

Generic Multicast Transport Services: Router Support for Multicast Applications

Brad Cain
Network Research
Nortel Networks
Billerica, MA 01821, USA
bcain@nortelnetworks.com

Don Towsley
Dept. of Computer Science
Univ. of Massachusetts
Amherst, MA 01003-4610, USA
towsley@cs.umass.edu

CMPSCI Technical Report TR 99-74
October 1999

Abstract

The development of scalable end-to-end multicast protocols is a tremendous challenge because of the problems in feedback implosion and transmission isolation. In this paper we describe a set of simple services, called *Generic Multicast Transport Services (GMTS)*, which are implemented in routers for the purpose of assisting the scalability of end-to-end multicast protocols. GMTS provides a rich set of filtering and aggregation functions that support feedback suppression and sub-tree multicast operations which are desirable for many multicast transport protocols. We describe the GMTS architecture, the set of services, and provide examples of how the services can be used to support reliable multicast, repair services, any-casting, and multicast congestion control.

1 Introduction

The development of scalable end-to-end multicast protocols poses a tremendous challenge to network protocol designers. For example the development of reliable multicast protocols has received considerable attention in recent years. Most protocols are based on an end-to-end solution [1, 5, 10] and have found the problem of scaling to 1000s or even 100s of receivers daunting. The primary obstacles to the development of scalable protocols have been *feedback implosion* and *transmission isolation*. The first of these concerns the difficulty for a large multicast application to limit feedback from receivers to a data source or to each other. The second concerns the difficulty of limiting the transmission of data to the subset of a multicast group that requires it.

There have been several proposals for adding functionality to routers for the purpose of improving the performance of multicast applications, particularly reliable multicast. Papadopoulos and Parulkar [7] introduced additional forwarding functionality to a router which would allow each router to identify a *special outgoing interface* over which to transmit a particular class of packets. They showed how this *turning point functionality* could be used to improve the performance of reliable multicast protocols. Levine and Garcia-Luna-Aceves [4] proposed the addition of *routing labels* to routing tables which could be used to direct packets over specific interfaces. One of these, called a distance label, was shown to be quite useful in reliable multicast for directing requests for repairs to nearby repair servers. The third and, perhaps most relevant proposal is the PGM

protocol [9]. Briefly, PGM is a reliable multicast protocol which uses negative acknowledgements (NACKs). The PGM protocol is an end-to-end transport protocol that contains a router component which performs NACK suppression and retransmission subcasting functionality. Our work is especially motivated by PGM and the recognition the utility in exporting a set of flexible, simple router-based functionality (such as was used in implementing PGM) for the purpose of protocol design. This would simplify the design of a large class of scalable multicast transport protocols.

In this paper, we present a set of Generic Multicast Transport Services (GMTS) that are intended to help protocol designers deal with these two problems. These services are designed to assist in the scaling of receiver feedback information and in providing subcasting services for large multicast groups. They consist of simple filtering and aggregation functions that reside within routers.

Signaling protocols are used from hosts to set up and invoke these services. Briefly, a session source first initializes one or more desired services on its multicast tree using GMTS setup messages. The GMTS-capable routers on the tree then aggregate feedback from receivers and/or isolate transmissions through the use of filters set by either the sender or the receivers. For robustness, periodic transmissions of setup messages on the multicast tree are used to refresh GMTS state in the face of routing changes and other possible errors. It should be stressed that GMTS services are only invoked for certain signaling packets; data packets are not treated any different and will not cause any additional processing in routers.

GMTS is not intended to provide sophisticated services which are difficult or impossible to implement in routers. GMTS services are implemented at the IP layer and provide unreliable *best-effort* services. Transport protocols which make use of GMTS must be robust in the face of failures and the absence of GMTS-capable routers in the network.

At a superficial level, GMTS resembles active networking. However, unlike active networking proposals, GMTS objects are simple and fixed (i.e. not dynamically uploadable modules). We feel that a small number of fixed services which, if made available, can benefit multicast transport protocols while, at the same time, are reasonable candidates for implementation in a router. Furthermore, GMTS objects are lightweight and contain only a small amount of state. This is in contrast to recently proposed active repair services that have been proposed by the active networking community, [2, 3, 8], which require the caching of packets for the purpose of providing retransmissions.

The paper is organized as follows. In the next section, we present a simple example of how GMTS can be used the context of a reliable multicast transport protocol. Section 3 introduces the generic transport services architecture. Section 4 describes a GMTS object, its state and methods, which is the fundamental building blocks for a GMTS session. Applications to the development of multicast transport protocols are given in Section 5. The contributions of the paper are summarized in Section 6.

2 A GMTS Example: Reliable Multicast

Before describing the details of GMTS, we present a simple example in the context of a PGM-like reliable multicast protocol. A more detailed description of a reliable multicast protocol based on forward error correction (FEC) can be found in Section 5.1.

Consider a NACK-based reliable multicast protocol which places the responsibility of packet loss detection on each receiver. Each time that a receiver detects a loss (based on a gap in the sequence numbers of the packets that it receives), it unicasts a request for a repair (NAK) to the

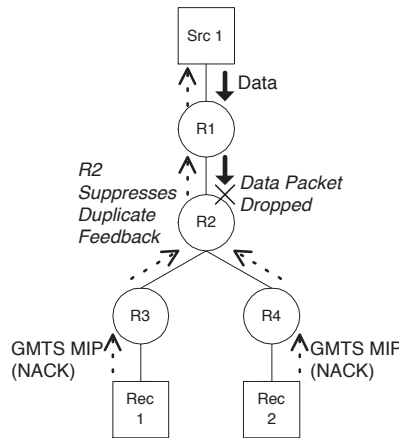


Figure 1: Example of network support for transport protocols.

sender. Upon receipt of a NAK for a specific packet, the sender retransmits the packet to all receivers.

This protocol faces considerable challenges in dealing with multiple NAKs for the same packet. First, there is the problem of the sender having to process many NAKs. Second, there is the problem of limiting the number of retransmissions to the same packet. GMTS can be used to (partially) solve these two problems. Prior to the transfer of any data, the application sets up a NAK aggregation object at each GMTS-capable router using an setup messages. This object is set up to suppress NAKs for the same packet. In addition, the router maintains information regarding the interfaces over which it has received NAKs so that it can subcast the retransmission own the portion of the multicast tree that contains receivers requiring a retransmission of the packet.

In Figure 1, we show how GMTS can be used to aggregate feedback information in a reliable multicast transport protocol. In this figure, a multicast source (Src 1) is transmitting to two receivers (Rec 1 and Rec 2). The data packets from Src 1 are treated as regular multicast packets and forwarding accordingly. On the link between router R1 and router R2, a data packet is lost. Assuming a NAK based reliable multicast protocol, this loss will cause the receivers to send a NAK to the source for the data that was lost. In the example, receivers use GMTS to send the feedback (i.e. the NAKs) to the source. GMTS router R2 treats these NAKs in a special manner, suppressing the redundant NAKs to the source. Therefore, only one NAK arrives at the source. We can see from this example that GMTS routers only certain types of packets require additional processing at GMTS routers and that the majority of end-to-end packets are forwarded according to normal multicast forwarding rules (i.e. without additional router processing).

3 Generic Multicast Transport Services

A GMTS session is instantiated by a multicast sender and operates over (unreliable) IP multicast. A GMTS session, identified by source and group address, consists of objects located in GMTS-capable routers on the multicast tree connecting the source to the receivers. A set of signaling protocols serve as the interface to GMTS objects and are used to initiate objects and to invoke object methods remotely. If a multicast application contains more than one sender, a GMTS session is established for each of them.

GMTS methods are only invoked for certain types of signaling packets generated by an end-

to-end multicast protocol; most data packets are sent end-to-end as regular multicast packets and are not treated specially by GMTS-capable routers. For example, in a reliable multicast transport protocol, NAKs would be trapped by GMTS-capable routers while regular data packets would flow through the normal router forwarding path. We stress this to show that GMTS does not incur significant overhead in multicast routers as only periodic signaling packets are specially processed.

In this section, we provide a high-level description of the objects, the signaling mechanisms, and illustrate how these components can be combined through a simple example.

3.1 GMTS Objects

Many GMTS services (i.e. feedback suppression) require that routers keep a certain amount of state in order to provide those services. The relationship between services and the state required for these services is analogous to the relationship between object-oriented classes and their methods. It is for this reason that we use object-oriented programming terminology to define a GMTS object. A GMTS object is a set of methods (i.e. the actual GMTS services) and their associated supporting state which exist in routers on a multicast tree.

GMTS objects are the fundamental components which provide a fixed set of simple services through their methods. GMTS objects consist of state-dependent filtering methods, the state required for these filters, and methods for modifying this state. GMTS objects are very flexible in that they can support a rich set of filters and state manipulation functions. In the paper, we illustrate the flexibility of these objects by illustrating their use in supporting an FEC-based reliable multicast protocol similar to PGM [9], end-host based repair services, multicast congestion control, and anycast.

GMTS objects are instantiations of available GMTS object types. A GMTS object consists of state variables and several methods that can be remotely invoked by either a source or a receiver. GMTS object types are predefined; that is, they are fixed specifications that are implemented in router software. The primary goal of these object types is to be able to set up and tear down simple filtering mechanisms that can be used by the routers to reduce the amount of end-to-end control traffic generated within a multicast session. GMTS objects are the actual per session instantiations of available GMTS object types.

GMTS objects and their methods are accessed via two signaling mechanisms. The first is the mechanism used to set up and refresh GMTS object state. State set up packets (SSPs) are used to set up objects and refresh their state. GMTS method invocation packets (MIPs) are used to invoke GMTS methods.

Because of our adherence to the soft-state philosophy, a timer is associated with each object. This timer is set at the time that the object is first initialized (i.e., the time that the router first receives a set up packet (SSP) listing that object), and reset each time that the router receives an additional SSP listing that object. If the timer expires prior to the receipt of a new set up packet, the object is deleted at the router. GMTS objects may have additional soft-state timers as required by the object. For example a NAK suppression object, as might be used in a reliable multicast protocol, would maintain a timer with the state associated with a particular sequence number.

GMTS object methods may be invoked on the multicast tree in either a reliable or an unreliable manner. Each set up packet contains a list of objects for which the sender wishes to instantiate (or to refresh the state for). Optionally, a sender may list options for methods specifying which

methods will require reliable or unreliable delivery. For a reliable method, a GMTS router will verify the receipt of a MIP to the next hop GMTS router. For an unreliable method, a router will only transmit the MIP once to next hop GMTS router.

GMTS object methods also include access rights for each method stating whether the sender, receiver, or both may invoke a given method. The GMTS sender may set access rights to all objects.

3.2 GMTS State Setup Packets

A state setup packet (SSP) declares objects to be set up in GMTS capable routers along a multicast forwarding tree. An SSP is multicast by the sender to its multicast group. As it traverses the multicast distribution tree, it is trapped by all GMTS-capable routers. Each router then creates objects corresponding to the object type declared within the SSP. An SSP can also be used to refresh an object, as will be discussed later, and to describe particular options for objects.

The second purpose of the SSP is to inform a GMTS router of the address of the next upstream GMTS router. This is used to allow GMTS sessions to traverse regions of non-GMTS capable routers. We define the GMTS tree as the tree of GMTS capable routers along the multicast forwarding tree.

GMTS must be robust in the face of network topology changes. Another use of a SSP is to refresh GMTS state and to re-instantiate it when the network topology changes. SSPs are sent periodically so that, in the event of a routing change, GMTS state will be set up along the new multicast routing tree.

3.3 GMTS Method Invocation Packet

The sender and receivers use a second signaling mechanism to remotely invoke methods on objects residing in GMTS routers. It consists of the transfer of method invocation packets (MIPs) which identify the methods to be invoked along with their required parameters. In addition, a MIP may include actual end-to-end data.

MIPs are primarily used by receivers to invoke object methods. These methods typically perform some kind of state or control message aggregation. Senders may also invoke methods. For example, a receiver might invoke a method to suppress NAKs during a reliable multicast session. A sender might invoke a method in order to change the timeout value of a GMTS object. Methods flowing toward receivers are always multicast. Methods flowing towards the source are always unicast between GMTS-capable routers.

GMTS methods may be invoked either in a reliable or unreliable manner. GMTS SSPs may contain per method options which specify whether a method should be invoked reliably or unreliably. Reliable methods are ones where a GMTS router ensures reliable delivery of the MIP along the GMTS tree. Unreliable methods are ones where a GMTS router does not ensure reliable delivery of a MIP.

The particular end-system protocol using GMTS needs to map GMTS services into its own set of variables. GMTS routers have no knowledge of the function of a particular piece of data on which a method is invoked. In order to provide an agnostic view of an end-system protocol to routers, we introduce GMTS *identifiers*. A GMTS identifier is associated with a particular object instantiation. It is used by an end-system protocol to map protocol specific data to GMTS

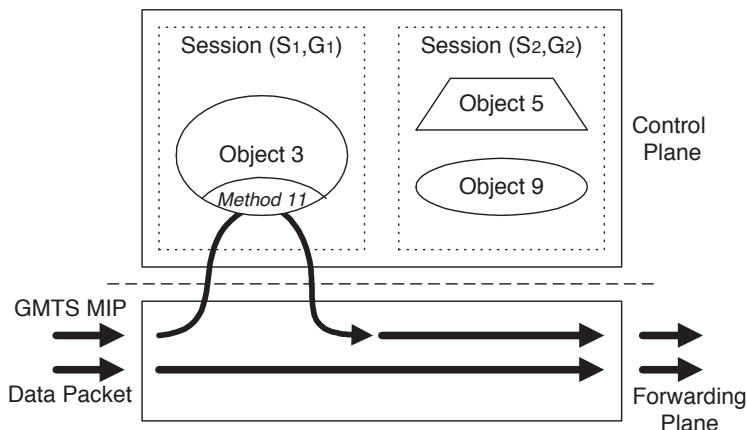


Figure 2: Paths that data packets and MIPs take through a GMTS router.

objects. For example, a NAK based reliable multicast protocol may create an identifier for its sequence number space.

To a GMTS capable router, an identifier associates an incoming packet with an object. MIP packets contain both an identifier and a method. A router then looks up the object using the identifier and applies the particular method to the packet. In order to make GMTS available to all types of protocols, routers have no knowledge of the mapping between objects and end-system protocol parameters. The source of GMTS SSP packets creates identifiers for each object that it wishes to create.

Figure 2 illustrates the behavior of a GMTS router when presented with a MIP and with a non-GMTS packet. The first packet is a GMTS MIP, which causes a router to perform special processing on the packet. This packet, for example, would contain end-to-end protocol signaling or would be a special data packet with special forwarding rules (e.g. subcasting). The second packet is a regular multicast data packet which was sourced from either a non-GMTS session or GMTS session. As stated earlier, GMTS sessions use regular multicast packets for the bulk of their data. These packets are forwarded normally, and without latency cost according to multicast forwarding rules.

We show two GMTS sessions within a router, identified by their tree forwarding entries. The first session, identified by the (S_1, G_1) forwarding entry has one object with identifier three. As the GMTS MIP packet traverses the router, it invokes a particular method (method 11) on object 3. In most cases, the MIP would contain signaling feedback or be a special data packet to be subcasted. In the second session, two objects of differing types are shown; in this case, a host is using services provided from two different object types.

3.4 GMTS Operation

The following describes the steps required in setting up a GMTS session:

1. The transport protocol designer decides which services will be needed for his/her protocol. Typically, GMTS services are chosen to achieve end-to-end scalable signaling for a multicast transport protocol. The services required of the end-to-end protocol will dictate which object types (and their associated methods) should be setup and invoked.

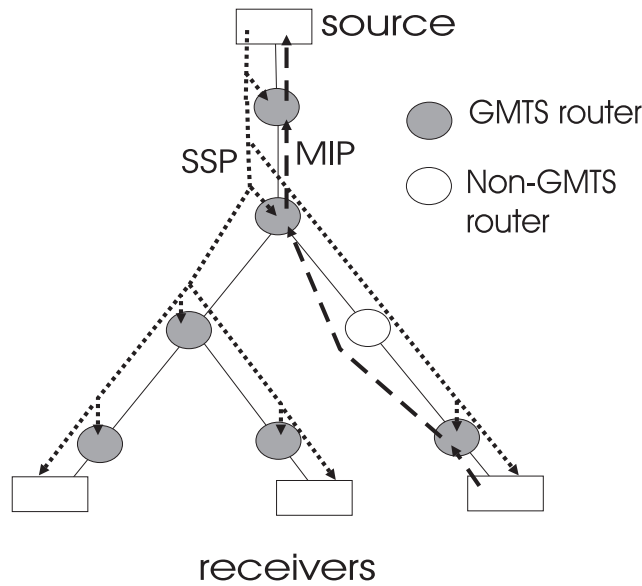


Figure 3: Operation of GMTS.

2. The transport protocol sender uses GMTS SSPs to instantiate the objects desired by the protocol. This is illustrated in Figure 2 with the multicast of a SSP. SSPs contain a list of all objects desired, their object types, their identifiers, and their timeout values.
3. The sender periodically issues SSPs, which flow over the multicast tree; each GMTS capable router on the tree instantiates the objects that were requested by the sender.
4. Receivers receive the SSPs which notify them of the identifiers to use for the instantiated objects
5. Receivers (or senders) issue MIPs which cause methods to be invoked on the multicast tree. MIPs contain the actual end-to-end signaling information as well as the type of service they desire from the routers. MIPs contain an object identifier and the method ID that they wish to invoke. Figure 2 illustrates the rightmost receiver sending a MIP towards the source. Observe that it is unicast between GMTS routers.
6. Periodically SSPs are sent from the sender to refresh the objects along the GMTS tree.

3.5 Interaction with Routing Protocols

GMTS works over all types of multicast forwarding trees. However, GMTS state is per session state and requires per session state in routers. If a source-based tree routing protocol is used to forward multicast datagrams, then this per session state will already exist in routers in the form of multicast forwarding entries. If a shared tree type multicast routing protocol is being used, then GMTS per session state must be maintained on the shared tree. This is simply because GMTS provides services per session and therefore must keep per session state in routers.

| Private Variable Name | Description |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>start_window</i> | Lowest allowable sequence number which is kept and on which methods may be invoked. |
| <i>end_window</i> | Highest allowable sequence number which is kept and on which methods may be invoked. |
| <i>state</i> | Array of sequence number state where the index is the sequence number and the value is a non-negative integer. <i>state(i)</i> identifies the sequence state for sequence number <i>i</i> . |
| <i>interface_vector</i> | Each sequence number may optionally have an interface vector that represents a set of interfaces for which the given predicate has matched. The interface vector array is an array of interface vector state where the array index is the sequence number state and the value is a linked list with state per interface on the router. <i>interface_vector(i)</i> identifies the set of interfaces marked for sequence number <i>i</i> . |
| <i>object_expiration_timer</i> | A value specified in seconds for the timeout of the object. |
| <i>state_expiration_timer</i> | A state expiration timer exists per sequence number state. It specifies the timeout in seconds of that sequence number state. |

Table 1: Private variables for the GMTS General Purpose Object.

4 GMTS General Purpose Object (GPO)

In this section we describe a general purpose GMTS object (GPO) type that provides a rich set of services and functionality to end-system multicast protocols. The GPO contains several methods which provide a set of core services which we believe are most useful to protocol designers and are reasonable for implementation in routers. These services are the following:

- **Suppression:** simple feedback suppression towards a source.
- **Predicate Suppression:** suppression of redundant feedback based on the boolean result of one or more comparison operations.
- **Subcasting:** the ability to forward a packet to a subset of the multicast forwarding tree.

4.1 Sequence Space

Many end-to-end multicast protocols make use of a sequence space. In order to apply suppression type operations per sequence state (i.e. NAK suppression), the GPO includes a large sequence number space which can be used to identify a set of state. A sequence number can be used to represent many different parameters in multicast transport protocols such as a packet or a burst. This sequence number is used to reference all state in the GPO. The private variables associated with the GPO are listed in Table 1.

The GPO sequence space exists to allow per sequence state for protocols which make use of a sequence space. *start_window* and *end_window* specify a range of sequence numbers which are currently valid and for which state is kept. Associated with sequence number i are two state variables. The first, *state_array*(i) is a nonnegative integer. The second, *interface_vector_array*(i) is a vector which contains as many integer valued components as outgoing interfaces at the router, including the one towards the sender. As we will observe shortly, this vector is used to determine the interfaces over which receiver and sender initiated methods are to be invoked. We will expand on this forwarding functionality shortly.

4.2 Methods

We now describe the methods associated with the GPO. Methods are used both for state maintenance and for the actual GMTS services. A summary of all GPO methods are listed in Table 2. Some methods make use of predicates in order to provide for more sophisticated processing of MIPs. These are discussed in the next section.

A GMTS sender uses the *modify_window* method to update the GPO sequence number window. Ordinarily, all state associated with sequence numbers outside the new window is discarded. The method, *clear_state*, is used to explicitly clear state associated with packets whose sequence numbers fall within the range defined by *start_range* and *end_range*. The method, *modify_state*, is used to explicitly set the state associated with a sequence number.

The method, *rcvr_update*($n, v, pred, f_s, f_v$), does the following. If $pred(v, state(n))$ is true, then $state(n) := f_s(v, state(n))$ and $interface_vector(n) = f_v(v, interface_vector(n), vec)$ where vec is the vector of all 0s except for a one in the position corresponding to the interface over which the method was invoked, If $state(n)$ has changed, then the *rcvr_update* method is invoked on the link directed towards the source with $v = state(n)$, and all other arguments unchanged. Otherwise, nothing occurs. It is expected that a receiver will normally invoke this method.

The method, *mcast_update*($n, v, pred, f_s, f_v$), behaves in a similar manner except that, whenever the predicate is true, the method is invoked on all outgoing links that are part of the multicast tree except the one that it arrived over. Receivers and senders can invoke this method. Note that the MIP containing the *mcast_update* is multicast over all outgoing links in the routing table and unicast on the outgoing link on the path to the next upstream GMTS router on the path towards the source associated with the GMTS session.

Last, an invocation of *forward*($n, v, g_s, g_v, data$) results in the invocation of the same method on the routers attached to all of the outgoing interfaces (except for the one that the original invocation arrived on) for which the corresponding components in *interface_vector*(n) are greater than v . In addition, $state(n) := g_s(v, state(n))$ and $interface_vector(n) := g_v(v, interface_vector(n))$. Upon receipt by a receiver, the parameters *data*, *state*, and n are delivered to the application.

In addition, there are timers associated with each individual state component and with the object itself. The constants, *state_expiration_timer*, and *object_expiration_timer* contain the values that these timers are set to when they are initialized. The object timer is set each time that the object is instantiated using an SSP. If it expires, the object is destroyed. The state timer for sequence number i is set each time that it is modified as a result of an method invocation. Last, the timer constants can be set by the application using the *set_object_timer* and *set_state_timer* methods.

| Method Name | Arguments | Description |
|---------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>modify_window</i> | <i>start, end</i> | Allows the sequence window to be changed to new values. |
| <i>rcvr_update</i> | <i>n, v,</i> <i>pred, f_s,</i> <i>f_v</i> | Method usually invoked by receiver to invoke suppression or an election depending on a predicate. If the predicate is TRUE, then set the state using $f_s()$ and the interface vector using $f_v()$. Predicates and functions have predefined types. |
| <i>clear_state</i> <i>forward</i> | <i>start, end</i> <i>n, v,</i> <i>g_s, g_v,</i> <i>data</i> | Clears all sequence state in the provided range. Method usually invoked by the sender to allow directed transmission of data on interface k such that $interfaces_vector(k) - value > 0$. State and interface vectors are set using the functions $g_s()$ and $g_v()$. The data for the directed transmission is included in the MIP packet. |
| <i>mcast_update</i> | <i>n, v,</i> <i>pred, f_s,</i> <i>f_v</i> | Similar to the <i>rcvr_update</i> method except that the MIP is multicast on all multicast interfaces except the one that it arrived on. |
| <i>modify_object</i> <i>_timer</i> | <i>v</i> | Set the object timer to <i>timer_value</i> . |
| <i>modify_state</i> <i>_timer</i> | <i>n, v</i> | Set the sequence number state timer to <i>timer_value</i> . |

Table 2: Methods for a GPO.

| $pred(x, y)$ | $f_s(x, y)$ | $f_v(a, v_1, v_2)$ | $g_s(x, y)$ | $g_v(a, v)$ |
|----------------------|-------------|----------------------------|------------------|------------------|
| $x < y, x > y$ | $x + 1$ | $v_1 \& \& v_2$ | $x - 1$ | v |
| $x \leq y, x \geq y$ | x | $v_1 v_2, a \times v_2$ | x | $\max(0, v - a)$ |
| $x = y$ | | $\max(v_1, a \times v_2)$ | $\max(0, v - 1)$ | |

Table 3: Predicates and functions.

4.3 Predicates and Operations

A number of methods require the specification of a predicate used to determine if state should be modified. If the application of the predicate is true, the given operations are performed. Functions may be supplied for describing the modifications to object state. For modifying the state of a sequence number, the $f_s()$ function is used. For modifying the interface vector, the $f_v()$ function is used. For modifying the interface vector with the forward method, $g_v()$ is used. Each predicate and function has a well-known value so that it may be easily specified in the MIPs. Table 3 illustrates predicates and functions that can be used with the GPO described above.

4.4 Reliable method invocation

Method invocations are available with two reliability semantics: best effort (unreliable) and reliable. The reliability semantics are chosen at the time that an object is initialized (and, of course, reinitialized with each successive SSP). In the case that full reliability is chosen, an ACK is returned to the downstream GMTS router as soon as the method has completed execution. Reliability is ensured through the use of a timer at the router initiating the invocation coupled with the ACK. The ACK contains v and f_s , parameters from the method invocation.

GMTS provides two ways of sending acknowledgements for reliable method invocation. The first is that the ACK is unicast to the GMTS router that invoked the method. The second is similar, except that the ACK is multicast to the multicast group over the interface on which the method was invoked. Normally, only the GMTS router that initially invokes the method will receive the ACK. However, it is possible that it will be received by some other GMTS router on a broadcast type subnet as well. (This may be useful for suppressing similar method invocations from other downstream routers. If a GMTS router receives an ACK for a method that it did not invoke, the router uses it to set the state for that router as if it had been initiated by a *rcvr_update* invocation.

Last, multicast ACK reliable invocation support is not provided for all methods. In particular, it will not be provided for *multicast_update*.

5 Examples

In this section, we present several examples of protocols that can be constructed using the general purpose object. In particular, we describe a receiver oriented reliable multicast protocol that can use forward error correction (FEC) to reduce the number of retransmissions required, a protocol that can be used to select a repair server in a reliable multicast session, an anycast protocol, and a protocol that can be used to add scalable congestion control to a multicast session.

5.1 A Reliable Multicast Protocol with FEC

FEC has been shown to be especially effective in the implementation of reliable multicast, [6]. Consequently, in this section we describe a receiver oriented protocol that uses FEC to reduce the number of retransmission requests. The design is similar to that of PGM [9]. However, the purpose of this exercise is not to develop new protocol but to illustrate the flexibility of GMTS.

Briefly, data is grouped into blocks consisting of B_{size} packets. each time that a receiver detects that it has not received all of the packets in a particular block, it forwards a request for a number of parity packets equal to the number of missing packets. Upon receipt of this request, a GMTS-capable router checks to see if an earlier request for at least as many parity packets has already passed through for this data block. If so, the parity request (PR) is discarded. If there has been no previous request for as many parity packets, then the request is forwarded towards the source. In either case, a record is made of the number of parity packets required to be sent down the interface over which the parity request arrived.

The source creates parity packets in response to a parity request and sends them down the tree to the receivers. Each GMTS router sends as many parity packets over each outgoing interface as has been recorded for that block.

This protocol can be constructed using a single object at each router whose purpose is to perform feedback suppression and parity transmission subcasting. It includes the following functionality:

1. Periodic multicast of source path packets (SSPs) for the purpose of instantiating the object at each router and refreshing it subsequently.
2. Propagation of parity requests from receivers to sender.
3. Parity transmission using subcasting.

The sender must maintain the following state:

1. A block sequence number window (n_{start}, n_{end}); this corresponds to the blocks which the sender is prepared to create parity packets for.
2. The sequence number for the next block to be transmitted, n_{next} .
3. A timer for clocking SSPs.
4. The maximum parity sequence number for block j , $P_{max}(j)$; this corresponds to the maximum number of parities requested for block j and is initially set to zero.

Each receiver must maintain the following:

1. The block sequence numbers of packets received in the last previously multicast sequence number window.
2. A parity request (PR) suppression timer for block j . $T_{pr}(j)$,
3. A PR loss detection timer for block j , $T_{loss}(j)$.
4. The number of parities required to recover the missing packets belonging to block j , $R_{rr}(j)$.

Initialization: The source periodically multicasts an SSP which declares a GPO and a timeout associated with that object. A MIP is sent immediately following the SSP to set the start and end of the sequence window and to initialize all previously uninitialized state to zero.

Data transfer: Data packets are multicast in blocks of size B_{size} to the group. Unique sequence numbers commencing with n_{start} are used to identify the blocks. They are incremented at the transmission of each new block. In addition, each packet has a unique sequence number in the range $(0, B_{size} - 1)$. The receivers use these sequence numbers to identify packets unsuccessfully received for each block. Last, these packets require no GMTS support. Therefore, they are transmitted as non-GMTS packets. They traverse the fast path illustrated in Figure 2.

Requests for repairs: When a receiver believes that it has lost one or more packets within block j , it sets a parity request counter $R_{pr}(j)$ to the number of missing packets and sets the PR suppression timer. If this timer goes off, then the receiver increments $R_{pr}(j)$ by the number of additional packets missing from block j and invokes the *rcvr_update* method with $v = R_{pr}(j)$, $f_s(x, y) = x$, and $f_v(a, v_1, v_2) = \max(v_1, a * v_2)$ at the first upstream GMTS router. In addition, the PR loss detection timer is set. If either all of the missing packets for block j or a PR for block j containing value greater than $R_{pr}(j)$ arrive while the suppression timer is set, the timer is turned off. In the case of receipt of a repair request, the RR loss detection timer is still set. Last, if the PR loss detection timer goes off, then the process is repeated.

Parity transmission: When the sender receives a PR packet for block j requesting v parities where $v > P_{max}(j)$, it constructs $v - P_{max}(j)$ new parity packets for block j and invokes the forward method $v - P_{max}(j)$ times to individually send them towards the receivers. The sender updates $P_{max}(j)$ to be v and invokes the forward method with $n = j$, $v = 0$, $g_s = x - 1$, $g_v = interface.vectors(n)$, and the appropriate parity packet as parameters.

We conclude this section with several observations. First we can configure a traditional non-FEC based reliable multicast protocol by setting $B_{size} = 1$ with the understanding that a parity packet is equivalent to the original data packet. Second, PGM [9] permits a combination of FEC and a selective repair mode. This can be implemented in GMTS by creating two objects, one to deal with FEC and a second to provide the selective repair mode. Last, note that the block size, B_{size} is an application-level parameter, not a GMTS-level parameter.

5.2 Repair Servers

GMTS can also be used to create and maintain a set of repair servers so that a receiver can request a repair from the closest repair server (measured in number of hops). We describe an approach that emulates the design presented in [4]. This requires the creation of a separate multicast group to which all receivers, the sender, and all repair servers belong. An object can be instantiated at all of the GMTS routers whose state will include the identity of the interface which lies on the path from that router to the closest repair server along with *distance labels* attached to all interfaces residing in the tree. The *mcast_update* method can be used to set up this state and the *forward* method can be used by a receiver to request a repair from the closest repair server.

Any member of the repair server multicast group can act as the source of SSPs. We will assume that it is the sender of data packets. The sender periodically multicasts SSPs. Each SSP declares an object and grants access rights to all members of the group to use the *forward/mcast_update* methods. Immediately after transmitting the SSP, a MIP is sent which initializes the state variable to infinity and the sequence number window.

Setting up shortest paths: Any repair server can initiate the shortest path computation

between receivers and repair servers. This is done by invoking *mcast_update* over the link connecting the repair server to the multicast tree connecting the group using the following predicate and functions f_s, f_v :

- $pred(x, y) = (x < y)$
- $f_s(x, y) = x + 1$
- $f_v(a, v_1, v_2) = a \times v_2$

As a consequence, *mcast_update* will eventually be invoked on all the routers on the tree and at the multicast group member end hosts. Whenever a MIP originating from a repair server containing a *mcast_update* invocation arrives at one of the members of the group and that group member has not already invoked *mcast_update* on its neighboring GMTS router, it does so. Otherwise, the member ignores it.

Last, whenever a receiver wishes to obtain a repair from the closest repair server, it invokes the forward method on the closest router on the multicast tree. The data parameter within the MIP contains the information required by the repair server to process the repair request.

We conclude this section with several observations. First, fewer packets will be required to perform the shortest path computation if the following predicate is used: $pred(x, y) = (x + 1 < y)$

Second, the part of this protocol that establishes the shortest paths to repair servers should be executed periodically, perhaps in response to SSPs, so as to adapt to routing changes. This may require the introduction of round numbers in order to ensure correct operation. These can be interpreted as sequence numbers in the maintenance of the GMTS object. Last, metrics other than the minimum number of hops can be used - provided that it is possible to access other router information such as loss summary information, bandwidth information, etc.

5.3 Anycasting

Anycasting can be supported in a similar way as repair services. This would be performed in a similar manner as repair services. One possible catch. The client requesting service may not already be part of the anycast tree connecting the servers. We need to be sure that the forward packet containing the request is routed to that tree and trapped by one of the routers. Once this takes place, the server that it arrives to can serve the request.

5.4 Congestion control

Several congestion control protocols require knowledge of the worst case receiver according to some metric such as loss rate, p , or the loss rate normalized by the round trip time, $RTT\sqrt{p}$. This is easily accommodated in the GMTS framework by setting up a suppression object and having the receivers periodically invoke the *rcvr_update* method up the tree. The largest loss rate metric would propagate up the tree and the routers would establish a path between source and the receiver that generated this value.

Note that the sender can communicate with all receivers by invoking the forward method with value equal to zero.

As these estimates would have to be reestablished periodically, sequence numbers could be used to distinguish between the rounds.

6 Summary and Future Work

GMTS provides an architecture for the development of fixed simple services in routers that assist control message scalability problems in end-to-end multicast protocols. The GPO object has been presented which provides a number of core services for multicast protocols. We have shown how GMTS can be used to complement an end-to-end reliable multicast protocol.

We wish to investigate further services which would be useful to a variety of multicast protocols and applications. Of particular interest is the area of congestion control and RTCP like reporting functions. Last, we are in the process of adding GMTS to a Linux-based router for the purpose of evaluating its performance and the benefits provided to multicast applications.

References

- [1] S. Floyd, V. Jacobson, S. McCanne, C. Lin, L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *IEEE/ACM Trans on Networking*, **5**, pp. 784-803, Dec. 1997.
- [2] S. Kasera, J. Kurose, D. Towsley. "A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast", *Proc. INFOCOM'98*, 1998.
- [3] L.H. Lehman, S.J. Garland, D. L. Tennenhouse. "Active Reliable Multicast", *Proc. INFOCOM'98*, 1998.
- [4] B.N. Levine, J.J. Garcia-Luna-Aceves. "Improving Internet Multicast with Routing Labels", *Proc. ICNP-97*, pp. 241-250, Oct. 1997.
- [5] J. Lin, S. Paul. "RMTP: A reliable multicast transport protocol", *Proc. of IEEE INFOCOM'95*, 1995.
- [6] J. Nonnenmacher, E. Biersack, D. Towsley. "Parity-Based Loss Recovery for Reliable Multicast Transmission", *IEEE/ACM Trans. on Networking*, Aug. 1998.
- [7] C. Papadoulos, G. Parulkar. "An Error Control Scheme for Large-Scale Multicast Applications", *Proc. INFOCOM'98*.
- [8] D. Rubenstein, S. Kasera, D. Towsley, J. Kurose. "Improving Reliable Multicast Using Active Parity Encoding Services (APES)", *Proc. INFOCOM'99*.
- [9] T. Speakman, D. Farinacci, S. Lin, A. Tweedly. "PGM Reliable Transport Protocol", IETF <draft-speakman-pgm-spec-02.txt>, August, 1998.
- [10] D. Towsley, J. Kurose, S. Pingali. "A comparison of sender-initiated and receiver-initiated reliable multicast protocols" *IEEE JSAC*, April 1997.