a receiver-initiated error recovery process that requires receiver nodes to send a NAK to their repair node every time they detect a packet loss. As a result, a receiver node that does not experience any packet loss will not send back any feedback to its repair node. Let us also assume that a repair node will immediately discard any packet that has exceeded its retention time. In practice, we expect these packets to be expelled whenever the repair node schedules a buffer sweep.

Observe that this NAK-based buffer management scheme does not guarantee that every repair node will always have in its buffer all the packets requested by any of its receiver nodes. In this case, the missing packets should be retransmitted from the upstream repair nodes, to provide reliable transmission. This will only work if that these upstream repair nodes have bigger retention times than their downstream repair nodes.

In addition, we need to consider that the possibility of having loop relationships between repair nodes. If this is the case, the receiver node, which had originally requested the packets, will never receive them. In the bottom-up scheme, loop relationships can be formed when two repair nodes simultaneously send a *QUERY* message to each other. The hybrid scheme can form these relationships whenever the *RPList* message includes repair nodes that are still not attached to the tree. **Fig. 4** shows an example of this situation.

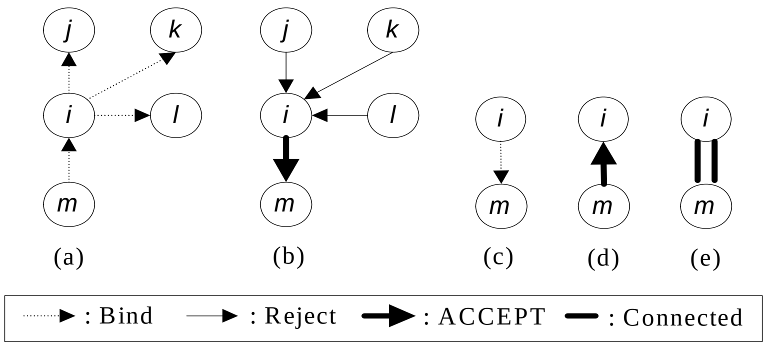


Fig. 4. Example of loop scenario

In our example shown in **Fig. 4**, (a) the repair node *i* sends a *RPQuery* message to the sender node to get some candidate repair nodes. The sender node will respond to the repair node *i* with *RPList* message including repair node *j*, *k*, *l* as candidate repair nodes. Let us also assume the repair node *m* sends a *RPQuery* message and the sender node's response includes repair node *i*. The repair node *m* will try to bind to repair node *i* as its upstream repair node. (b) Then, the repair node *i* will accept the repair node *m* as a downstream repair node. (c) Let us assume now that it was not possible to attach the repair node *i* to any of the in repair nodes *j*, *k*, and *l* because too many nodes already were attached to them. As a result, the repair node *i* will receive *REJECT* messages from all of its candidate repair nodes. In this case, the repair node *i* will re-send a *RPQuery* message. If the sender node's response includes repair node *m* and (d) the repair node *i* bind to the repair node *m*, (e) this makes a loop relationship between repair node *i* and *m*.

In order to deal with this problem, we require each repair node to report their status information to their repair node distributor with a *STATUS* message when they are attached to the tree. The repair node distributor will thus maintain two separate repair node lists. The first list contains all repair nodes that are attached to the tree. Let us call this the ON_TREE list. The other list will contain all repair nodes that are not attached to the tree yet. Let us call this the NON_TREE list. When the repair node distributor receives an *InfoRP* message from a repair node, it initially stores the repair node's information in the NON_TREE list. The information will be moved to the ON_TREE list when the repair node distributor receives a *STATUS* message from the repair node.

When a *RPQuery* message arrives, the repair node distributor selects some candidate repair nodes only from ON_TREE repair node list. This candidate list is sent to each repair node by *RPList* message. Hence, the *RPList* message lists current ON_TREE repair nodes within its domain. As a result, each NON_TREE repair node can only be attached to an ON_TREE repair node. This feature prevents a loop-relationship between repair nodes, because there is no loop relationship between the ON_TREE repair nodes.

3.3 Selecting a New Repair Node

All previous schemes have focused on constructing repair trees that mimic the physical topology of the network. In order to achieve this goal, they require each receiver node to select its repair node, having a shortest TTL distance among the given candidate repair nodes. Each receiver node will maintain connection to the repair node until the multicast session ends.

Unfortunately, transmission errors might sometimes prevent repair nodes to perform their tasks. In the context of multicast applications, these transmission errors will normally result from either data link layer errors or router buffer overflows. Since most of today's networks have fairly reliable links, packet losses are much more likely to be occasioned by router buffer overflows. As a result, packet losses will often be strongly correlated. One possible scenario is that the router intentionally discards some packets when it experiences buffer overflow than by any other cause. The discarded packets then will not be delivered to the receiver nodes. Hence, the receiver nodes attached to the router will experience continuous packet losses.

Unfortunately, these packets will not be available at its repair node's buffer whenever the repair node is under same router. This situation frequently occurs in real networks. Therefore, we have to apply a different repair node selection algorithm to deal with this problem.

Let us assume the receiver nodes and their repair node are connected to the same router. Since most packet errors will be caused by router buffer overflow, the repair node will rarely be able to answer retransmissions requests from its receiver nodes. It will have to forward these requests to its upstream repair node or to the sender node. Hence, receiver nodes will experience sudden delays. We propose to let each receiver node to change its repair node whenever it experiences very long packet loss recovery delays. In particular, receiver nodes that experience sudden long recovery delay should select another repair node for error recovery.

A straightforward method is to let each receiver node select its new repair node that has the shortest TTL distance among all its candidate repair nodes. This method is not satisfactory because the TTL values do not contain location information about repair nodes. Hence, the receiver node is likely to select a new repair node that is attached to an even lower level router than its current repair node. A second solution would to let each receiver node consult the closest router, because this router knows the network topology. However, it seems not feasible, since it requires all routers to support this functionality for multicast services.

We decided instead to add this functionality to the repair node distributors. We assume each *InfoRP* message contains a TTL distance between the repair node distributor and the repair node. Each receiver node will send a *Switch_Repair* message to the repair node distributor when it experiences a long error recovery delay. The repair node distributor will then find one or more repair nodes having the shortest TTL distance by referencing the *InfoRP* message.

These new repair nodes will be attached to an upper-level router than the current repair node of the receiver node. This process will be repeated until the receiver nodes show a reasonable recovery delay, having found an alternative repair node whose packet loss is independent of the spatial locality of the current repair node's packet loss. The summary of the processing step for our new scheme is as follows.

1. The sender node multicasts a *BEACON* message to indicate tree creation. The *BEACON* message contains all necessary information for tree creation such as the multicast group address, the address of the sender node and a list of repair node distributors.
2. When the *BEACON* message arrives, all repair nodes enroll themselves to the repair node distributor located in their respective domains by sending an *InfoRP* message. This *InfoRP* message contains the repair node's information, such as its IP address, and its node ID. The repair node distributor collects that information and adds these repair nodes to its NON_TREE list of candidate repair nodes.
3. After that, the repair node and the receiver nodes located close to the sender node reply with a *BIND* message to the sender node to indicate they want to join the tree. If the receiver nodes fill up the available capacity of the sender node, the sender node cannot accept any repair node. As this situation would affect the performance of tree construction, the sender node needs to restrict the number of receiver nodes it will accept by defining a receiver node quota. This will set aside a given space only for repair nodes.
4. If the repair nodes receive a *ACCEPT* message from the sender node, they will transmit a *STATUS* message to their repair node distributor. The repair node distributor then moves these repair nodes from its NON_TREE to its ON_TREE candidate repair node list. These candidate repair nodes are now said to be *active*.
5. Other repair nodes send a *RPQuery* message to the repair node distributor. Then, the repair node distributor selects several active candidate repair nodes for each repair node by consulting its ON_TREE repair nodes list.
6. Other receiver nodes also ask the repair node distributor which repair nodes are active in the domain when the *BEACON* message has arrived. Then, the repair node distributor will respond to the receiver node with a list of active candidate repair nodes.
7. Each tree node now sends a *QUERY* message to these candidate repair nodes.
8. Then, the candidate repair nodes will respond with an *ADVERTISE* message. This message exchange is analogous to the ping mechanism. Each node selects the best repair node, based on the TTL values from the arrived *ADVERTISE* messages. If there are some repair nodes having the same TTL distance, round-trip time is used as a second metric.
9. After that, they send a *BIND* message to the best repair node. Finally, the selected repair node accepts or rejects the node after considering its capacity. These procedures are repeated until all nodes are attached to the logical tree.

10. Receiver nodes that suffer from long error recovery delays will send a *Switch_Repair* message to their repair node distributor. The repair node distributor will then select a new repair node for these nodes.

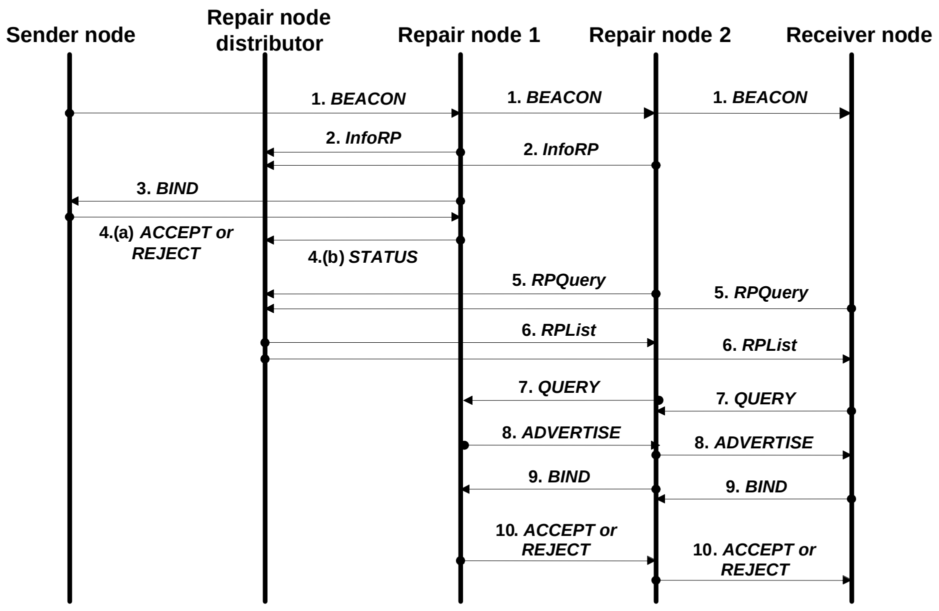


Fig. 5. New hybrid scheme

This mechanism achieves scalability by distributing the repair node selection task among the repair node distributors. Note that the processes for step 5 and 6 are performed simultaneously. The message sequence diagram of our scheme is shown in **Fig. 5**.

A receiver node will switch repair nodes if the error recovery delay exceeds the predefined threshold. This threshold value depends on the packet-sending rate of the sender node, the router's loss probability due to buffer overflow, and repair node's packet loss probability due to the link error. The rule for setting the threshold value depends on the implementation. One example would be for each repair node to suggest the expected average and maximum error recovery delay to its receiver nodes based on previous experience on the router. Each receiver node sets its threshold based on this suggestion. If the error recovery delay exceeds the threshold, it switches to a different repair node using our dynamic repair node reselection scheme.

4. Performance

4.1 Message and Time Overhead

In this section, we show the performance of the proposed tree construction scheme in terms of message and time overhead. We developed a discrete events simulator to test the proposed scheme. In our simulation, the location of each node is randomly generated in a plane of 35 by 35 elements, where each element of the plane represents the location of one node. This allowed us to perform all our simulation experiments for up to 1000 nodes. The link distance between any pair of nodes is given by calculating the TTL link distance between those two nodes on the plane. We allocate 10% of all the nodes as the repair nodes and each repair node allows 25 nodes as its receiver nodes. We set the favorable TTL distance to 5, meaning that a node is considered located close to the sender node if its TTL distance from the sender node is less or equal to 5. Additionally, we assigned 5% of all the nodes as the repair node distributors.

In order to evaluate the message overhead, we counted all generated control messages for top-down, bottom-up, hybrid and new hybrid schemes. In the bottom-up scheme, the authors did not define the time for selecting the best repair node. Instead, each receiver node sends a *QUERY* message with a TTL value up to the defined maximum. In our simulation, each receiver node sends a *QUERY* message with a TTL value equal to 1 and increases it by 1 until it reaches 10. We allow them to select the best repair node from the given repair nodes list.

We adapt the same simulation environments performed in [12] and replay the simulation. All of the simulation experiments are performed with SMPL libraries [17]. In order to evaluate the tree construction time, on running our program, we observed the elapsed time when all the nodes are attached to the tree.

Fig. 6 illustrates our measurements of the message overhead in the tree construction procedure for various numbers of nodes in the network. It shows that all hybrid schemes significantly reduce the number of the control messages for a logical tree construction, compared with top-down and bottom-up schemes. **Fig. 7** shows the simulation result for different numbers of repair nodes in the network with fixed number of tree nodes. The performance of the bottom-up scheme becomes better as the number of repair nodes increases because each receiver node can find its repair node with less *QUERY* messages. In contrast, the performance of the top-down scheme shows the opposite result. All hybrid schemes exhibit performances that are relatively independent of the number of repair nodes as the message overhead increases very slowly when the number of repair node decreases.

We also evaluated the total tree construction time for all four schemes. These times are shown in **Fig. 8**. Note that the performance of the bottom-up and all hybrid schemes is almost the same, because they all use parallel tree construction mechanisms. This result also implies that the top-down scheme is not suitable for large-scale networks, whereas our hybrid schemes perform well regardless of network size.

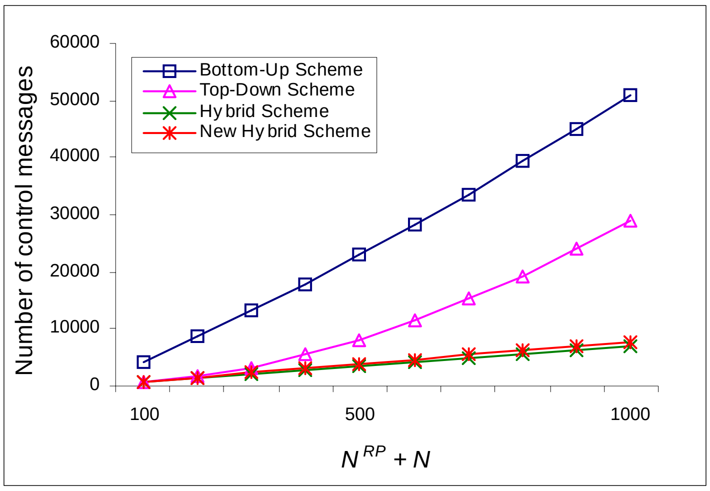


Fig. 6. Number of control messages for different number of tree nodes

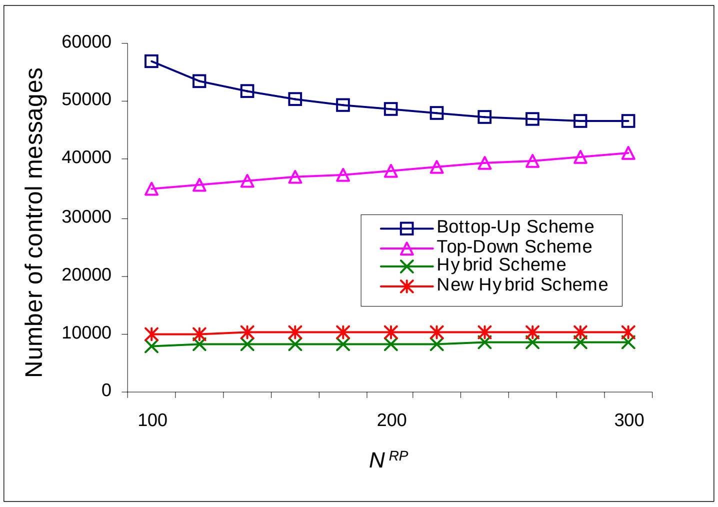


Fig. 7. Number of control messages for different number of repair nodes

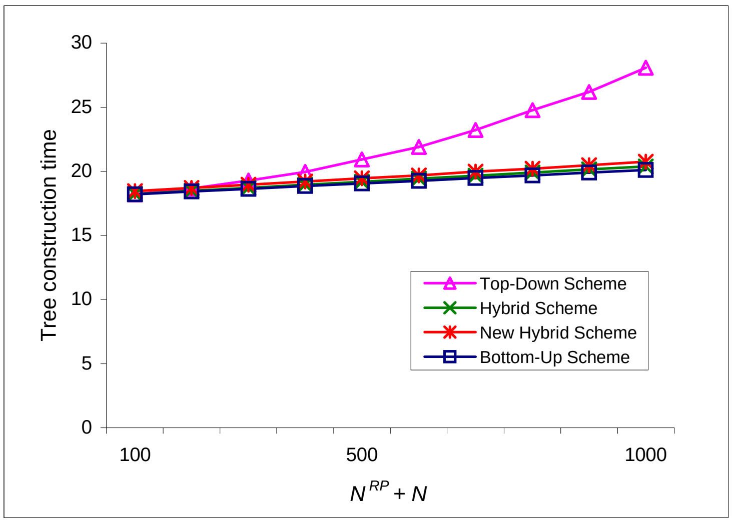


Fig. 8. Total tree construction time

The new hybrid scheme shows about an 83% reduction of message overhead compared with the bottom-up scheme, when the number of tree nodes is 1000. Under the same circumstances, it shows about a 70% reduction compared with the top-down scheme. In addition, all hybrid schemes construct the logical tree in almost the same time as the bottom-up scheme. The proposed hybrid scheme carries slightly more messages than the old hybrid scheme because it introduces new message types including *STATUS* and *Switch_Repair* message. However, this penalty is sufficiently compensated by the loop-free tree that decentralizes sender workload. It also allows receiver nodes to switch repair nodes for optimal performance. Fig. 9 shows the effect of repair node distributors on control messages processed by the sender node by assigning 5% of the nodes as the repair node distributors.

As we can see in **Fig. 9**, our scheme can significantly reduce the sender workload because processing of the *InfoRP* messages and *RPQuery* messages can be done by some repair node distributors rather than the sender node.

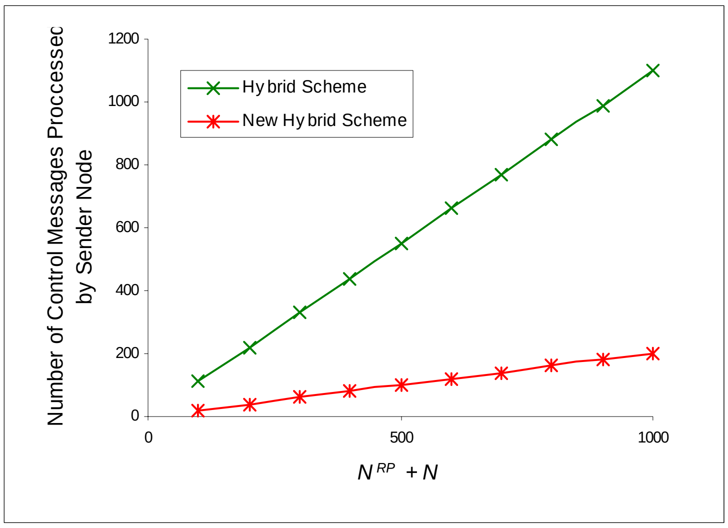


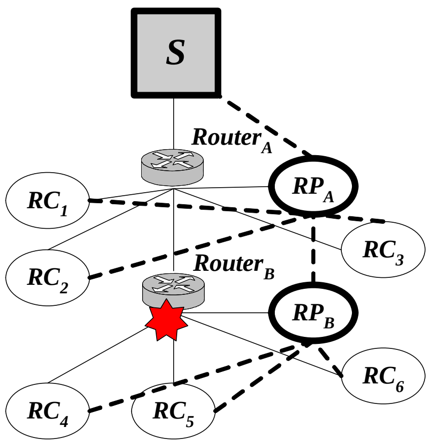
Fig. 9. The number of control messages processed by sender node

4.2 Additional Retransmissions

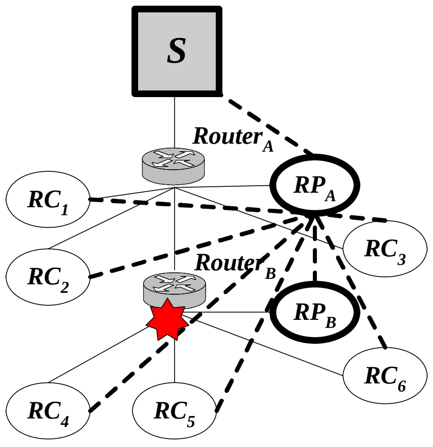
In this section, we focus on how a receiver node does not get affected by an error burst if it can reselect repair nodes. In our analysis, we show how additional retransmissions can be reduced when a receiver node can switch repair nodes whenever it experiences a temporal packet loss.

Since we focus on the behavior of each receiver node, we use a simple network model. **Fig. 10** shows a network topology we use for our analysis. Each topology has a route tree for packet transmission and a logical tree for packet error recovery. **Fig. 10** (a) represents the logical tree constructed by other schemes. In this topology, repair node *B* has a shortest TTL distance (in this example, it is equal to 0). Hence, the receiver node 4, 5, and 6 binds the repair node *B* and remains the connection until the multicast session ends. Let us assume that router *B* experiences buffer overflow. The router will drop some packets due to overfill until there is space available after some packets are discarded. As a result, until this time, any new packets arrivals are subject to overflow. As such any repair node or receiver nodes under this router will experience continuous packet loss for the series of packets. Consequently, the requested packets will not be available in the repair node's buffer, because it also has not successfully received the packets.

Recall that our new scheme requires the receiver nodes to reselect their repair node when they experience a sudden recovery delay as shown in **Fig. 10** (b). In this example, receiver node 4, 5, and 6 will switch to repair node *A* whenever router *B* losses too many packets. Since the new repair node *A* is attached to a different router, it will not be affected by the behavior of router *B*. We also assume that underlying reliable multicast protocol uses a NAK-based buffer management scheme [6][7][8][9][10][11][12][13][14][15], where each receiver node sends a NAK to its repair node whenever it detects a packet loss. The repair node will retransmit the packet and discard some packets from its buffer after a time interval.



(a) Fixed logical tree



(b) Dynamic logical tree

Fig. 10. Simple network topology

In addition, we make the following assumptions:

- There are N receiver nodes for each repair node. Hence, each repair node is responsible for resending the packets requested with NAKs from N receiver nodes.
- Each router node r_X has a packet loss probability equal to $1 - r_X$.
- The probability p_i that a node i receives a packet that has gone through routers r_1, r_2, …, r_M is given by:

  $$p_i = R_M S_i \,,$$

  where $R_M = \prod_{i=1}^{M} r_j$ and $1 - S_i$ is the probability of a packet loss outside a router.
- Each receiver node has an independent NAK timer *NAK_TIMER*$_i$.
- The repair node has an independent NAK timer as well as a PACKET_DISCARD timer it uses to decide when to discard packets from its buffer.

Packet losses outside a router include a corrupted packet or lost packet caused by a link error. In our analysis, we assume a receiver node will not try to fix the error. Instead, it requests retransmission of the corrupted packet to the repair node.

Let us first consider the scheme placing a repair node B and its N receiver nodes on the same router and consider the probability that the repair node will not able to retransmit a requested packet. Let be M_B the probability that the repair node B does not have the requested packet in its buffer.

There are two cases to consider. First, the requested packet can be missing because the repair node B never received it. Hence, additional retransmissions will be required until the packet arrives at the repair node B. We call this case M_{B1}. The probability $P(M_{B1})$ will be given by

$P(M_{B1})$ = P(the repair node B did not receive the packet)

$\times$ P(some receiver nodes did not receive the packet)

$$= (1 - R_M S_B)(1 - R_M \prod_{i=1}^{N} Si) \tag{1}$$

Second, the packet will not be available if the receiver nodes request it after the repair node B has already discarded it. We call this case M_{B2}. Even then, some packets could still be available if other NAKs for the same packet arrive before the timer expired. We must also consider the impact of lost NAKs. If the NAKs of all the receiver nodes that did not receive the packet fail to reach the repair node B, then the repair node B will discard the packet before it receives a second request for that packet from one of the receiver nodes. If b_i represents the probability that a NAK sent by receiver node i and received by repair node B has reached B before the timer expired, this probability $P(M_{B2})$ will be given by

$P(M_{B2})$ = P(the repair node B correctly received the packet)
$\times$ P(some receiver nodes did not receive the packet and none of the NAKs sent by these receiver nodes reached the repair node on time)

$$= R_M S_B [(\prod_{i=1}^{N} (1 - (1 - R_M S_i) r_M S_i b_i) - R_M \prod_{i=1}^{N} S_i)] \tag{2}$$

since the NAK sent by a receiver node i will have to go through router r_M.

Hence, a realistic estimate of the packet missing probability $P(M_B)$ is given by

$$P(M_B) = P(M_{B1}) + P(M_{B2}) \tag{3}$$

Observe that in the first case M_{B1}, the repair node B has not received the packet that is requested by one or more of its receiver nodes. In that case, the repair node B will send a single NAK to its original sender node and hold on the NAKs for its receiver nodes as it knows it will be able to service them soon enough. This single NAK from the repair node B will not be counted as an additional retransmission, because the sender node makes no distinction between repair node and receiver nodes.

Let us consider now our new scheme, which places the repair node in one router upstream from its N receiver nodes. This means that the packets sent by the sender node to the repair node will only have to go through routers r_1, r_2,…, r_{M-1}. Conversely, the NAKs sent by the receiver nodes will now have to go through routers r_M and r_{M-1}. Hence all receiver nodes will have a feedback loss probability equal to

$$(1 - r_M r_{M-1})(1 - S_i).$$

The probability $P(M_{A1})$ is now given by

$P(M_{A1})$ = P(the repair node A did not receive the packet)
$\times$ P(some receiver nodes did not receive the packet)

$$= (1 - R_{M-1} S_A)(1 - R_M \prod_{i=1}^{N} S_i) \tag{4}$$

Similarly, the probability $P(M_{A2})$ will be given by

$P(M_{A2})$ = P(the repair node A correctly received the packet)
$\times$ P(some receiver nodes did not receive the packet and none of the NAKs sent

by these receiver nodes reached the repair node on time)

$$= R_{M-1} S_A [(\prod_{i=1}^{N} (1-(1-R_M S_i) r_M r_{M-1} S_i b_i) - R_M \prod_{i=1}^{N} S_i)] \tag{5}$$

As before, the packet missing probability $P(M_A)$ will given by

$$P(M_A) = P(M_{A1}) + P(M_{A2}) \tag{6}$$

Fig. 11 compares the two schemes and shows how the number of receiver nodes per repair node affects the probability of not finding a requested packet in the repair node buffer.

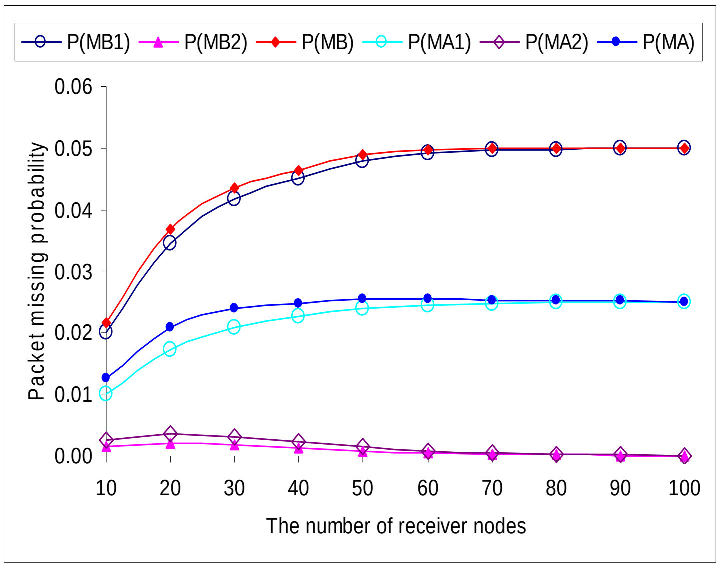


Fig. 11. Missing packet probability

The following parameters are used for our analysis. First, we set the probability b_i for each receiver node i to 0.8, meaning that each repair node keeps in its buffer recently received packets for a time sufficient to handle 80 percent of the retransmission requests. If we apply our heuristic buffer management scheme [5], we can increase the probability, because some unreliable nodes will send delayed ACKs. In this simulation, we are assuming the conventional NAK-based scheme and the NAK transmission delays are roughly distributed according to a normal law. Therefore, the repair node keeping packets in its buffer for the average NAK transmission delay of its receiver node plus one standard deviation should be able to answer more than 80 percent of all packet retransmission requests.

Second, there are up to 100 receiver nodes. Each receiver node i has a packet loss probability 1– $R_M S_i$ between 0.05 and 0.2. And the repair node A and repair node B have packet loss probabilities, 1– $R_{M-1} S_A$, and 1– $R_M S_B$, (0.025 and 0.05) respectively assuming repair node A is under a more reliable router experiencing less loss and also that router A is independent of router B's spatial locality packet losses.

We also estimated that repair node B experiences a common packet loss, either due to buffer overflow or link error as all of its receiver nodes in its local group as long as the loss is universal in the group. This is a rather conservative estimate given that repair node B and all its receiver nodes are affected to same router *B* and are thus equally affected by any failure of router *B* to forward a packet.

The results show that the repair node *B* will require a lot of additional retransmissions either from its upstream repair nodes or sender node, because it will not have in its buffer many packets that can be requested by any of its receiver nodes. Even though the $P(M_{B2})$ decreases with the group size *N*, the $P(M_{B1})$ tends to increase when the same group size *N* increases. As a result, the comprehensive performance of the previous schemes is always significantly worse than our scheme.

On the other hand, repair node *A* does not require many additional retransmissions, since it has an independent spatial locality of packet losses to its receiver nodes. These results indicate that our scheme provides a faster error recovery for receiver nodes and reduces unnecessary network traffics between the repair nodes. As shown in **Fig. 6** and **Fig. 8**, our scheme constructs this logical tree with almost the same message overhead as the scheme requiring the smallest number of messages while being as fast as the fastest scheme.

4.3 Error Recovery Delay

Additional retransmissions directly affect error recovery delay for the receiver nodes. They increase the error recovery delay since the repair node cannot retransmit the requested packet immediately from its buffer. In order to show that the dynamic repair node reselection allows receiver nodes to alleviate the problem, we evaluate the average recovery delay for each receiver node and compare the results with those of the old schemes that did not have a dynamic repair node reselection. To evaluate the minimum delay difference, we assume that the additional retransmission from repair node *B* is always correctly transmitted from the repair node *A*. Otherwise, the difference would be more prominent. We also assume that the round trip time between each network entity is 30ms and the sender node transmits 10,000 multicast packets, which roughly represents a transfer of 10 megabytes with a packet size equal to 1 kilobyte. **Fig. 12** shows the difference in terms of error recovery delay when the receiver nodes are attached under repair node *A* and *B*.

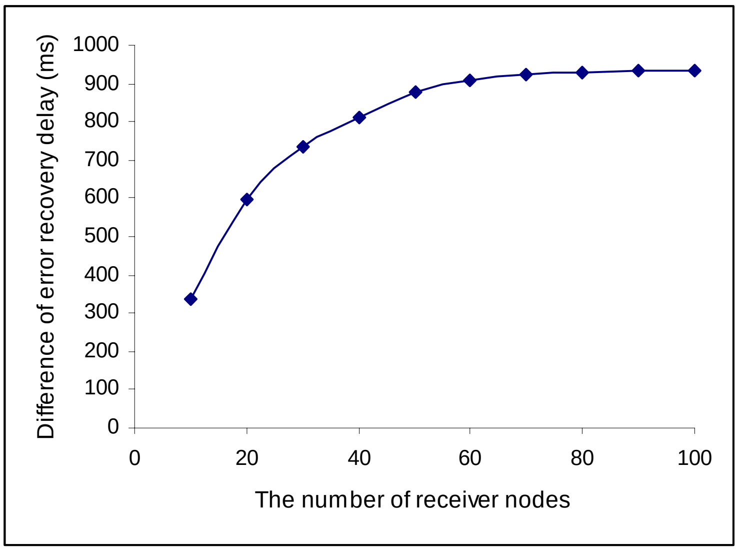


Fig. 12. Difference of error recovery delay

5. Conclusion

We have described how the error recovery tree is constructed. There were three approaches for a logical tree construction: top-down, bottom-up, and hybrid scheme. We have suggested an improved hybrid approach eliminating many limitations of previous schemes.

However, we need to consider packet losses that tend to occur in bursts separated by long periods of relatively error-free transmission. There is also a significant spatial correlation in loss among the receiver nodes in a multicast session. In order to consider these properties, we have proposed a scalable recovery tree construction scheme. Our scheme achieved this goal (1) by distributing some of the workload of the sender node sender's workloads to nodes acting as repair node distributors and (2) by requiring receiver nodes to select a new repair node when they experience unacceptable error recovery delays. We defined some conditions for a well-organized tree. Simulations results indicate that the tree, constructed by our new scheme, satisfy these conditions. First, our new hybrid scheme constructs a loop-free logical tree within a reasonable time. Second, our scheme performs the same regardless of the network size and its message overhead is reasonable. Finally, our tree is adaptive to correlated packet losses. As a result, our new scheme can combine the advantages of previous schemes. We believe that this process makes the logical tree stable and widely suitable when it is applied to the multicast services.

Acknowledgement

This work was supported in part by the National Science Foundation under grant CCR-9988390.

References

[1] H. Cho, S-H Lee, Y Choi, F. Yu and S-H Kim, "Efficient Overlay Multicast Protocol in Mobile Ad hoc Networks," *In Proc of the 65th IEEE Vehicular Technology Conference*, pp. 51-55, Apr. 2007.

[2] L. Lao, J-H. Cui, M. Gerla and S. Chen, "A Scalable Overlay Multicast Architecture for Large-Scale Applications," *IEEE Transactions on Parallel and Distributed Systems*, 18(4), pp. 449-459, Apr. 2007.

[3] D-N. Yang and W. Liao, "On Bandwidth-Efficient Overlay Multicast," *IEEE Transactions on Parallel and Distributed Systems*, 18(11), pp. 1503-1515, Nov. 2007.

[4] J. S. Baek and J. F. Pâris, "A Buffer Management Scheme for Tree-Based Reliable Multicast Using Infrequent Acknowledgments," *In Proc. of the 23rd IEEE International Performance Computing and Communications Conference*, pp. 13-20, Apr. 2004.

[5] J. S. Baek and J. F. Pâris, "A Heuristic Buffer Management and Retransmission Control Scheme for Tree-Based Reliable Multicast," *ETRI Journal*, 27(1):1-12, Feb. 2005.

[6] S. Floyd et al., "A Reliable Multicast Framework for Lightweight Sessions and Application-Level Framing," *IEEE/ACM Trans. on Networking*, 5(6):784-803, Dec. 1997.

[7] T. Gemmel et al., "The use of Forward Error Correction in Reliable Multicast," IETF draft-ietf-rmt-info-fec-02.txt, Oct. 2002.

[8] M. Kadansky et al., "Reliable Multicast Transport Building Block: Tree Auto-Configuration," IETF Internet Draft, draft-ietf-rmt-bb-tree-config-01.txt, Nov. 2000.

[9] J. C. Lin, S. Paul, "RMPT: A Reliable Multicast Transport Protocol," *In Proc. of the 15th IEEE Conference on Computer Communications*, pp. 1414-1424, Mar. 1996.

[10] B. Whetten and G. Taskale, "The Overview of Reliable Multicast Transport Protocol II," *IEEE Networks*, 14(1):37-47, Jan.-Feb. 2000.

[11] Z. Xiao, K. P. Birman and R. Renesse, "Optimizing Buffer Management for Reliable Multicast," *In Proc. of the 2002 International Conference on Dependable Systems and Networks*, pp. 187-202, June 2002.
[12] S. J. Koh et al., "Configuration of ACK Trees for Multicast Transport Protocols," *ETRI Journal*, 23(3):111-120, Sep. 2001.
[13] R. Yavatkar, J.Griffioen and M. Sndan, "A reliable dissemination protocol for interactive collaborative applications," *In Proc. of the ACM 1995 Multimedia Conference*, pp. 333-344, Nov. 1995.
[14] J. S. Baek and E. J. Lee, "An Improved Logical Tree Construction Scheme for Tree-Based Reliable Multicast," *In Proc. of the 2003 International Conference on Telecommunication Systems*, pp. 110-121, Oct. 2003.
[15] S. K. Kasera, J. Kurose and D. Towsley, "Buffer Requirements and Replacement Polices for Multicast Repair Service," *In Proc. of the 2nd Network Group Communication Workshop*, pp. 5-14, Nov. 2000.
[16] M. Yajnik, J. Kurose and D. Towsley, "Packet Loss Correlation in the Mbone Multicast Network," *In Proc. of the IEEE Global Internet Conference*, pp. 94-99, Nov. 1996.
[17] M. MacDouall, "Simulation Computer Systems, Technologies and Tools," MIT Press, 1987.

Jinsuk Baek is Assistant Professor of Computer Science at the Winston-Salem State University (WSSU), Winston-Salem, NC. He is the director of Network Protocols Group at the WSSU. He received his B.S. and M.S. degrees in Computer Science and Engineering from Hankuk University of Foreign Studies (HUFS), Korea, in 1996 and 1998, respectively and his Ph.D. in Computer Science from the University of Houston (UH) in 2004. Dr. Baek was a post doctorate research associate of the Distributed Multimedia Research Group at the UH. He acted as a consulting expert on behalf of Apple Computer, Inc in connection with Rong and Gabello Law Firm which serves as legal counsel to Apple computer. His research interests include scalable reliable multicast protocols, mobile computing, network security protocols, proxy caching systems, and formal verification of communication protocols. He is a member of the IEEE.

Jehan-François Pâris is Professor of Computer Science at the University of Houston. He was formerly faculty at Purdue University and the University of California, San Diego. Dr. Pâris received his Ingénieur Civil degree from the Université Libre de Bruxelles, Belgium, his Diplôme d'Etudes Approfondies from the Université de Paris VI, France, his Licence et Maîtrise en Informatique from the Facultés Universitaires de Namur, Belgium and his Ph. D. in EECS from the University of California, Berkeley. His research interests include memory hierarchies, scalable reliable multicast protocols, distribution protocols for video-on-demand, and distributed systems security. He is a member of the ACM and a senior member of the IEEE.