Reliable Multicasting over a Datagram Network

CS 640
Project Description

Revision 1.0

Fall 1999

1 Introduction

As a motivation for this project, consider an application that distributes over the network newspapers or movies from a content provider (e.g., New York Times or HBO) to a large population of users. The application uses *network multicasting*, i.e., the network 'clones' the transferred file to all the receivers along a distribution tree, instead of sending the file to each receiver separately. Multicasting is a more resource-efficient way of group communication for both the network and the sender.

In this project, we focus on *reliable multicasting*, i.e., all receivers are able to get correctly the complete transferred file, and on *single-sender multiple-receivers* multicast groups, as opposed to *multiple-senders multiple-receivers* groups. The multicast protocols that will be designed and implemented have to be *scalable* to a large number of receivers, which is a major requirement for multicasting in general.

2 Project Overview

You will design and implement protocols for a reliable multicast transport layer and for a datagram multicast-capable network layer. The underlying link layer will be emulated with the exchange of UDP datagrams between two hosts through AF_INET sockets. An application for this protocol hierarchy will be provided to you in order to test and evaluate your project. This application will be a multicast-type of FTP, and it will be linked with your implementation. This handout provides a detailed description of the project, including the definition of the API that you must implement, and a list of requirements and checkpoints that have to be met.

The first component of the project is a *datagram multicast-capable network layer*. The network layer consists of the following protocols:

1. Datagram Network Protocol (DNP)
2. Dynamic Routing Protocol (DRP)
3. Group Management Protocol (GMP)

DNP is responsible for *forwarding* unicast or multicast datagrams from a network node to one or more other nodes. The DNP communication is not reliable, i.e., packets can be corrupted, duplicated, or lost. The *routing tables*, which specify to DNP the link(s) on which a packet with a certain destination address should be sent to, are constructed using the DRP and GMP protocols. DRP is responsible for 'discovering' the topology of the network, in order to construct unicast routing paths which are both efficient (they are short), and robust (they avoid loops). DRP has to be able to discover and deal with dynamic changes in the network topology, caused by link or node failures and reactivations. GMP is responsible for communicating between network nodes information about the membership in a multicast group. The membership information is dynamic, in the sense that receivers are able to *join* or *leave* the multicast group whenever they want. More information on the network layer is provided in Section 3.

The second component of the project is a *reliable multicast transport layer*. The transport layer consists of the following two protocols:

1. Reliable Announcement Protocol (RAP)
2. Reliable Multicast Transport Protocol (RMTP)

RAP is a unicast client-server protocol which allows *potential receivers* to get information about upcoming multicast transfers from a sender. The sender is the RAP server, while the potential receivers of the multicast transfer are the RAP clients. The RAP communication takes place before the start of the multicast distribution and it provides the receivers with the following information:

- name/description of the transferred file
- length of the file
- starting time of the multicast distribution
- multicast address for the distribution

RMTP uses the information that has been announced with RAP in order to perform the reliable multicast distribution. RMTP has to deal with *loss-recovery*, through Negative Acknowledgments (NACKs) and retransmissions from the sender, and with *flow-control*, using some type of *window advertisements* (credits) from the receivers, depending on their available buffer space. Both loss-recovery and flow-control have to be performed in a scalable manner; this requires that the sender should not maintain state for each receiver, and that the sender should not get feedback packets (NACKs or flow-control related packets) from each receiver. These requirements can be met with appropriate *aggregation* and/or *pruning* of these feedback packets as they flow from the receivers to the sender in the the reverse multicast tree. For loss-recovery and flow-control reasons, the network layer (and specifically GMP) has to be *RMTP-aware*, i.e., packets which are related to an RMTP distribution have to be identified in each network node[1] in order to be aggregated, pruned, or just to update some transport-related state in that network node. More information on the transport layer is provided in Section 4.

[1]This is a case of layer violation of course.

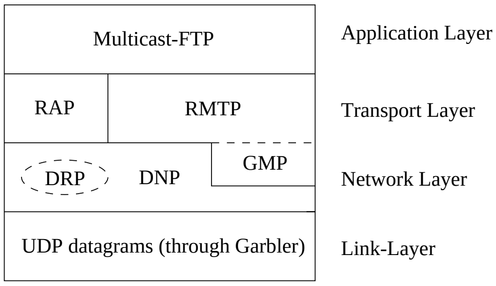

Figure 1: The protocol hierarchy.

Figure 1 gives the hierarchy of the protocols required in this project. You will have considerable freedom in designing the service interfaces, the packet-header formats, and the implementation mechanisms for these protocols. However, since your code will be linked with an application that will be provided by the TAs, you must adhere rigidly to the transport APIs defined in Section 4.3 of this document (and more formally specified in the form of a header file which will be made available soon).

3 Network Layer Protocols

3.1 Datagram Network Protocol (DNP)

The task of forwarding packets from one node to another is performed with DNP. The two other network-layer protocols, DRP and GMP, can be thought of as internal components of DNP, responsible for creating the routing tables and for managing the multicast groups membership. Actually, the DRP and GMP packets are 'encapsulated' in DNP packets, i.e., they have, as every other packet in this project, a DNP header, and they are forwarded from network nodes as any other DNP packet.

A central consideration in DNP is the *addressing* of packets. Since DNP supports both unicast and multicast forwarding, there will be two classes of addresses. A unicast-class address is associated with each network node (known from the network configuration file), while a multicast-class address is associated with each multicast group. A multicast address is known initially only to the sender of that multicast group. How the sender acquires this address is out of the scope of this project; we guarantee though, that the same multicast address will not be assigned to multiple active groups. The address field will be 16 bits, with the higher-half of the address space reserved for multicast addresses.

DNP does *datagram forwarding*, as opposed to *virtual-circuit forwarding*, in the sense that there is no signaling for the establishment of a forwarding path from the sender to the receiver(s) before the communication starts. Each packet is routed independently, based on the information in its destination address only.

DNP is *unreliable*, in the sense that packets can be corrupted, lost, reordered, or duplicated in the network. The reliable transport protocols need to detect such problems, and deal with them.

DNP packets are limited to 1024 bytes, including their header. For longer messages, the transport protocols have to do the corresponding fragmentation and reassembly of the message to/from different packets.

Some of the design issues for DNP follow:

- packet header, field specifications
- forwarding mechanism for unicast and multicast packets
- structure of routing tables
- memory management of packet buffers
- interactions with GMP and DRP

3.2 Dynamic Routing Protocol (DRP)

DRP is responsible for constructing the unicast routing tables, based on the current connectivity in the network. You have to use some version of *link-state routing* in order to construct the routing table for unicast forwarding in each network node. This table contains information of the form: 'A packet destined to unicast address X should be sent to the output link that connects me (the local node) to node Y'. The routing tables for multicast forwarding can be constructed using the unicast routing tables and the group membership information that GMP maintains in each network node. In this case the table contains information of the form: 'A packet destined to the multicast address X should be sent to the output links that connect me to nodes $Y_1, Y_2, ...Y_N$'. An algorithm for constructing the multicast routing tree, based on the reverse paths from the receivers to the sender, is described in Appendix-1 (you can use other multicast routing mechanisms if you want).

An important challenge in the design of DRP is how to deal with link and node failures and reactivations, which dynamically change the network topology. Some type of *HELLO* messages will have to be exchanged between nodes, in order to dynamically discover the network's topology. For simplicity, the failures that will be introduced will never partition the network.

Note that routing packets can also be dropped in the network. Consequently, you have to decide whether to implement a reliable transfer mechanism within the routing protocol. An alternative would be to use probabilistic mechanisms in order to ensure that the routing protocol will recover from the losses in a timely manner (for example, using more frequent HELLO messages and redundant flooding of the routing changes).

Some of the design issues for DRP follow:

- packet header, field specifications
- unicast routing algorithm

- multicast routing algorithm
- state machine
- mechanism for detecting topology changes
- mechanism for avoiding routing loops
- mechanism for dealing with lost routing packets
- interactions with GMP and DNP

3.3 Group Management Protocol (GMP)

In order to perform multicast forwarding each network node has to know the active multicast groups, and which adjacent network nodes are part of the multicast tree for each group; this is achieved with GMP. GMP can be used for two purposes. First, the receivers inform their upstream network node of their desire to *join* or *leave* a certain multicast group, specified by a certain multicast address. Second, the GMP packets can flow inside the network *upstream*, from the receivers to the sender, in order to setup in each node of the multicast routing tree some *forwarding state* for the multicast group. This second role of GMP can be coupled with DRP in order to build the multicast routing tree (see Appendix-1)[2].

An important aspect of the GMP forwarding state is that it does not require explicit signaling in order to be established or released in a network node. On the contrary, it is *soft-state*, in the sense that it automatically expires, and the corresponding data structures are released, if a certain GMP *REFRESH* packet does not update it periodically. You have to carefully select the state-refresh period, taking into account that the GMP packets may also be lost.

Some of the design issues for GMP follow:

- packet header, field specifications
- structure of GMP state in each network node
- state machine
- period of state-refreshing
- mechanism for generating/pruning/aggregating the GMP packets
- interactions with DRP and DNP

[2]Note that GMP is significantly different from the IGMP protocol that is used in the Internet for multicast group management.

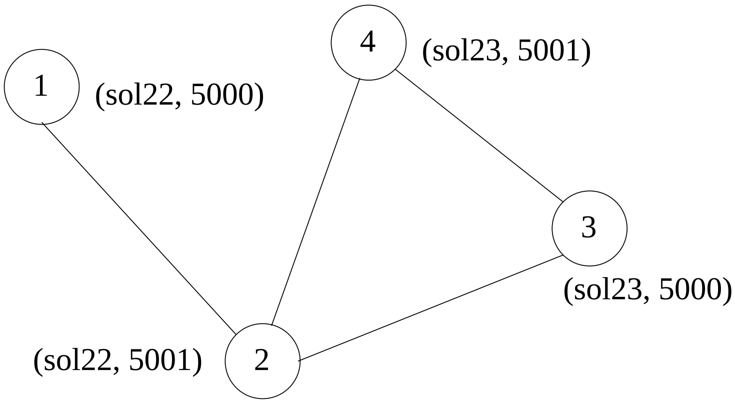

Figure 2: The network topology that corresponds to the given configuration.

3.4 Network Configuration

Each network node will be emulated by a separate execution of your program. Nodes will run on a subset of the instructional Sun workstations Sol1-Sol40 and Nova1-Nova40 with each node corresponding to a (machine address, UDP port) pair. Therefore, each execution of your program will represent a virtual node communicating with other virtual nodes, each through its own UDP port. There may be many such virtual nodes running on a single workstation or running on several workstations.

A configuration file will specify the virtual nodes in the network and the links between them. The node identifiers in the configuration file also describe the unicast addresses of these nodes. Your program will take two command line arguments: the name of the configuration file, and the identifier of the node that this process will be emulating. The configuration file will then have to be parsed, so that each node knows the addresses of its adjacent nodes in the initial network topology. The format of the network configuration file is shown next, while the topology of the network represented by that file is shown in Figure 2.

```
Node 1 sol22, 5000 links 2
Node 2 sol22, 5001 links 1 3 4
Node 3 sol23, 5000 links 2 4
Node 4 sol23, 5001 links 2 3
```

3.5 Garbler

In order to test the reliable-transport capabilities of the project, your code will interact with a *Garbler* that you will be provided with. The Garbler is a set of routines that intercepts a DNP datagram just before it is sent to a UDP port. The Garbler potentially corrupts, duplicates, or drops the datagram, prior to forwarding it to its destination. We will provide more information about the Garbler in the next few weeks.

4 Multicast Layer Protocols

4.1 Reliable Announcement Protocol (RAP)

RAP is a reliable unicast unidirectional transport protocol, which provides single-packet transport capabilities. Specifically, RAP can only transfer an *announcement* (short string) from a node, which acts as a RAP server to a requesting RAP client. We guarantee that the RAP message (including the RAP header) will be limited to one DNG packet, so that you don't have to worry about message fragmentation/reassembly in RAP.

RAP can be used for different applications, carrying different information. In this project it will be used to announce information from the sender of a multicast group to any potential receivers, before the multicast distribution starts. This information includes the following fields:

1. name/description of the file to be multicasted
2. length of the file
3. starting time of the multicast distribution
4. multicast address for the distribution

The RAP header should be generic, independent of the announcement semantics. The announcement should be carried in the payload of the RAP packet as a character string, together with appropriate field delimiters for easy parsing. For example, a RAP packet, as used in this project, can transfer the following string which announces two upcoming multicast events:

```
"1: NAME='Star Wars, Episode I (movie)' LENGTH='102934098'
STARTS='933012245' M_ADDRESS='45542'
 2: NAME='The Sixth Sense (movie)' LENGTH='101856567'
STARTS='933013349' M_ADDRESS='45544'"
```

The RAP transfers should be reliable against packet corruption, or losses. For this reason, RAP packets have to include a checksum (see RMTP) and appropriate retransmission timeouts. A mechanism such as the three-way handshake in TCP connection establishment can be used for the implementation of RAP.

Some of the design issues in RAP follow:

- packet structure, field specifications
- state machine
- checksum length and computation
- retransmission timeout computation

4.2 Reliable Multicast Transport Protocol (RMTP)

RMTP is a reliable multicast unidirectional transport protocol, for the transfer of a file from a sender to the set of receivers that have joined the corresponding multicast group. Before a certain node can act as an RMTP sender, it has to first announce the multicast event using RAP. Similarly, before a node can act as an RMTP receiver, it has to first get the announcement from the sender through RAP, and then join the corresponding multicast group. The RMTP sender starts the actual transfer at the time that has been specified in the RAP announcement (we will assume that all network nodes have well-synchronized clocks, which is true in a LAN environment that uses the Network Time Protocol in general).

Some simplifications in the design of RMTP follow:

1. A node will not become an RMTP receiver after the starting time of the multicast transfer (i.e., there are no "late joins" in the multicast group after the start of the transfer).
2. An RMTP sender will not abort a multicast transfer, after the transfer has been announced with RAP.

Error Detection: RMTP has to protect packets against bit-errors (corruption) through a checksum included in the packet's header. The exact checksum length and computation algorithm will be determined by you.

Loss Recovery: The receivers must have a way to tell the sender that one or more packets have been lost, so that the appropriate retransmissions take place. The detection of lost packets can be triggered by out-of-order arrivals[3], and timeouts.

You have significant freedom in designing the loss-recovery mechanism. For scalability purposes, though, the receivers should not send individual acknowledgements to the sender, i.e., the positive or negative acknowledgements should be aggregated/pruned as much as possible in the network. Also, the sender should not maintain state for each of the receivers. A possible loss-recovery mechanism is described in Appendix-2.

Flow-Control: The receivers must have a way to tell the sender how fast to send packets. Otherwise, the available packet buffers in the receivers may be exhausted, causing drops of incoming packets. The process of rate-limiting the sender based on feedback from the receivers is called flow-control.

You have significant freedom in designing the flow-control mechanism. As in the case of loss-recovery, though, the sender should not maintain separate state for each receiver, and you should try as much as possible to prune or aggregate any flow-control feedback flowing from the receivers to the sender. A possible flow-control mechanism is described in Appendix-3.

Some of the design issues in RMTP follow:

[3] Although out-of-order arrivals do not guarantee that the missing packets were lost.

- packet header, field specifications
- state machine
- sequence numbers (note that the transferred file in an RMTP session can be several megabytes)
- checksum length and computation
- loss recovery: structure of NACK packets, retransmission timeout computation, pruning of NACKs in the network, pruning of retransmissions in the network, interaction between RMTP and GMP (for pruning of NACKs and retransmissions), etc.
- flow-control: detailed description of the mechanism, structure of any related control packets, pruning or aggregation mechanisms of such control packets in the network, interactions with loss recovery, etc.

4.3 RAP and RMTP Application Programming Interface (API)

Since you will be linking your code with an application provided by the TAs, you must adhere exactly to the following API for the transport protocols RAP and RMTP. We will provide a header file with the exact specification of this interface later.

Just as an Internet application is specified by an address-port pair, the applications in this project are specified by an address-service_id pair. This pair makes up a Service Access Point (SAP):

```
struct SAP_640 {
  u_short address;
  u_short service_id;
};
```

- `int start_all(int argc, char **argv)`

  Start-up all network services. The application layer will call this function with the standard arguments to main() before any network communication is attempted. Return -1 in case of any error.

- `int terminate_all()`

  Shutdown all network services. Return -1 in case of any error.

- `u_short register_service(u_short service_id)`

  Register a service. Before any data can be received by an application, it must register itself as a service (like getting a port number in UNIX). The return value is the service_id that is registered. To have the system assign you an unreserved and unused service, specify the argument as zero. Returns 0 in case of any error.

- `int rap_post_announce(SAP_640 local_sap, char* announcement,`
  `u_short announce_len)`

  Announce a character string (of a given length) with RAP at a given local SAP. The structure of the string is specified by the application. Returns 0 if success, -1 if the string is too long for a single DNP packet, and -2 in case of other errors.

- `int rap_get_announce(SAP_640 local_sap, SAP_640 remote_sap,`
  `char* announcement, int max_len)`

  Get a RAP announcement at a local SAP from a RAP server (addressed by a remote SAP). Returns -1 if the RAP server does not reply, -2 in case of other errors, and the length of the announcement otherwise.

- `int rmtp_send(SAP_640 local_sap, u_short multicast_addr, FILE* fp,`
  `int file_len)`

  Send a given file of a certain length to a multicast address from a local SAP. This is a blocking function. It returns 0 if the file transfer has been completed and there are no other members in the multicast group. Returns -1 if the file does not exist, and -2 in case of other errors. This function will be called at the time that the multicast session has been scheduled for.

- `int rmtp_recv(SAP_640 local_sap, u_short multicast_addr, FILE* fp,`
  `int file_len, u_short max_rcv_window,`
  `SAP_640 remote_sap, time_t start_time);`

  Receive at a local SAP a file of a given length from a multicast address. The SAP of the multicast group sender is also given, together with the starting time of the multicast session, and the initial maximum receiver window. This function is non-blocking. It is up to the application to check that the transfer is complete. Returns 0 if success, -1 if the multicast session has already started, and -2 in case of other errors.

- `int rmtp_recv_wind(SAP_640 local_sap, u_short max_rcv_window);`

  Modify the maximum receiver window to a given value, possibly while the multicast transfer is in progress. Returns 0 if success, -1 if the new window size would lead to packet drops, and -2 in case of other errors.

- `int rmtp_abort(SAP_640 local_sap)`

  Abort the receipt of a multicast session at a local SAP. The receiver has to explicitly leave the multicast group. Returns 0 if success, and -1 in case of any errors.

- `void dump_tables(short statis_flag)`

  When the input argument is one, your program should generate and report (in a well-formatted way) the routing table, the multicast groups in the local node, and all other statistics that you maintain. This function will be used for debugging and evaluation.

5 System Structure

Figure 1 provides an overview of the various protocols in this project and how they relate to each other. Note that this figure does not dictate the exact system structure that your project should have. The manner in which you structure your program is extremely important. **We are requiring that you implement all functionality in a single multi-threaded process**. You still have considerable design leeway in many areas, however. Some of the issues that you must decide are: how to assign threads to tasks, how data is going to flow through the various layers of your system, how mutual exclusion will be provided on key data structures, and how threads will synchronize with each other.

An important point is that your program should collect during run-time statistics from both the network and the transport layer. These statistics can include the packet forwarding rate, the loss-rate, the number of active multicast groups, the number of receivers in each group etc. The display of these statistics will be switched on/off using the *dump_tables()* function.

6 Requirements, Design Report, and Deadlines

The general requirements that you should follow in both the design and implementation phases are:

1. Reliability: You need to consider any packet loss/corruption scenarios, as well as any node/link failure cases.
2. Scalability: You need to consider how to make your protocols and implementation scalable for a large number of network nodes and multicast groups.
3. Performance: Since performance is a ruler to measure how good is your design and your implementation, we are going to give credits to projects that include performance-enhancing mechanisms.

You will have to submit a design proposal that gives a detailed description of how you plan to address the issues raised in the previous sections. We will comment on your design proposal and return it to you for revision. A final version, which will be graded, must be submitted soon after. Your design document should be as detailed as possible. The better and more detailed the design, the easier it will be to do the implementation. At a minimum, your design proposal should describe the following:

Complete Specifications
For each protocol, specify in detail the algorithms, state machines, packet formats, and other design issues for that protocol. Some of these issues have been mentioned in this handout.

General Issues

- Detailed division of responsibility among partners
- Description of the data structures to be used
- Naming and programming conventions
- Plan for testing the individual components of your program
- Detailed implementation schedule
- Performance enhancement strategies (e.g., reducing the number of memcpy
- Description of the statistics that you will collect and report during run-time at each network node calls)

Deadlines

- First version of design report: October 11
- Revised version of design report: November 1
- First demonstration: November 23 (this demo will not include the garbler and link/node failure scenarios)
- Final demonstration: December 20,21

Appendix-1: Reverse path multicasting

The receiver nodes send periodically GMP JOIN messages to the sender. These messages, however, are processed and aggregated by the intermediate network nodes, in order to form the multicast routing tree for that group. The tree that is formed in this manner is the aggregation of the reverse unicast paths from the receivers to the sender. Since these paths are probably the shortest (depending on DRP), the resulting multicast tree is expected to also be efficient and without loops.

Consider the eight-node network shown in Figure 3. The sender of the multicast group is located in node 1, while the three receivers are located in nodes 4, 7, and 8. If the DRP shortest path from node 7 to node 1 is through 5 and 2, the resulting multicast tree will be as shown in the figure. The traffic that goes to receivers R1 and R2 is aggregated in the link from 1 to 2, i.e., node 1 sends the multicast packets only once to node 2; node 2, on the other hand, sends each packet of that group to both nodes 4 and 5. Note that the multicast forwarding state that is maintained in each node needs to know only the adjacent downstream links of the tree, and not the receiver addresses in that subtree. This example also illustrates that the multicast tree that is formed using the reverse path multicasting mechanism is not always optimal; an optimal tree in this example would connect node 8 to the multicast group through node 7. If you decide to use another multicast routing mechanism, please discuss with the TAs first.

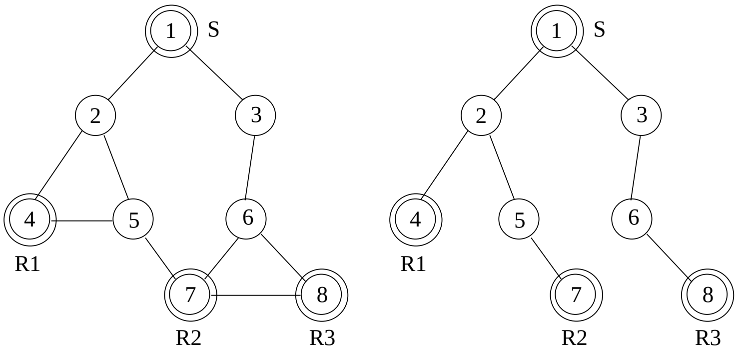

Figure 3: Reverse path multicasting: the network (left) and the multicast tree (right).

Appendix-2: A possible loss-recovery mechanism for RMTP

The receivers send negative acknowledgements (NACKs) to the sender when they think that a packet has been corrupted or lost. This action can be triggered by bad checksums, out-of-order arrivals, and retransmission timeouts. You can decide to have NACKs for a single packet, or for a range or set of lost packets.

The flow of NACKs from the receivers to the sender should be pruned by the nodes of the multicast tree whenever the NACKs are redundant. In the example of Figure 3, suppose that a packet is lost in the link $1 \rightarrow 2$. A NACK sent by receiver R1 is processed by the (RMTP-aware) GMP state of node 2 to indicate that the lost packet of that multicast group has been requested by the subtree rooted at node 4. Now, suppose that the NACK from receiver R2 arrives at node 2 before the retransmission of the lost packet reaches that node. This NACK is redundant and is pruned by node 2, after noticing that this particular retransmission should be also forwarded to node 5 upon its arrival. The retransmission state in node 2 is released when the retransmitted packet arives at that node.

The sender, on the other hand, retransmits lost packets to the entire multicast group. It is again the responsibility of the network to prune the retransmitted packets, so that they arrive only to the subtree of the multicast group that has requested them.

Appendix-3: A possible flow-control mechanism for RMTP

A receiver sends after each packet arrival, or periodically, a *Maximum Acceptable Byte (MAB)* message (assuming per-byte sequence numbers) which indicates the *maximum sequence number that the receiver can store in its receiving buffer*. If a packet with a larger sequence number arrives to that receiver, it will probably have to be dropped.

The MABs are aggregated as they flow upstream to the sender. The aggregation rule is: *Upon the arrival of at least one MAB from all the downstream links of a multicast group, generate a new MAB message advertising the* **minimum** *of the received MAB advertisements, and send it* *upstream to the multicast tree.* This aggregation rule limits the rate of the multicast distribution to the rate of the slowest receiver.

It is possible that a receiver, or an entire subtree of the group, becomes very slow or temporarily disconnected, and so it does not generate MABs for sometime. How long should the upstream node of that subtree wait before generating its own aggregate MAB? What if that subtree or receiver is permanently disconnected or crashed? Note that it is not a good idea to rely on the GMP REFRESH timeout for dealing with 'crashed' receivers, since that timeout should be considerably large[4]. To deal with this issue, a network node may need to also generate periodic MABs (called *advanced MABs*), upon the expiration of a timeout, even though not all downstream nodes have sent their MABs. The advanced MABs can cause packet drops at the slow receivers and retransmissions, but they can also be useful for dealing in a timely manner with receivers that have crashed, or that have aborted without explicitly leaving the group.

[4]The GMP REFRESH timeout has to be large, because after a receiver is dropped out of the group, it is not allowed to join again if the session is in progress.