RMT Working Group
INTERNET-DRAFT
draft-ietf-rmt-pi-norm-06
Expires: September 2003

B. Adamson/NRL
C. Bormann/Tellique
M. Handley/ACIRI
J. Macker/NRL
March 2003

NACK-Oriented Reliable Multicast Protocol (NORM)

Status of this Memo

Abstract

This document describes the messages and procedures of the Negative-acknowledgement (NACK) Oriented Reliable Multicast (NORM) protocol. This protocol is designed to provide end-to-end reliable transport of bulk data objects or streams over generic IP multicast routing and forwarding services. NORM uses a selective, negative acknowledgement mechanism for transport reliability and offers additional protocol mechanisms to conduct reliable multicast sessions with limited "a priori" coordination among senders and receivers. A congestion control scheme is specified to allow the NORM protocol fairly share available network bandwidth with other transport protocols such as Transmission Control Protocol (TCP). It is capable of operating with both reciprocal multicast routing among senders and receivers and with asymmetric connectivity (possibly a unicast return path) from the senders to receivers. The protocol offers a number of features to allow different types of applications or possibly other higher level transport protocols to utilize its service in different ways. The protocol leverages the use of FEC-based repair and other IETF reliable multicast transport (RMT) building blocks in its design.

1.0 Introduction and Applicability

The Negative-acknowledgement (NACK) Oriented Reliable Multicast (NORM) protocol is designed to provide reliable transport of data from one or more sender(s) to a group of receivers over an IP multicast network. The primary design goals of NORM are to provide efficient, scalable, and robust bulk data (e.g. computer files, transmission of persistent data) transfer across possibly heterogeneous IP networks and topologies. The NORM protocol design provides support for distributed multicast session participation with minimal coordination among senders and receivers. NORM allows senders and receivers to dynamically join and leave multicast sessions at will with minimal overhead for control information and timing synchronization among participants. To accommodate this capability, NORM protocol message headers contain some common information allowing receivers to easily synchronize to senders throughout the lifetime of a reliable multicast session. NORM is designed to be self-adapting to a wide range of dynamic network conditions with little or no pre-configuration. The protocol is purposely designed to be tolerant of inaccurate timing estimations or lossy conditions that may occur many networks including mobile and wireless. The protocol is also designed to exhibit convergence and efficient operation even in situations of heavy packet loss and large queueing or transmission delays.

This document is a product of the IETF RMT WG and follows the guidelines provided in RFC 3269 [1]. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [2].

1.1 NORM Delivery Service Model

A NORM protocol instance (NormSession) is defined within the context of participants communicating connectionless (e.g. Internet Protocol (IP) or User Datagram Protocol (UDP)) packets over a network using pre-determined addresses and host port numbers. Generally, the participants exchange packets using an IP multicast group address, but unicast transport may also be established or applied as an adjunct to multicast delivery. In the case of multicast, the participating NormNodes will communicate using a common IP multicast group address and port number that has been chosen via means outside the context of the given NormSession. Other IETF data format and protocol standards exist that may be applied to describe and convey the required "a priori" information for a specific NormSession (e.g. Session Description Protocol (SDP) [5], Session Announcement Protocol (SAP) [6], etc).

The NORM protocol design is principally driven with the assumption of a single sender transmitting bulk data content to a group of receivers. However, the protocol MAY operate with multiple senders within the context of a single NormSession. In initial implementations of this protocol, it is anticipated that multiple senders will transmit independently of one another and receivers will maintain state as necessary for each independent sender. However, in future versions of NORM, it is possible that some aspects of protocol operation (e.g. round-trip time collection) may provide for alternate modes allowing more efficient performance for applications requiring multiple senders.

NORM provides for three types of bulk data content objects (NormObjects) to be reliably transported. These types include:

1) static computer memory data content (NORM_OBJECT_DATA type),

2) computer storage files (NORM_OBJECT_FILE type), and

3) non-finite streams of continuous data content (NORM_OBJECT_STREAM type).

The distinction between NORM_OBJECT_DATA and NORM_OBJECT_FILE is simply to provide a "hint" to receivers in NormSessions serving multiple types of content as to what type of storage should be allocated for received content (i.e. memory or file storage). Other than that distinction, the two are identical, providing for reliable transport of finite (but potentially very large) units of content. These static data and file services are anticipated to be useful for multicast-based cache applications with the ability to reliably provide transmission of large quantities of static data. Other types of static data/file delivery services might make use of these transport object types, too. The use of the NORM_OBJECT_STREAM type is at the application's discretion and could be used to carry static data or file content also. The NORM reliable stream service opens up additional possibilities such as serialized reliable messaging or other unbounded, perhaps dynamically produced content. The NORM_OBJECT_STREAM provides for reliable transport analogous to that of the Transmission Control Protocol (TCP), although NORM receivers will be able to begin receiving stream content at any point in time. The applicability of this feature will depend upon the application.

The NORM protocol also allows for a small amount of "out-of-band" data (sent as `NORM_INFO` messages) to be attached to the data content objects transmitted by the sender. This readily-available "out-of-band" data allows multicast receivers to quickly and efficiently determine the nature of the corresponding data, file, or stream bulk content being transmitted. This allows application-level control of the receiver node's participation in the current transport activity. This also allows the protocol to be flexible with minimal pre-coordination among senders and receivers. The `NORM_INFO` content is designed to be atomic in that its size MUST fit into the payload portion of a single NORM message.

NORM does _not_ provide for global or application-level identification of data content within in its message headers. Note the `NORM_INFO` out-of-band data mechanism could be leveraged by the application for this purpose if desired, or identification could alternatively be embedded within the data content. NORM does identify transmitted content (NormObjects) with transport identifiers that are applicable only while the sender is transmitting and/or repairing the given object. These transport data content identifiers (NormTransportIds) are assigned in a monotonically increasing fashion by each NORM sender during the course of a NormSession. Each sender maintains its NormTransportId assignments independently so that individual NormObjects may be uniquely identified during transport with the concatenation of the sender session-unique identifier (NormNodeId) and the assigned NormTransportId. The NormTransportIds are assigned from a large, but fixed, numeric space in increasing order and may be reassigned during long-lived sessions. The NORM protocol provides mechanisms so that the sender application may terminate transmission of data content and inform the group of this in an efficient manner. Other similar protocol control mechanisms (e.g. session termination, receiver synchronization, etc) are specified so that reliable multicast application variants may construct different, complete bulk transfer communication models to meet their goals.

In summary, the NORM protocol's goal is to provide reliable transport of different types of data content (including potentially mixed types). The senders enqueue and transmit bulk content in the form of static data or files and/or non-finite, ongoing stream types. The sender will provide for repair transmission of this content in response to NACK messages received from the receiver group. Mechanisms for "out-of-band" information and other transport control mechanisms are specified for use by applications to form complete reliable multicast solutions for different purposes.

1.2 NORM Scalability

Group communication scalability requirements lead to adaptation of negative acknowledgement (NACK) based protocol schemes when feedback for reliability is required [7]. NORM is a protocol centered around the use of selective NACKs to request repairs of missing data. NORM provides for the use of packet-level forward error correction (FEC) techniques for efficient multicast repair and optional proactive transmission robustness[8]. FEC-based repair can be used to greatly reduce the quantity of reliable multicast repair requests and repair transmissions[9]. The principal factor in NORM scalability is the volume of feedback traffic generated by the receiver set to facilitate reliability and congestion control. NORM uses probabilistic suppression of redundant feedback based on exponentially distributed random backoff timers. The performance of this type of suppression relative to other techniques is described in [10]. NORM dynamically measures the group's roundtrip timing status to set its suppression and other protocol timers. This allows NORM to scale well while maintaining reliable data delivery transport with low latency relative to the network topology over which it is operating. Feedback messages can be either multicast to the group at large or sent via unicast routing to the sender. In the case of unicast feedback, the sender "advertises" the feedback state to the group to facilitate feedback suppression. In typical Internet environments, it is expected that the NORM protocol will readily scale to group sizes on the order of tens of thousands of receivers. A study of the quantity of feedback for this type of protocol is described in [11]. NORM is able to operate with a smaller amount of feedback than a single TCP connection, even with relatively large numbers of receivers. Thus, depending upon the network topology, it is possible that NORM may scale to larger group sizes. With respect to computer resource usage, the NORM protocol does _not require that state be kept on all receivers in the group. NORM senders maintain state only for receivers providing explicit congestion control feedback. NORM receivers must maintain state for for each active sender. This may constrain the number of simultaneous senders in some uses of NORM.

1.3 NORM Environmental Requirements and Considerations

All of the environmental requirements and considerations that apply to the RMT FEC Building Block and the the RMT TCP-Friendly Multicast Congestion Control (TFMCC) Building Block also apply to NORM. When the RMT GRA Building Block is used with NORM, its environmental requirements and considerations SHALL also apply.

The NORM protocol SHALL be capable of operating in an end-to-end fashion with no assistance from intermediate systems beyond basic IP multicast group management, routing, and forwarding services. The NORM protocol SHOULD be compatible with techniques like Generic Router Assist (GRA) [12] for performance benefits when applicable. While the techniques utilized in NORM are principally applicable to "flat" end-to-end IP multicast multicast topologies, they could also be applied in the sub-levels of hierarchical (e.g. tree-based) multicast distribution if so desired. NORM can make use of reciprocal (among senders and receivers) multicast communication under the Any-Source Multicast (ASM) model defined in RFC 1112 [13], but SHALL also be capable of scalable operation in asymmetric topologies such as Source Specific Multicast (SSM) [14] where there may only be unicast routing service from the receivers to the sender(s).

NORM is compatible with IPv4 and IPv6. Additionally, NORM may be used with networks employing Network Address Translation (NAT) providing the NAT device supports IP multicast and/or can cache UDP traffic source port numbers for remapping feedback traffic from receivers to the sender(s).

2.0 NORM Architecture Definition

A *NormSession* is comprised of participants (*NormNodes*) acting as senders and/or receivers. NORM senders transmit data content in the form of *NormObjects* to the session destination address and the NORM receivers attempt to reliably receive the transmitted content using negative acknowledgments to request repair. Each NormNode within a NormSession is assumed to have a preselected unique 32-bit identifier (*NormNodeId*). NormNodes MUST have uniquely assigned identifiers within a single NormSession to distinquish between possible multiple senders and to distinguish feedback information from different receivers. There are two reserved NormNodeId values. A value of `0x00000000` is considered an invalid NormNodeId value and a value of `0xffffffff` is a "wildcard" NormNodeId. Whilte, the protocol does not preclude multiple sender nodes concurrently transmitting within the context of a single NORM session (i.e. many- to-many operation), any type of interactive coordination among NORM senders is assumed to be controlled by the application or higher protocol layer. There are some optional mechanisms specified in this document which can be leveraged for such application layer coordination.

As previusly noted, NORM allows for reliable transmission of three different basic types of data content. The first type is `NORM_OBJECT_DATA` which is used for static, persistent blocks of data content maintained in the sender's application memory storage. The second type is `NORM_OBJECT_FILE` which corresponds to data stored in the sender's non-volatile file system. The `NORM_OBJECT_DATA` and `NORM_OBJECT_FILE` types both represent "NormObjects" of finite but potentially very large size. The third type of data content is `NORM_OBJECT_STREAM` which corresponds to an ongoing transmission of undefined length. This is analogous to the reliable streaming content provide by TCP for unicast data transport. The format of the stream content is application-defined and may be byte or message oriented. The NORM protocol provides for "flushing" of the stream to expedite delivery or possible enforce application message boundaries. NORM protocol implementations may offer either (or both) in-order delivery of the stream data to the receive application or out-of-order (more immediate) delivery of received segments of the stream to the receiver application. In either case, NORM sender and receiver implementations provide buffering to facilitate repair of the stream as it is transported. All NormObjects are logically segmented into FEC coding blocks and segments for transmission by the sender.

NormObjects and associated transmission segments are temporarily yet uniquely identified within the NormSession context using the given sender's NormNodeId and a temporarily unique NormObjectTransportId. These data content identifiers are sender-assigned and applicable and valid only during a NormObject's actual _transport_ (i.e. for as long as the sender is transmitting and providing repair of the indicated NormObject). For a long-lived session, the NormObjectTransportId field can wrap and previously-used identifiers may be re-used. Note that globally unique identification of transported data content is not provided by NORM and, if required, must be managed by the NORM application. Individual NormObject segments are further identified with FEC coding block and symbol (segment) indentifiers. This is discussed in detail later in this document.

2.1 NORM Protocol Operation Overview

A NORM sender primarily generates messages of type `NORM_DATA` that carry the NormObject data content segments and related FEC parity-based repair segments for the bulk data/file or stream objects being transferred. By default, FEC segments are sent only in response to receiver repair requests (NACKs) and thus normally impose no additional transmission overhead. However, the NORM implementation MAY be optionally configured to proactively transmit some amount of FEC segments along with the data content to potentially enhance performance (e.g., improved delay) at the cost of additional overhead with initial data transmission. This configuration may be sensible for certain network conditions and can allow for robust, asymmetric multicast (e.g., unidirectional routing, satellite, cable) [19] with reduced receiver feedback, or, in some cases, no feedback.

A sender message of type `NORM_INFO` is also defined and is used to carry any optional "out-of-band" context information for a given transport object. A single `NORM_INFO` message can be associated with a NormObject. Because of its atomic nature, missing `NORM_INFO` messages can be NACKed and repaired with a slightly lower delay process than NORM's general FEC-encoded data content. `NORM_INFO` may serve special purposes for some bulk transfer, reliable multicast applications where receivers join the group mid-stream and need to ascertain contextual information on the current content being transmitted. The NACK process for `NORM_INFO` will be described later.

The sender also generates messages of type `NORM_CMD` to assist in certain protocol operations such as congestion control, end-of-transmission flushing, round trip time estimation, receiver synchronization, and optional positive acknowledgement requests or application defined commands. The transmission of `NORM_CMD` messages from the sender is accomplished by one of three different processes. These are: single, best effort unreliable transmission of the command; repeated redundant transmissions of the command; and positively-acknowledged commands. The transmission technique used for a given command depends upon the function of the command. Several core commands are defined for basic protocol operation. Additionally, implementations MAY wish to consider providing the OPTIONAL application-defined commands that can take advantage of the transmission methodologies available for commands. This allows for application-level session management mechanisms which can make use of information available to the underlying NORM protocol engine (e.g. round-trip timing, transmission rate, etc).

NORM receivers generate messages of type `NORM_NACK` or `NORM_ACK` in response to transmissions of data and commands from a sender. The `NORM_NACK` messages are generated to request repair of detected data transmission losses. Receivers generally detect losses by tracking the sequence of transmission from a sender. Sequencing information is embedded in the transmitted data packets and end-of-transmission commands from the sender. `NORM_ACK` messages are generated in response to certain commands transmitted by the sender. In the general (and most scalable) protocol mode, `NORM_ACK` messages are sent only in response to congestion control commands from the sender. The feedback volume of these congestion control NORM_ACK messages is controlled using the same timer-based probabilistic suppression techniques as for NORM_NACK messages to avoid feedback implosion. In order to meet potential application requirements for positive acknowledgement from receivers, other NORM_ACK messages are defined and available for use. All sender and receiver transmissions are subject to rate control governed by a peak transmission rate set for each participant by the application. This can be used to limit the quantity of multicast data transmitted by the group. When NORM's congestion control algorithm is enabled the rate for senders is automatically adjusted. In some networks, it may be desirable to establish minimum and maximum bounds for the rate adjustment depending upon the application even when dynamic congestion control is enabled. However, in the case of the general Internet, congestion control policy SHALL be observed which is compatible with coexistent TCP flows.

2.2 NORM Protocol Building Blocks

The operation of the NORM protocol is based upon the concepts presented in the Nack-Oriented Reliable Multicast (NORM) Building Block document[15]. This includes the basic NORM architecture and the data transmission, repair, and feedback strategies discussed in that document. NORM also makes use of Forward Error Correction encoding techiques for repair messaging and optional transmission robustness as described in [16]. NORM uses the FEC Payload ID as specified by the FEC Building Block Document[17]. Additionally, for congestion control, the NORM protocol specifies a mechanism based on the TCP-Friendly Multicast Congestion Control (TFMCC) Building Block described in [18].

2.3 NORM Design Tradeoffs

While the various features of NORM are designed to provide some measure of general purpose utility, it is important to emphasize the understanding that "no one size fits all" in the reliable multicast transport arena. There are numerous engineering tradeoffs involved in reliable multicast transport design and this requires an increased awareness of application and network architecture considerations. Performance requirements affecting design can include: group size, heterogeneity (e.g., capacity and/or delay), asymmetric delivery, data ordering, delivery delay, group dynamics, mobility, congestion control, and transport across low capacity connections. NORM contains various parameters to accommodate many of these differing requirements. The NORM protocol and its mechanisms MAY be applied in multicast applications outside of bulk data transfer, but there is an assumed model of bulk transfer transport service that drives the trade-offs that determine the scalability and performance described in this document.

The ability of NORM to provide reliable data delivery is also governed by any buffer constraints of the sender and receiver applications. NORM protocol implementations SHOULD be designed to operate with the greatest efficiency and robustness possible within application-defined buffer constraints. Buffer requirements for reliability, as always, are a function of the delay-bandwidth product of the network topology. NORM performs best with additional buffering as compared to typical point-to-point transport NORM feedback suppression based upon randomly-delayed transmissions from the receiver set. There are definitive tradeoffs between buffer utilization, group size scalability, and efficiency of performance. Large buffer sizes allow the NORM protocol to perform most efficiently in large delay-bandwidth topologies and allow for longer feedback suppression backoff timeouts. This yields improved group size scalability. NORM can operate with reduced buffering but at a cost of decreased efficiency (lower relative goodput) and reduced group size scalability.

3.0 Conformance Statement

This Protocol Instantiation document, in conjunction with the following Building Block documents identified in [15], [16], [17], and [18] completely specifies a working reliable multicast transport protocol that conforms to the requirements described in RFC 2357 [3].

4.0 NORM Message Formats

As mentioned in Section 2.1, there are two primary classes of NORM messages: sender messages and receiver messages. NORM_CMD, NORM_INFO, and NORM_DATA message types are generated by senders of data content, and NORM_NACK and NORM_ACK messages generated by receivers within a NormSession. An auxillary message type of NORM_REPORT is also provided for experimental purposes. This section described the message formats used by the NORM protocol. These messages and their fields are referenced in the detailed functional description of the NORM protocol given in Section 5.0. Individual NORM messages are designed to be compatible with the MTU limitations of encapsulating Internet protocols including IPv4, IPv6, and UDP. The current NORM protocol specification assumes UDP encapsulation and leverages the transport features of UDP. The NORM messages are independent of network addresses and can be used in IPv4 and IPv6 networks.

4.1 NORM Common Message Header

There are some common message fields contained in all NORM message types. All NORM protocol messages begin with a common header with information fields as follows:

NORM Common Message Header Format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |     type      |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

The "version" field is a 8-bit value indicating the protocol version number. Currently, NORM implementations SHOULD ignore received messages with a different protocol version number than their own. This number is intended to indicate and distinguish upgrades of the protocol which may be non-interoperable.

The message "type" field is a 8-bit value indicating the NORM protocol message type. These types are defined as follows:

Message	Value
NORM_INFO	1
NORM_DATA	2
NORM_CMD	3
NORM_NACK	4
NORM_ACK	5
NORM_REPORT	6

The "sequence" field is a 16-bit value that is set by the message originator as a monotonically increasing number incremented with each NORM message transmitted to the session's destination address. The "sequence" field SHOULD not be incremented for messages not sent to the session group address (e.g. unicast NACKs or unicast ACKs). This value can be monitored by receiving nodes to detect packet losses in the transmission from a sender. Note that this value is NOT used in the NORM protocol to detect missing reliable data content and does NOT identify the application data or FEC payload that may be attached. This sequence number is intended for use in estimating raw packet loss for congestion control purposes. The size of this field is intended to be sufficient to allow detection of a reasonable range of packet loss within the delay-bandwidth product of expected network connections.

The "source_id" field is a 32-bit value identifying the node that sent the message. A participant's NORM node identifier (NormNodeId) can be set according to the application needs but unique identifiers must be assigned within a single NormSession. In some cases, use of the host IP address or a hash of it can suffice, but alternative methodologies for assignment and potential collision resolution of node identifiers within a multicast session need to be considered. For example, the "source identifier" mechanism defined in the Real-Time Protocol (RTP) specification [20] may be applicable to use for NORM node identifiers. At this point in time, the protocol makes no assumptions about how these unique identifiers are actually assigned.

4.2 NORM Sender Messages

NORM sender messages include the `NORM_DATA` type, the `NORM_INFO` type, and the `NORM_CMD` type. `NORM_DATA` and `NORM_INFO` messages contain application data content while `NORM_CMD` messages for various protocol functions.

4.2.1 NORM_DATA Message

The `NORM_DATA` message is expected to be the predominant type transmitted by NORM senders. These messages are used to encapsulate segmented data content for objects of type NORM_OBJECT_DATA, NORM_OBJECT_FILE, and NORM_OBJECT_STREAM. `NORM_DATA` messages may contain original or FEC-encoded application data content. The payload size of these messages SHALL be limited to a maximum of the sender's *NormSegmentSize*. A sender's NormSegmentSize is assumed to be constant for the duration of a given sender's term of participation in the session. The NormSegmentSize is expected to be configurable by the sender application prior to session participation as needed for network topology maximum transmission unit (MTU) considerations. For IPv6, MTU discovery may be leveraged at session startup

NORM_DATA Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 2    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     flags     |     grtt      |     gsize     | fec_id = 129  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        fec_instance_id        |        fec_num_parity         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       fec_max_block_len       |         segment_size          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |      object_size (msb)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       object_size (lsb)                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       fec_block_number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          fec_block_len        |         fec_symbol_id         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          payload_len*         |         offset (msb)*         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          offset (lsb)*                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          payload_data*                        |
```

**Note: The "payload_len" and "offset" fields for NORM_DATA messages containing parity information are actually values computed from FEC encoding of the "payload_len" and "offset" fields of the data segments of the applicable coding block. So, for parity segments, these do _not_ represent actual values. Parity packets can be identified as packets where "fec_symbol_id >= fec_block_len".*

The "version", "type", "sequence", and "source_id" fields form the NORM Common Message Header asdescribed in Section 4.1.

The "flags" field contains a number of different binary flags providing information and hints regarding how the receiver should handle the identified object. Defined flags in this field include:

Flag	Value	Purpose
NORM_FLAG_REPAIR	0x01	Indicates message is a repair transmission
NORM_FLAG_EXPLICIT	0x02	Indicates a repair segment intended which meets a specific receiver erasure, as compared to parity segments provided by the sender for general purpose (with respect to an FEC coding block) erasure filling.
NORM_FLAG_INFO	0x04	Indicates availability of `NORM_INFO` for object
NORM_FLAG_UNRELIABLE	0x08	Indicates that repair transmissions for the specified object will be unavailable. (One-shot, best effort transmission)
NORM_FLAG_FILE	0x10	Indicates object is "file-based" data (hint to use disk storage for reception)
NORM_FLAG_STREAM	0x20	Indicates object is of type NORM_OBJECT_STREAM.

The `NORM_FLAG_REPAIR` flag is set when the associated message is a repair transmission. This information can be used by receivers to help observe a join policy where it is desired that newly joining receivers only begin participating in the NACK process upon receipt of new (non-repair) data content. The `NORM_FLAG_EXPLICIT` flag is used to mark repair messages sent when the data sender has exhausted its ability to provide "fresh" (previously untransmitted) parity segments as repair. This flag may be used by intermediate systems implementing Generic Router Assist (GRA) functionality to control subcasting of repair content to different legs of a reliable multicast topology with disparate repair needs. The `NORM_FLAG_INFO` flag is set only when there optional `NORM_INFO` content is available for the associated object. Thus, receivers will NACK for retransmission of `NORM_INFO` only when it is available. The `NORM_FLAG_UNRELIABLE` flag is set when the sender wishes to transmit an object with only "best effort" delivery and will not supply repair transmissions for the object. The `NORM_FLAG_FILE` flag can be set as a "hint" from the sender that the associated object should be stored in non-volatile storage. The `NORM_FLAG_STREAM` flag is set when the identified object is of type `NORM_OBJECT_STREAM`.

The "grtt" field contains a non-linear quantized representation of the sender's current estimate of group round-trip time (GRTT) (This is also referred to as R_max in the TFMCC Building Block [18]). This value is used to control timing of the NACK repair process and other aspects of protocol operation as described in this document. The algorithm for encoding and decoding this field is described in the RMT NORM Building Block document[15].

The "gsize" field contains a representation of the sender's current estimate of group size. This value is used to control feedback suppression mechanisms within the protocol for more optimized performance for different group sizes. The 8-bit "gsize" field consists of 4 bits of mantissa in the 4 most significant bits and 4 bits of base 10 exponent (order of magnitude) information in the 4 least significant bits. For example, to represent an approximate group size of 100 (or 1e02), the value of the upper 4 bits is 0x01 (to represent the mantissa of 1) and the lower 4 bits value would be 0x02 for an 8-bit representation of "0x12". As another example, a group size of 9000 (9e03) would be represented by the value 0x93. The group size does not need to be represented with a high degree of precision to appropriately scale backoff timers, etc.

The "fec_id" field corresponds to the FEC Encoding Identifier described in the FEC Building Block document [17]. Note the packet format illustrated above assumes "Small Block Systematic Codes" that corresponds to an FEC Encoding Identifier equal to 129. The other "fec_" fields may be interpreted or sized differently to supportother FEC Encoding Identifier types in the future.

The "fec_instance_id" corresponds to the "FEC Instance ID" of the FEC Object Transmission Informatiom given in the FEC

Building Block document[17]. The "fec_instance_id" SHALL be a value corresponding to the particular type of Small Block Systematic Code being used (e.g. Reed-Solomon GF(2^8), Reed-Solomon GF(2^16), etc). The standardized assignment of FEC Instance ID values is described in [17].

The "fec_num_parity" corresponds to the "maximum number of of encoding symbols that can be generated for any source block" as described in for FEC Object Transmission Information for Small Block Systematic Codes in the FEC Building Block document [17]. For example, Reed-Solomon codes may be arbitrarily shortened to create different code variations for a given block length. In the case of Reed-Solomon (GF(2^8) and GF(2^16) codes, this value indicates the maximum number of parity segments available from the sender for the coding blocks. This field MAY be interpreted differently for other systematic codes as they are defined.

The "fec_max_block_len" indicates the current maximum number of user data segments per FEC coding block to be used by the sender during the session. This allows receivers to allocate appropriate buffer space for buffering blocks transmitted by the sender.

The "segment_size" field indicates the sender's current setting for maximum message payload content (in bytes). This allows receivers to allocate appropriate buffering resources and to determine other information in order to properly process received data messaging.

The "object_transport_id" field is a monotonically and incrementally increasing value assigned by a sender to the object being transmitted. Transmissions and repair requests related to that object use the same "object_transport_id" value. For sessions of very long or indefinite duration, the "object_transport_id" field may be repeated, but it is presumed that the 16-bit field size provides an adequate enough sequence space to prevent temporary object confusion amongst receivers and sources (i.e. receivers SHOULD re-synchronize with a server when receiving object sequence identifiers sufficiently out-of-range with the current state kept for a given source). During the course of its transmission within a NORM session, an object is uniquely identified by the concatenation of the sender "node_id" and the given "object_transport_id". Note that `NORM_INFO` messages associated with the identified object carry the same "object_transport_id" value.

The 48-bit "object_size" field indicates the total size of the object (in bytes) for the static object types of `NORM_OBJECT_FILE` and `NORM_OBJECT_DATA`. This information is used by receivers to determine storage requirements and/or allocate storage for the received object. Receivers with insufficient storage capability may wish to forego reliable reception (i.e. not NACK for) of the indicated object. In the case of objects of type `NORM_OBJECT_STREAM`, the "object_size" field is used to by the sender to indicate the size of its stream buffer to the receiver group. In turn, the receivers SHOULD use this information to allocate a stream buffer for reception of corresponding size.

The "fec_block_number", "fec_block_len", and "fec_symbol_id" fields correspond to the "Source Block Number", "Source Block Length, and "Encoding Symbol ID" fields of the FEC Payload ID format given by the FEC Building Block document[17]. The "fec_block_number" identifies the coding block's relative position with a NormObject. Note that, for NormObjects of type NORM_OBJECT_STREAM, the "fec_block_number" may wrap for very long lived sessions. The "fec_block_len" indicates the number of user data segments in the identified coding block. Given the "fec_block_len" (Source block length) information of how many symbols of application data is contained in the block, the receiver can determine whether the attached segment is data or parity content and treat it appropriately. The "fec_symbol_id" identifies which specific symbol (segment) within the coding block the attached payload conveys. Depending upon the value of the "fec_symbol_id" and the associated "fec_block_len" and "fec_num_parity" parameters for the block, the symbol (segment) referenced may be a user data or an FEC parity segment. For systematic codes, symbols numbered 0 through `(fec_block_len-1)` contain application data while segments numbered `(fec_block_len)` through `(fec_block_len+fec_num_parity-1)` contain the parity symbols calculated for the block.

The concatenation of *object_tranport_id::fec_block_number::fec_symbol_id* can be viewed as a unique transport data unit (TPDU) identifier for the attached segment with respect to the NORM sender.

The "payload_len" and "offset" fields are used to identify the relative position and quantity of the content of the message payload. For senders employing systematic FEC encoding, these fields will correspond to actual length and offset values for `NORM_DATA` messages which contain original data content. For `NORM_DATA` messages containing calculated parity content, these fields will actually contain values computed by FEC encoding of the "payload_len" and "offset" values of the `NORM_DATA` segments of the corresponding FEC coding block. Thus, the "payload_len" and "offset" values of missing data content can be determined when decoding an FEC coding block.

The "payload_data" field contains original data or computed parity content of the identified segment. The maximum length of this field corresponds to the sender's NormSegmentSize. The length of this field for messages containing parity content will always be of the length NormSegmentSize. When encoding data segments of varying sizes, the FEC encoder SHALL assume zero value padding for data segments with length less than the NormSegmentSize. The receiver will use the "payload_len" information to properly retrieve received data content and deliver it to the application.

4.2.2 NORM_INFO Message

The `NORM_INFO` message is used to convey OPTIONAL, application-defined, "out-of-band" context information for transmitted NormObjects. An example `NORM_INFO` use for bulk file transfer is to place MIME type information for the associated file, data, or stream object into the `NORM_INFO` payload. Receivers may use the `NORM_INFO` content to make a decision as whether to participate in reliable reception of the associated object. Each NormObject can have an independent unit of `NORM_INFO` associated with it. `NORM_DATA` messages contain a flag to indicate the availability of `NORM_INFO` for a given NormObject. NORM receivers may NACK for retransmission of `NORM_INFO` when they have not received it for a given NormObject. The size of the `NORM_INFO` content is limited to that of a single NormSegmentSize for the given sender. This atomic nature allows the `NORM_INFO` to be rapidly and efficiently repaired within the NORM reliable transmission process.

When `NORM_INFO` content is available for a NormObject, the NORM_FLAG_INFO flag SHALL be set in `NORM_DATA` messages for the corresponding "object_transport_id" and the `NORM_INFO` message shall be transmitted as the first message for the NormObject.

NORM_INFO Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 1    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     flags     |     grtt      |     gsize     | fec_id = 129  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       fec_encoding_name       |        fec_num_parity         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       fec_max_block_len       |         segment_size          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |      object_size (msb)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        object_size (lsb)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         payload_data                          |
```

The "version", "type", "sequence", and "source_id" fields form the NORM Common Message Header asdescribed in Section 4.1.

The "flags", "grtt", "gsize", "fec_id", "fec_encoding_name", "fec_num_parity", "fec_max_block_len", "segment_size", "object_transport_id", and "object_size" fields carry the same information and serve the same purpose as with `NORM_DATA` messages. These values allow the receiver to prepare appropriate buffering, etc, for further transmissions from the sender when `NORM_INFO` is the first message received.

The `NORM_INFO` "payload_data" field contains sender application-defined content which can be used by receiver applications for various purposes as described above.

4.2.3 NORM_CMD Message

`NORM_CMD` messages are transmitted by senders to perform a number of different protocol functions. This includes functions such as round-trip timing collection, congestion control functions, synchronization of sender/receiver repair "windows", and notification of sender status. A core set of `NORM_CMD` messages is enumerated. Additionally, a range of command types remain available for potential application-specific use. Some `NORM_CMD` types may have dynamic content attached. Any attached content will be limited to maximum length of the sender NormSegmentSize to retain the atomic nature of commands. All `NORM_CMD` message begins with a common set of fields, after the usual NORM message common header. The standard `NORM_CMD` fields are:

NORM_CMD Standard Fields

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 3    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     grtt      |     gsize     |    flavor     |      ...      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The "version", "type", "sequence", and "source_id" fields form the NORM Common Message Header as described in Section 4.1.

The "grtt" and "gsize" fields provide the same information and serve the same purpose as with `NORM_DATA` and `NORM_INFO` messages. The "flavor" field indicates the type of command to follow. The remainder of the `NORM_CMD` message is dependent upon the command type ("flavor"). The command flavors include:

Command	Flavor Value	Purpose
`NORM_CMD`(FLUSH)	1	Used to indicate sender temporary or permanent end-of-transmission. (Assists in robustly initiating outstanding repair requests from receivers).
`NORM_CMD`(SQUELCH)	2	Used to advertise sender's current repair window in response to out-of-range NACKs from receivers.
`NORM_CMD`(ACK_REQ)	3	Used to request positive acknowledgement from a list of receivers.
`NORM_CMD`(REPAIR_ADV)	4	USed to advertise sender's aggregated repair state for suppression of unicast receiver feedback.
`NORM_CMD`(CC)	5	Used for GRTT measurement and explicitly collection of congestion control feedback.
`NORM_CMD`(APPLICATION)	6	Used for application-defined purposes which may need to temporarily preempt data transmission.

NORM_CMD(FLUSH) Message

The `NORM_CMD`(FLUSH) command is sent when the sender reaches the end of all data content and pending repairs it has queued for transmission. This command is repeated once per 2*GRTT to excite the receiver set for any outstanding repair requests up to and including the transmission point indicated within the `NORM_CMD`(FLUSH) message. The number of repeats is equal to `NORM_ROBUST_FACTOR`. The greater the `NORM_ROBUST_FACTOR`, the greater the probability that all applicable receivers will be excited for repair requests (NACKs) _and_ that the corresponding NACKs are delivered to the sender. If a `NORM_NACK` message interrupts its flush process, the sender will re-initiate the flush process when any resulting repair transmissions are completed. Note that receivers also employ a timeout mechanism to self-initiate NACKing when no messages are received from a sender. This inactivity timeout is related to `2*GRTT*NORM_ROBUST_FACTOR` and will be discussed more later. With a sufficient `NORM_ROBUST_FACTOR` value, data content is delivered with a high assurance of reliability. The penalty of a large `NORM_ROBUST_FACTOR` value is potentially excess sender `NORM_CMD`(FLUSH) transmissions and a longer timeout for receivers to self-initiate the terminal NACK process.

For finite-size transport objects such NORM_OBJECT_DATA and NORM_OBJECT_FILE, the flush process (if there are no further pending transmissions) will occur at the end of these objects and thus any FEC repair information is available for repairs in response to repair requests elicited by the flush command. However, for NORM_OBJECT_STREAM, the flush may occur at any time, including in the middle of an FEC coding block if systematic FEC codes are emplyed. In this case, the sender will not yet be able to provide FEC parity content as repair for the concurrent coding block and will be limited to explicitly repairing stream data content for that block. Applications that anticipate frequent flushing of stream content SHOULD be judicious in the selection of the FEC coding block size (i.e. do not use a very large coding block size if frequent flushing occurs). For example, a reliable multicast application transmitting an on-going series of intermittent, relatively small messaging content will need to trade-off using the NORM_OBJECT_DATA paradigm versus the NORM_OBJECT_STREAM paradigm with an appropriate FEC coding block size. This is analogous to application trade-offs for other transport protocols such as the selection of different TCP modes of operation such as "no delay", etc.

NORM_CMD(FLUSH) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 3    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     grtt      |     gsize     |  flavor = 1   |     flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |    fec_block_number (msb)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    fec_block_number (lsb)     |         fec_symbol_id         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In addition to the NORM common message header and standard NORM_CMD fields, the NORM_CMD(FLUSH) message contains fields to identify the current status and logical transmit position of the sender.

The "flags" field contains sender status information. A single NORM_CMD(FLUSH) flag is currently defined:

NORM_FLUSH_FLAG_EOT = 0x01

When the NORM_FLUSH_FLAG_EOT flag is set, this indicates the sender is preparing to terminate transmission and will no longer provide response to repair requests. This allows the receiver set to gracefully reach closure of operation with this sender and free any resources that are no longer needed.

The "object_transport_id", "fec_block_number", and "fec_symbol_id" fields indicate the sender's current logical "transmit position". These fields are interpreted in the same manner as the fields of the same names in the NORM_DATA message type. Upon receipt of the the NORM_CMD(FLUSH), receivers are expected to check their completion state _through_ (including) this transmission position. If receivers have outstanding repair needs in this range, they SHALL initiate the NORM NACK Repair Process as described in Section 5.3. If receivers have no outstanding repair needs, no response is generated.

For NORM_OBJECT_STREAM objects, receivers MUST request "explicit-only" repair of the identified "fec_block_number" if the given "fec_symbol_id" is less than the sender's "fec_max_block_len - 1". This condition indicates the sender has not yet completed encoding the corresponding FEC block and parity content is not yet available. An "explicit-only" repair request consists of NACK content for the applicable "fec_block_number" which does not include any requests for parity-based repair. This allows NORM sender applications to "flush" an ongoing stream of transmission when needed, even if in the middle of an FEC block. Once the sender resumes stream transmission and passes the end of the pending coding block, subsequent NACKs from receivers SHALL request parity-based repair as normal. Note that the use of a systematic FEC code is assumed here. Normal receiver NACK inititation and construction is discussed in detail in Section 5.3.

NORM_CMD(SQUELCH) Message

The `NORM_CMD`(SQUELCH) command is transmitted in response to invalid `NORM_NACK` content received by the sender. Invalid `NORM_NACK` content consists of repair requests for NormObjects for which the sender is unable or unwilling to provide repair. This includes repair requests for outdated objects, aborted objects, or those objects which the sender previously transmitted marked with the NORM_FLAG_UNRELIABLE flag. This command indicates to receivers what content is available for repair, thus serving as a description of the sender's current "repair window". Receivers SHALL not generate repair requests for content identified as invalid by a `NORM_CMD`(SQUELCH).

The `NORM_CMD`(SQUELCH) command is sent once per 2*GRTT at the most. The `NORM_CMD`(SQUELCH) advertises the current "repair window" of the sender by identifying the earliest (lowest) transmission point for which it will provide repair, along with an encoded list of objects from that point forward that are no longer valid for repair. This mechanism allows the sender application to cancel or abort transmission and/or repair of previously enqueued objects. The list also contains the identifiers for any objects within the repair window which were sent with the `NORM_FLAG_UNRELIABLE` flag set. In normal conditions, it is expected the `NORM_CMD`(SQUELCH) will be needed infrequently, and generally only to provide a reference repair window for receivers who have fallen "out-of-sync" with the sender due to extremely poor network conditions.

The starting point of the invalid NormObject list begins with the lowest invalid NormTransportId greater than the current "repair window" start from the invalid NACK(s) that prompted the generation of the squelch. The length of the list is limited by the sender's NormSegmentSize. This allows the receivers to learn the status of the sender's applicable object repair window with minimal transmission of `NORM_CMD`(SQUELCH) commands. The format of the `NORM_CMD`(SQUELCH) message is:

NORM_CMD(SQUELCH) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |            sequence           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            source_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      grtt     |     gsize     |  flavor = 2   |   reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |    fec_block_number (msb)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     fec_block_number (lsb)    |         fec_symbol_id         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        invalid_object_list  ...               |
```

In addition to the NORM common message header and standard `NORM_CMD` fields, the `NORM_CMD`(SQUELCH) message contains fields to identify the earliest logical transmit position of the sender's current repair window and an "invalid object list" beginning with the index of the logically earliest invalid repair request from the offending NACK message which initiated the squelch transmission.

The "object_transport_id", "fec_block_number", and "fec_symbol_id" fields are concatenated to indicate the beginning of the sender's current repair window (i.e. the logically earliest point in its transmission history for which the sender can provide repair). This serves as an advertisement of a "synchronization point" for receivers to request repair. Note, that while the "fec_symbol_id" is provided here, the sender's repair window will generally be incremented on an FEC coding block basis and the "fec_symbol_id" will be zero.

The "invalid_object_list" is a list of 16-bit NormTransportIds that, although they are within the sender's current repair window, are no longer available for repair from the sender. For example, a sender application may dequeue an out-of-date object even though it is still within the repair window. The total size of the "invalid_object_list" content is implied by the packets payload length and is limited to a maximum of the NormSegmentSize of the sender. Thus, for very large repair windows, it is possible that a single `NORM_CMD`(SQUELCH) message may not be capable of listing the entire set of invalid objects in the repair window. In this case, the sender SHALL ensure that the list begins with a NormObjectId that is greater than or equal to the lowest ordinal invalid NormObjectId from the NACK message(s) that prompted the `NORM_CMD`(SQUELCH) generation. The NormObjectIds in the "invalid_object_list" must be greater than the "object_transport_id" marking the beginning of the sender's repair window. This insures convergence of the squelch process, even if multiple invalid NACK/ squelch iterations are required. This explicit description of invalid content within the sender's current window allows the sender application (most notably for discrete "object" based transport) to arbitrarily invalidate (i.e. dequeue) portions of enqueued content (e.g. certain objects) for which it no longer wishes to provide reliable transport.

NORM_CMD(REPAIR_ADV) Message

The `NORM_CMD`(REPAIR_ADV) message is used by the sender to "advertise" its aggregated repair state from accumulated `NORM_NACK` messages accumulated during a repair cycle and/or congestion control feedback received. This message is sent only when the sender has received `NORM_NACK` and/or `NORM_ACK`(RTT) (when congestion control is enabled) messages via unicast transmission instead of multicast. By "echoing" this information to the receiver set, suppression of feedback can be achieved even when receivers are unicasting that feedback instead of multicasting it among the group[11].

NORM_CMD(REPAIR_ADV) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      grtt     |     gsize     |  flavor = 4   |     flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   cc_flags    |    cc_rtt     |            cc_rate            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    repair_adv_content ...                     |
```

The "grtt", "gsize" and "flavor" fields serve the same purpose as in other NORM_CMD messages.

The "flags" field provide information on the NORM_CMD(REPAIR_ADV) content. There is currently one NORM_CMD(REPAIR_ADV) flag defined:

NORM_REPAIR_ADV_FLAG_LIMIT = 0x01

This flag is set by the sender when it is unable to fits its full current repair state into a single NormSegmentSize. If this flag is set, receivers should limit their NACKing to generating NACKs only up through the maximum ordinal transmission position *(objectId::fecBlockId::fecSymbolId)* included in the "repair_adv_content".

When congestion control operation is enabled, the "cc_flags", "cc_rtt", and "cc_rate" fields contain values for the receiver with the lowest calculated congestion control rate from which feedback was received since the last NORM_CMD(REPAIR_ADV) transmission. These fields are used by receivers to suppress rounds of congestion control feedback. The definition of these fields is given in the description of the NORM_CMD(CC) message below.

The "repair_adv_content" is in exactly the same form as the "nack_content" of NORM_NACK messages and can be processed by receivers for suppression purposes in the same manner with the exception of the condition when the NORM_REPAIR_ADV_FLAG_LIMIT is set.

NORM_CMD(CC) Message

The NORM_CMD(CC) messages contains fields to enable sender->receiver group greatest round-trip time (GRTT) measurement and provide congestion control information to the group. The NORM_CMD(CC) message is usually transmitted as part of NORM congestion control operation. If NORM is operated in a private network with congestion control operation disabled, the NORM_CMD(CC) message is then used to facilitate GRTT measurement by the sender.

NORM_CMD(CC) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      grtt     |     gsize     | flavor = 5    |     flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         send_time_sec                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         send_time_usec                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           send_rate           | cc_sequence   |   reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         cc_node_list ...                      |
```

The NORM common message header and standard NORM_CMD fields serve their usual purposes.

The "flags" field is used to indicate NORM_CMD(CC) options. Currently a single NORM_CMD(CC) flag is defined:

```
NORM_CC_FLAG_ENABLE = 0x01
```

When set, this indicates the sender has enabled congestion control feedback collection, and receivers should respond observing the procedures describe in Section 5.5.2, "NORM Congestion Control Operation". When this flag is cleared (i.e. congestion control feedback collection is disabled), this indicates the sender is not observing congestion control operation and the NORM_CMD(CC) message is being used only to provide a reference timestamp for GRTT measurement via receiver NORM_NACK feedback.

The "send_time" field is a timestamp indicating the time that the NORM_CMD(CC) message was transmitted. This consists of a 64-bit field containing 32-bits with the time in seconds ("sent_time_sec") and 32-bits with the time in microseconds ("send_time_usec") since some reference time the source maintains (usually 00:00:00, 1 January 1970). The byte ordering of the fields is "Big Endian" network order. Receivers use this timestamp adjusted by the amount of delay from when they received the NORM_CMD(CC) message to when they respond for the "grtt_response" portion of NORM_ACK and NORM_NACK messages generated. This allows the sender to evaluate the round-trip time to different receivers for congestion control and other (e.g. GRTT determination) purposes.

The "send_rate" field indicates the sender's current transmission rate in bytes per second. The 16-bit "send_rate" field consists of 12 bits of mantissa in the most significant byte and 4 bits of base 10 exponent (order of magnitude) information in the least significant byte. The 12-bit mantissa portion of the field is scaled such that a floating point value of 0.0 corresponds to 0 and a floating point value of 10.0 corresponds to 4096. Thus:

```
value = (int) (mantissa * 4096.0 / 10.0 + 0.5)
```

For example, to represent a transmission rate of 256kbps (3.2e+04 bytes per second), the lower 4 bits of the 16-bit field contain a value of 0x04 to represent the exponent while the upper 12 bits contain a value of 0x51f as determined from the equation given above:

```
value = (int)((3.2 * 4096.0 / 10.0) + 0.5) = 1311 = 0x51f
```

To decode the "send_rate" field, the following equation can be used:

```
sendRate = <upper12bits> * 10.0 / 4096.0 * power(10.0, <lower4bits>)
```

Note the maximum transmission rate representable by this scheme is approximately 9.99e+15 bytes per second.

The "cc_sequence" field is a sequence number applied by the sender to congestion control command messages. The greatest received "cc_sequence" value is recorded by receivers and fed back to the sender in any `NORM_ACK` or `NORM_NACK` messages generated by the receivers for that sender.

The "reserved" field is for potential future use and should be set to zero in this version of the NORM protocol.

The "cc_node_list" consists of a list of NormNodeIds and their associated congestion control status. This includes the current limiting receiver (CLR) node, any potential limiting receiver (PLR) nodes which have been identified, and some number of receivers for which congestion control status is being provided, most notably including the receivers' current RTT measurement. The length of the "cc_node_list" provides for at least the CLR and one other receiver, but may be configurable for more timely feedback to the group. The list length can be inferred from the length of the `NORM_CMD`(CC) message.

Each item in the "cc_node_list" is in the following format:

Congestion Control Node List Item Fields

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           cc_node_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    cc_flags   |     cc_rtt    |            cc_rate            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The "cc_node_id" is the NormNodeId of the receiver which the item represents.

The "cc_flags" field contains flags indicating the congestion control status of the indicated receiver. The following flags are defined:

Flag	Value	Purpose
NORM_CC_FLAG_CLR	0x01	Receiver is the current limiting receiver (CLR)
NORM_CC_FLAG_PLR	0x02	Receiver is a potential limiting receiver (PLR)
NORM_CC_FLAG_RTT	0x04	Receiver has measured RTT with respect to sender
NORM_CC_FLAG_START	0x08	Sender/receiver is in "slow start" phase of congestion control operation (i.e. The receiver has not yet detected any packet loss and the "cc_rate" field is a function of the receiver's measured receive rate).
NORM_CC_FLAG_LEAVE	0x10	Receiver is imminently leaving the session and its feedback should not be considered in congestion control operation.

The "cc_rtt" contains a quantized representation of the receiver's individual sender<->receiver RTT as measured by the sender. This field is valid only if the `NORM_FLAG_RTT` flag is set in the "cc_flags" field. This one byte field is a quantized representation of the RTT using the algorithm described in the NORM Building Block document [15].

The "cc_rate" field contains a representation of the receiver's current calculated (during steady-state congestion control operation) or twice its measured (during the "slow start" phase) congestion control rate. This field is encoded and decoded using the same technique as described for the `NORM_CMD`(CC) "send_rate" field.

NORM_CMD(ACK_REQ) Message

The `NORM_CMD`(ACK_REQ) message is used by the sender to request acknowledgement from a specified list of receivers. This message is used in providing a lightweight positive acknowledgement mechanism that is OPTIONAL for use by the reliable multicast application. The NORM protocol defines a specific acknowledgement mechanism to determine that watermark points in the reliable transmission have been achieved by specific receivers. Addtionally, a range of acknowledgement request types is provided for use at the application's discretion. Provision for application-defined, positively-acknowledged commands allows the application to automatically take advantage of transmission and round-trip timing information available to the NORM protocol. The details of the NORM positive acknowledgement process including transmission of the `NORM_CMD`(ACK_REQ) messages and the receiver response (`NORM_ACK`) are described in Section 5.5.3. The format of the `NORM_CMD`(ACK_REQ) message is:

NORM_CMD(ACK_REQ) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            source_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      grtt     |     gsize     |  flavor = 3   |   ack_type    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         ack_req_content                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    ack_req_content (cont'd)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        acking_node_list ...                   |
```

The NORM common message header and standard NORM_CMD fields serve their usual purposes.

The "ack_type" field indicates type of acknowledemgent being requested and thus implies rules for how the receiver will treat this request. The following "ack_type" values are defined and are also used in NORM_ACK messages described later:

ACK Type	Value	Purpose
NORM_ACK(WATERMARK)	1	Used to request acknowledgement of reliable reception of watermark transmission point.
NORM_ACK(CC)	2	Used to identify NORM_ACK messages sent for congestion control only.
NORM_ACK(RESERVED)	3-15	Reserved for possible future NORM protocol use.
NORM_ACK(APPLICATION)	16-255	Used at application’s discretion.

The "ack_req_content" field consists of 8 bytes which is interpreted differently for different "ack_type" values.

The "acking_node_list" field is a list of NormNodeIds. The listed NormNodes are expected to explicitly respond to the acknowledgement request according the rules for the type of acknowledgment requested and the NORM Positive Acknowledgment procedure described in Section 5.3.3.

The NORM_ACK(WATERMARK) type indicates the sender wishes to receive acknowledgement from receivers in the "acking_node_list" who have achieved completion of reception through a specific "watermark point" in terms of a logical transmission position. This "watermark point" is given in the "ack_req_content" field.

The format of the NORM_CMD(ACK_REQ(WATERMARK)) message is:

NORM_CMD(ACK_REQ(WATERMARK)) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 3    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     grtt      |     gsize     |  flavor = 3   | ack_flavor = 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |    fec_block_number (msb)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     fec_block_number (lsb)    |         fec_symbol_id         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      acking_node_list ...                     |
```

The NORM common message header and standard NORM_CMD fields serve their usual purposes. The "ack_flavor" is set to a value of one.

The "object_transport_id", "fec_block_number", and "fec_symbol_id" are used to identify the watermark point for which positive acknowledgement is requested. This watermark point is similar to the transmission position given in

NORM_CMD(FLUSH) messages. Furthermore, NORM receivers (whether or not they are included in the "acking_node_list") SHALL treat the ACK_REQ(WATERMARK) command as equivalent to a NORM_CMD(FLUSH) command and appropriately initiate NACK repair cycles in response to any detected missing data up through the indicated watermark point.

The "acking_node_list" field contains the NormNodeIds of the current NORM receivers which should positive acknowledge (NORM_ACK) this request. The packet payload length implies the length of the "acking_node_list" and its length is limited to the NormSegmentSize. The individual NormNodeId items are listed in network (Big Endian) order. If a receiver is included in the "acking_node_list" and it has no repair needs up through the watermark point, it SHALL schedule transmission of a NORM_ACK message as described in Section 5.5.3.

The NORM_ACK(CC) type is provided only for when receivers generate NORM_ACK messages in response to NORM_CMD(CC) messages for congestion control operation. There is no corresponding NORM_CMD(ACK_REQ(CC)) message.

The NORM_ACK(RESERVED) range of types is provided for possible future NORM protocol use.

The NORM_ACK(APPLICATION) range of types is provided so that NORM applications may implement application-defined, positively-acknowledged commands which are able to leverage internal transmission and round-trip timing information available to the NORM protocol implementation. The interpretation of the "ack_req_content" is application-defined in this case.

NORM_CMD(APPLICATION) Message

This command allows the NORM application to robustly transmit application-defined commands. The command message preempts any ongoing data transmission and is repeated NORM_ROBUST_FACTOR times at a rate of once per 2*GRTT. This rate of repetition allows the application to collect any response (if that is the application's purpose for the command) before it is repeated. Possible responses might include initiation of data transmission , NORM_CMD(APPLICATION) messages, or even application-defined, positively-acknowledge commands from other NormSession participants. The transmission of these commands will preempt data transmission when they are scheduled and may be multiplexed with ongoing data transmission. This type of robustly transmitted command allows NORM applications to define a complete set of session control mechanisms with less state than the transfer of FEC encoded reliable content requires while taking advantage of NORM transmission and round-trip timing information.

NORM_CMD(APPLICATION) Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            source_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     grtt      |     gsize     |  flavor = 6   |   reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   application defined content ...             |
```

The NORM common message header and NORM_CMD fields are interpreted as previously described.

The "application-defined content" contains information in a format at the discretion of the application. The size of this payload is limited a maximum of the sender's NormSegmentSize setting.

4.3 Receiver Messages

The NORM message types generated by pariticipating receivers consist of NORM_NACK and NORM_ACK message types. NORM_NACK messages are sent to request repair of missing data content from sender transmission and NORM_ACK messages are generated in response to certain sender commands including NORM_CMD(CC) and NORM_CMD(ACK_REQ).

4.3.1 NORM_NACK Message

The principal purpose of NORM_NACK messages is for receivers to request repair of sender content via selective, negative acknowledgement upon detection of incomplete data. NORM_NACK messages will be transmitted according to the rules of NORM_NACK generation and suppression described in Section 5.3. The content of these messages is in a format that can potentially be used by compatible intermediate systems [12] to provide assistance in promoting protocol scalability and efficiency when available. NORM_NACK messages also contain additional fields to provide feedback to the sender(s) for purposes of round-trip timing collection and congestion control.

The payload of NORM_NACK messages contains one or more repair requests for different objects or portions of those objects. The NORM_NACK message format is as follows:

NORM_NACK Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |  type = 3     |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            source_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            server_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        grtt_response_sec                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        grtt_response_usec                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    cc_flags   |     cc_rtt    |            cc_loss            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            cc_rate            |  cc_sequence  |    reserved   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          nack_content ...                     |
```

The NORM common message header fields serve their usual purposes.

The "server_id" field identifies the NORM sender to which the NORM_NACK message is destined.

The "grtt_response" fields contain an adjusted version of the timestamp from the most recently received NORM_CMD(CC) message for the indicated NORM sender. The format of the "grtt_response" is the same as the "send_time" field of the

NORM_CMD(CC). The "grtt_response" value is _relative_ to the "send_time" the source provided with a corresponding NORM_CMD(CC) command. The receiver adjusts the source's NORM_CMD(CC) "send_time" timestamp by adding the time differential from when the receiver received the NORM_CMD(CC) to when the NORM_NACK is transmitted to calculate the value in the "grtt_response" field. This is the "receive_to_response_differential" value used in the following formula:

```
"grtt_response" = NORM_CMD(CC) "send_time" + receive_to_response_differential
```

The receiver SHALL set the "grtt_response" to a ZERO value, to indicate that the it has not yet received a NORM_CMD(CC) message from the indicated sender and that the sender should ignore the "grtt_response" in this message.

The "cc_flags" field contains bits representing the receiver's state with respect to congestion control operation. The possible values for the "cc_flags" field are those specified for the NORM_CMD(CC) message node list item flags.

The "cc_rtt" field SHALL be set to a default maximum value and the NORM_CC_FLAG_RTT flag SHALL be cleared when the receiver has not yet received RTT measurement information. When the receiver has received RTT measurement information, it shall set the "cc_rtt" value accordingly and set the NORM_CC_FLAG_RTT flag in the "cc_flags" field.

The "cc_loss" field is the receiver's current packet loss fraction estimate for the indicated source. The loss fraction is a value from 0.0 to 1.0 corresponding to a range of zero to 100 percent packet loss. The 16-bit "cc_loss" value is calculated by the following formula:

```
"cc_loss" = decimal_loss_fraction * 65535.0
```

The "cc_rate" field represents the receivers current local congestion control rate. During "slow start", when the receiver has detected no loss, this value is set to twice the actual rate it has measured from the corresponding sender and the NORM_CC_FLAG_START is set in the "cc_flags' field. Otherwise, the receiver calculates a congestion control rate based on its loss measurement and RTT measurement information (even if default) for the "cc_rate" field.

The "cc_sequence" field contains the current greatest "cc_sequence" number of received NORM_CMD(CC) messages from the corresponding sender. This information can assist the sender in congestion control operation by providing an indicator of how current ("fresh") the receiver's round-trip measurement reference time is and whether the receiver has been successfully receiving recent congestion control probes. For example, if it is apparent the receiver has not been receiving recent congestion control probes (and thus possibly other messages from the sender), the sender may choose to take congestion avoidance measures.

The "reserved" field is for potential future NORM use and SHALL be set to ZERO for this version of the protocol.

The "nack_content" of the NORM_NACK message specifies the repair needs of the receiver with respect to the NORM sender indicated by the "server_id" field. The receiver constructs repair requests based on the NORM_DATA and/or NORM_INFO segments it requires from the sender in order to complete reliable reception. A single repair request consists of a list of items, ranges, and/or FEC coding block erasure counts for needed NORM_DATA and/or NORM_INFO content. Multiple repair requests may be concatenated within the "nack_content" field of a NORM_NACK message. Note that a single repair request can possibly include multiple "items", "ranges", or "erasure_counts". In turn, the "nack_content" field may contain multiple repair request. A single repair request has the following format:

NACK Repair Request Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     form      |     flags     |            length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |     fec_block_number (msb)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      fec_block_number (lsb)   | fec_symbol_id or erasure_count|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

The "form" field indicates currently whether the repair request content that follows is a list of NORM_NACK_ITEMS, NORM_NACK_RANGES, or NORM_NACK_ERASURES. Possible values for the "form" field include:

Form	Value
NORM_NACK_ITEMS	1
NORM_NACK_RANGES	2
NORM_NACK_ERASURES	3

When the repair request consists of individual NORM_NACK_ITEMS, each concatenation of *object_transport_id::fec_block_number::fec_symbol_id* identifies an individual repair need. When the repair request "form" is NORM_NACK_RANGES, the inclusive range of sender information needed by the receive is given in pairs of *object_transport_id::fec_block_number::fec_symbol_id*. When the repair request form is NORM_NACK_ERASURES, each *object_transport_id::fec_block_number::erasure_count* concatenation listed indicates the receiver's FEC erasure count for the identified object and FEC encoding block.

The "flags" field is currently used to indicate if the NACK content applies to NORM_DATA content, NORM_INFO content, or both. Thus, defined flags in this field include:

Flag	Value	Purpose
NORM_NACK_SEGMENT	0x01	Indicates the listed segment(s) are required as repair.
NORM_NACK_BLOCK	0x02	Indicates the entire listed block(s) are required as repair.
NORM_NACK_INFO	0x04	Indicates the object's NORM_INFO is required as repair.
NORM_NACK_OBJECT	0x08	Indicates the entire listed object(s) are required as repair.

When the NORM_FLAG_SEGMENT flag is set, the "object_transport_id", "fec_block_number" and "fec_symbol_id" fields are concatenated to determine which sets or ranges of individual NORM_DATA segments are needed to repair complete content at this receiver. When the NORM_FLAG_BLOCK flag is set, this indicates the receiver is completely missing the indicated coding block(s) and requires transmissions sufficient to repair the indicated block(s) in their entirety. In this case the "fec_symbol_id" repair request fields are ignored. When the NORM_NACK_INFO flag is set, this indicates the receiver is missing the NORM_INFO segment for the indicated "object_transport_id". Note the NORM_NACK_INFO may be set in combination with the NORM_NACK_BLOCK or NORM_NACK_SEGMENT flags, or may be set alone. When the NORM_NACK_OBJECT flag is set, this indicates the receiver is missing the entire NormTransportObject referenced by the "object_transport_id". This also implicitly requests any available NORM_INFO for the NormObject, if applicable. The "fec_block_number" and "fec_symbol_id" fields are ignored when the flag NORM_NACK_OBJECT is set.

The "length" field is given (in bytes) to indicate the length of the list of repair request items or ranges. Multiple lists of repair request items and/or ranges may be concatenated together within a single NORM_NACK message.

The "object_transport_id", "fec_block_number, and "fec_symbol_id" fields comprise repair request list items to be interpreted according to the repair request "form" and "flags" fields. As noted, when the "form" is NORM_NACK_RANGES, pairs of *object_transport_id::fec_block_number::erasure_count* define each repair request list item.

NORM_NACK Content Examples:

In Example 1, a list of individual NORM_NACK_ITEM repair requests is given. In Example 2, a list of NORM_NACK_RANGE requests _and_ a single NORM_NACK_ITEM request are concatenated to illustrate the possible content of a NORM_NACK message. Note that FEC coding block erasure counts are provided in each case. The erasure counts are not really necessary since the sender can easily determine the erasure count while processing the NACK content. However, the erasure count option may be useful for operation with Generic Router Assist (GRA).

Example 1:
NORM_NACK content for: Object 12, Coding Block 3, Segments 2,5,8

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 3    | flags = 0x01  |        length  = 8            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 12    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number(lsb) = 3   |    erasure_count  = 3         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 1    | flags = 0x01  |        length  = 24           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 12    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 3  |    fec_symbol_id  = 2         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 12    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 3  |    fec_symbol_id  = 5         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 12    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 3  |    fec_symbol_id  = 8         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example 2:
NORM_NACK content for: Object 18 Coding Block 6, Segments 5, 6, 7, 8, 9, 10; and Object 19 NORM_INFO and Coding Block 1, segment

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 3    | flags = 0x01  |        length  = 8            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 18    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 6  |    erasure_count  = 6         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 2    | flags = 0x01  |        length  = 16           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 18    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 6  |    fec_symbol_id  = 5         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 18    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 6  |    fec_symbol_id  = 10        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 3    | flags = 0x05  |        length  = 8            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 19    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 1  |    erasure_count  = 1         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 1    | flags = 0x05  |         length  = 8           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   object_transport_id = 19    |   fec_block_number (msb) = 0  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   fec_block_number (lsb) = 1  |    fec_symbol_id  = 3         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.3.2 NORM_ACK Message

The NORM_ACK message is primarily used as part of NORM congestion control operation and round-trip timing measurement. The generation of NORM_ACK messages for round-trip timing and congestion-control operation is described in Sections 5.5.1 and 5.5.2, respecctively. Some applications may benefit from some limited form of positive acknowledgement for certain functions. A simple, scalable positive acknowledgement scheme is defined in Section 5.5.3 which can be leveraged by protocol implementations when appropriate.

NORM_ACK Message Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |   type = 3    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           server_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_sec                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_usec                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   cc_flags    |    cc_rtt     |            cc_loss            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            cc_rate            |  cc_sequence  |   ack_type    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         ack_content ...                       |
```

The NORM common message header fields serve their usual purposes.

The "server_id", "grtt_response", "cc_flags", "cc_rtt", "cc_loss", "cc_rate", and "cc_sequence" fields serve the same purpose as the corresponding fields in `NORM_NACK` messages.

The "ack_type" field indicates the nature of the `NORM_ACK` message. This directly corresponds to the "ack_type" field of the `NORM_CMD`(ACK_REQ) message.

The "ack_content" format is a function of the "ack_type". The `NORM_ACK`(CC) message has no attached content. Only the `NORM_ACK` header applies. In the case of `NORM_ACK`(WATERMARK), a specific "ack_content" format is defined:

NORM_ACK(WATERMARK) Ack Content

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      object_transport_id      |    fec_block_number (msb)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    fec_block_number (lsb)     |         fec_symbol_id         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The "object_transport_id", "fec_block_number", and "fec_symbol_id" are used by the receiver to acknowledge a `NORM_CMD`(ACK_REQ(WATERMARK)) transmitted by the sender identified by the "server_id" field.

The "ack_content" of `NORM_ACK` messages for application-defined "ack_type" values is specific to the application but is limited in size to a maximum the NormSegmentSize of the sender referenced by the "server_id".

4.4 General Messages

4.4.1 NORM_REPORT

This is an optional message generated by NORM participants. This message could be used for periodic performance reports from receivers in experimental NORM implementations. The format of this message is currently undefined. Experimental NORM implementations may define NORM_REPORT formats as needed for test purposes.

5.0 Functionality Definition

This section describes the detailed interactions of senders and receivers participating in a NORM session. A simple synopsis of protocol operation is given in the following items.

1) The sender periodically transmits `NORM_CMD`(CC) messages as needed to initialize and collect roundtrip timing and congestion control feedback from the receiver set.

2) The sender transmits an ordinal set of NormObjects segmented in the form of `NORM_DATA` (and optional `NORM_INFO`) messages labeled with NormTransportIds and logically identified with FEC encoding block numbers and symbol identifiers.

3) As receivers detect missing content from the sender, they initiate repair requests with `NORM_NACK` messages. Note the receivers track the sender's most recent *object_transport_id::fec_block_number::fec_symbol_id* transmit position and NACK _only_ for content ordinally prior to that transmit position. The receivers use random backoff timeouts before generating `NORM_NACK` messages and wait an appropriate amount of time before repeating the `NORM_NACK` if their repair request is not satisified.

4) The sender aggregates repair requests from the receiver set and logically "rewinds" to send appropriate repair messages. The sender sends repairs for the earliest ordinal transmit position first and maintains this ordinal repair transmission sequence. Previously untransmitted FEC parity content for the applicable FEC coding block is used for repair transmissions to the greatest extent possible. If the sender exhausts its available FEC parity content on multiple repair cycles for the same coding block, it resorts to an explicit repair strategy (again using parity content) to complete repairs. (The use of explicit repair is expected to be an exception in general protocol operation, but the possibility does exist for extreme conditions). The sender immediately assumes transmission of new content once it has sent pending repair transmissions.

5) The sender transmits `NORM_CMD`(FLUSH) messages when it reaches the end of newly available transmit content. Receivers respond to the `NORM_CMD`(FLUSH) messages with `NORM_NACK` transmissions (following the same suppression backoff timeout strategy as for data) if they require further repair.

6) The sender transmission rate is subject to rate control limits determined by congestion control. Each sender in a NormSession maintains its own independent congestion control state. Receivers provide congestion control feedback in `NORM_NACK` and `NORM_ACK` messages. This feedback is controlled using suppression mechanism similar to that for `NORM_NACK` messages.

While the overall concept of the protocol is relatively simple, there are details to each of these aspects that need to be addressed for successful, robust, and scalable operation.

5.1 NORM Sender Initialization and Transmission

Upon startup, the NORM sender immediately begins sending NORM_CMD(CC) messages to collect GRTT and other information from the potential group. If congestion control operation is enabled the NORM_CC_FLAG_ENABLE MUST be set. Congestion control operation SHALL be observed at all times when operating in the general Internet. Even if congestion control operation is disabled at the sender, it may be desirable to set the the NORM_CC_FLAG_ENABLE to proactively collect feedback from the receivers to have input to GRTT measurement prior to NACK initiation.

In some cases, applications may wish for the sender to also proceed with data transmission immediately. In other cases, the sender may wish to defer data transmission until it has received some feedback or request from the receiver set indicating that receivers are indeed present. Note, in some applications (e.g. web push), this indication may come out-of-band with respect to the multicast session via other means. The periodic transmission of NORM_CMD(CC) messages may precede actual data transmission in order to have initial GRTT measurement.

The NORM protocol sender message headers contain all information necessary to configure receivers for subsequent reliable reception. This includes FEC coding parameters, the sender NormSegmentSize, and other information. Additionally, applications may leverage the use of NORM_INFO messages associated with the session data objects in the session to provide application-specific context information for the session and data being transmitted.

The NORM sender begins segmenting application-enqueued data into NORM_DATA segments and transmitting it to the group. The rate of transmission is controlled via the congestion control mechanisms described in this document or at a fixed rate if desired for closed network operations. The receivers participating in the multicast group provide feedback to the sender as needed. When the sender reaches the end of data it has enqueued for transmission or any pending repairs, it transmits a series of NORM_CMD(FLUSH) messages at a rate of one per 2*GRTT. Receivers may respond to these NORM_CMD(FLUSH) messages with additional repair requests. A protocol parameter "NORM_ROBUST_FACTOR" determines the number of flush messages sent. If receivers request repair, the repair is provided and flushing occurs again at the end of repair transmission.

5.2 NORM Receiver Initialization and Reception

The NORM protocol is designed such that receivers may join and leave the group at will. However, some applications may be constrained such that receivers need to be members of the group prior to start of data transmission. NORM applications may use different policies to constrain the impact of new receivers joining the group in the middle of a session. For example, a useful implementation policy is for new receivers joining the group to restrain requesting repair of transport objects in progress. The NORM sender implementation may wish to impose additional constraints to limit the ability of receivers to disrupt reliable multicast performance by joining, leaving, and rejoining the group often. Different receiver "join policies" may be appropriate for different applications and/or scenarios. A default policy of allowing receivers to request repair only for coding blocks with a NormTransportId and FEC coding block number greater than or equal to the first non-repair NORM_DATA or NORM_INFO message received upon joining the group is RECOMMENDED for general purpose operation.

5.3 NORM Receiver NACK Procedure

When the receiver detects it is missing data from a sender's NORM transmissions, it initiates its NACKing procedure. The NACKing procedure SHALL be initiated _only_ at NormObject boundaries, FEC coding block boundaries, or upon receipt of a NORM_CMD(FLUSH) or NORM_CMD(ACK_REQ(WATERMARK)) message.

The NACKing procedure begins with a random backoff timeout. The duration of the backoff timeout is chosen using the

"RandomBackoff" algorithm described in the NORM Building Block document [15] using (K*GRTTsender) for the "maxTime" parameter and the sender advertised group size (GSIZEsender) as the "groupSize" parameter. The backoff factor "K" MUST be greater than one to provide for feedback suppression. A value of K = 4 is RECOMMENDED for the Any Source Multicast (ASM) model while a value of K = 6 is RECOMMENDED for Single Source Multicast (SSM) operation. Thus:

```
T_backoff = RandomBackoff(K*GRTTsender, GSIZEsender)
```

During this backoff time, the receiver accumulates external pending repair state from NORM_NACK messages and NORM_CMD(REPAIR_ADV) messages received. At the end of the backoff time, the receiver SHALL generate a NORM_NACK message only if the following conditions are met:

1) The sender's current transmit position (in terms of *object_transport_id::fec_block_number::fec_symbol_id*) exceeds the earliest repair position of the receiver.

2) The repair state accumulated from NORM_NACK and NORM_CMD(REPAIR_ADV) messages do not equal or supersede the receiver's repair needs.

If these conditions are met, the receiver immediately generates a NORM_NACK message when the backoff timeout expires.

The content of the NORM_NACK message contains repair request content beginning with lowest ordinal repair position for the receiver up to the most recently heard ordinal transmission position for the sender. If the size of the NORM_NACK content exceeds the NormSegmentSize, the NACK content is limited to that point so that the receiver only generates a single NORM_NACK message per NACK cycle for a given sender.

For each partially-received FEC coding block requiring repair, the receiver SHALL, on its _first_ repair attempt for the block, request the parity portion of the FEC coding block beginning with the lowest ordinal _parity_ "fec_symbol_id" and request the number of symbols corresponding to its data segment erasure count for the block. On _subsequent_ repair cycles for the same coding block, the receiver SHALL request only those repair symbols from the first set it has not yet received up to the remaining erasure count for that applicable coding block. Note that the sender may have provided other additional parity segments for other receivers that could also be used to satisfy the local receiver's erasure-filling needs. In the case where the erasure count for a partially-received FEC coding block exceeds the maximum number of parity symbols available from the sender for the block (as indicated by the NORM_DATA "fec_num_parity" field), the receiver SHALL request all available parity segments and the ordinally highest missing data segments required to satisfy its erasure needs for the block. The goal of this strategy is for the overall receiver set to request a lowest common denominator set of repair symbols for a given FEC coding block. This allows the sender to construct the most efficient repair transmission segment set and enables effective NACK suppression among the receivers even with uncorrelated packet loss. This approach also requires no synchronization among the receiver set in their repair requests for the sender.

For FEC coding blocks or NormObjects missing in their entirety, the NORM receiver constructs repair requests with NORM_NACK_BLOCK or NORM_NACK_OBJECT flags set as appropriate. The request for retransmission of NORM_INFO is accomplished by setting the NORM_NACK_INFO flag in a corresponding repair request.

5.4 NORM Sender NACK Processing and Repair Transmission

The principle goal of the sender is to make forward progress in the transmission of data its application has enqueued. However, the sender must occasionally "rewind" to satisfy the repair needs of receivers who have NACKed. Aggregation of multiple NACKs is used to determine an optimal repair strategy when a NACK event occurs. Since receivers initiate the NACK process on coding block or object boundaries, there is some loose degree of synchronization of the repair process.

5.4.1 NORM Sender Repair State Aggregation

When a sender is in its normal state of transmitting new data and receives a NACK, it begins a procedure to accumulate NACK repair state from `NORM_NACK` messages before beginning repair transmissions. Note that this period of aggregating repair state does _not_ interfere with its ongoing transmission of new data.

The period of time during which the sender aggregates `NORM_NACK` messages is equal to K*GRTT where "K" is the same backoff scaling value used by the receivers and "GRTT" is the sender's current estimate of the group's greatest round-trip time. When this period ends, the sender "rewinds" by incorporating the accumulated repair state into its pending transmission state and begins transmitting repair messages, then continues with new transmissions of any enqueued data. Also, at this point in time, the sender begins a "holdoff" timeout of 1*GRTT during which time the sender constrains itself from initiating a new repair aggregation cycle, even if `NORM_NACK` messages arrive. If additional `NORM_NACK` messages are received during this hold-off period, the sender will immediately incorporate these "late messages" into its pending transmission state ONLY if the NACK content is ordinally greater than the sender's current transmission position. This "holdoff" time allows worst case time for the sender to propagate its current transmission sequence position to the group, thus avoiding redundant repair transmissions. After the holdoff timeout expires, a new NACK accumulation period can be begun (upon arrival of a NACK) in concert with the pending repair and new data transmission. The sender repeats the same process of incorporating accumulated repair state into its transmission plan during the the new aggregation period and subsequently "rewinding" to transmit the lowest ordinal repair data.

5.4.2 NORM Sender FEC Repair Transmission Strategy

The NORM sender should leverage transmission of FEC parity content for repair to the greatest extent possible. Recall that the receivers use a strategy to request a lowest common denominator of explicit repair (including parity content) in the formation of their `NORM_NACK` messages. Before falling back to explicitly satisfying all of the different receivers' repair needs, the sender can make use of the general erasure-filling capability of FEC-generated parity segments. The sender can determine the maximum erasure filling needs for individual FEC coding blocks from the `NORM_NACK` messages received during the repair aggregation period. Then, if the sender has a sufficient number (less than or equal to the maximum erasure count) of previously unsent parity segments available for the applicable coding blocks, the sender can transmit these in lieu of the specific packets the receiver set has requested. Only after exhausting its supply of "fresh" (unsent) parity segments for a given coding block should the sender resort to explicit transmission of the receiver set's repair needs. In general, if a sufficiently powerful FEC code is used, the need for explicit repair will be an exception, and the fulfillment of reliable multicast can be accomplished quite efficiently. However, the ability to resort to explicit repair allows the protocol to be reliable under even very extreme circumstances.

`NORM_DATA` messages sent as repair transmissions are flagged with the NORM_FLAG_REPAIR flag. This allows receivers to obey any policies that limit new receivers from joining the reliable transmission on repair transmissions.

To facilitate operation with Generic Router Assist (GRA) [12], the sender can additionally flag `NORM_DATA` transmissions sent as explicit repair with the NORM_FLAG_EXPLICIT flag. The GRA router needs to only subcast a sufficient count of non-explicit parity repairs to satisfy the sub-tree's erasure filling needs for a given FEC coding block. When the sender has resorted to explicit repair, the GRA router will subcast all of the explicit repair packets to those portions of the routing tree still requiring repair for a given coding block. (Note the GRA router will be required to conduct repair state accumulation for sub-routes in a manner similar to the sender's repair state accumulation in order to have sufficient information to perform the subcasting. Additionally, the GRA router can perform additional `NORM_NACK` suppression/aggregation as it conducts this repair state accumulation for NORM repair cycles).

5.4.3 NORM Sender NORM_CMD(SQUELCH) Generation

If the sender receives a `NORM_NACK` message for repair of data it is no longer supporting, the sender generates a `NORM_CMD`(SQUELCH) message to advertise its repair window and squelch any receivers from additional NACKing of invalid data. The transmission rate of `NORM_CMD`(SQUELCH) messages is limited to once per 2*GRTT. The "invalid_object_list" (if applicable) of the `NORM_CMD`(SQUELCH) message SHALL begin with the lowest "object_transport_id" from the invalid `NORM_NACK` messages received since the last `NORM_CMD`(SQUELCH) transmission. Lower ordinal invalid "object_transport_ids" should be included only while the `NORM_CMD`(SQUELCH) payload is less than the sender's NormSegmentSize parameter.

5.4.4 NORM Sender NORM_CMD(REPAIR_ADV) Generation

When a NORM sender receives `NORM_NACK` messages from receivers via unicast transmission, it uses `NORM_CMD`(REPAIR_ADV) messages to advertise its accumulated repair state to the receiver set since the receiver set is not directly sharing their repair needs via multicast communication. The `NORM_CMD`(REPAIR_ADV) message is multicast to the receiver set by the sender. The payload portion of this message has content in the same format as the `NORM_NACK` receiver message payload. Receivers are then able to perform feedback suppression in the same manner as with `NORM_NACK` messages directly received from other receivers. Note the sender does not merely retransmit NACK content it receives, but instead transmits a representation of its aggregated repair state. The transmission of `NORM_CMD`(REPAIR_ADV) messages are subject to the sender transmit rate limit and NormSegmentSize limitation. When the `NORM_CMD`(REPAIR_ADV) message is of maximum size, receivers SHALL consider the maximum ordinal transmission position value embedded in the message as the senders "current" transmission position and suppress requests for ordinally higher repair. For congestion control operation, the `NORM_CMD`(REPAIR_ADV) fields of "cc_flags", "cc_rtt", and "cc_rate" contain the "worst case" values received for each field since the last `NORM_CMD`(REPAIR_ADV) transmission. This means the minimum received "cc_rate" and the set of "cc_flag" values resulting in the most suppression (i.e. the `NORM_CC_FLAG_RTT` flag is unset if _any_ congestion control feedback was received with that flag unset since the last `NORM_CMD`(REPAIR_ADV) transmission).

5.5 Additional NORM Protocol Mechanisms

In addition to the principal function of data content transmission and repair, there are some other protocol mechanisms that help NORM to adapt to network conditions and play fairly with other coexistent protocols.

5.5.1 NORM Greatest Round-trip Time (GRTT) Collection

For NORM receivers to appropriately scale backoff timeouts and the senders to use proper corresponding timeouts, the participants must agree on a common timeout basis. Each NORM sender monitors the round-trip time of active receivers and determines the group greatest round-trip time (GRTT). The sender advertises this GRTT estimate in every message it transmits so that receivers have this value available for scaling their timers. To measure the current GRTT, the sender periodically sends `NORM_CMD`(CC) messages which contain a locally generated timestamp. Receivers are expected to record this timestamp along with the time the `NORM_CMD`(CC) message is received. Then, when the receivers generate feedback messages to the sender, an adjusted version of the sender timestamp is embedded in the feedback message (`NORM_NACK` or `NORM_ACK`). The adjustment adds the amount of time the receiver held the timestamp before generating its response. Upon receipt of this adjusted timestamp, the sender is able to calculate the round-trip time to that receiver.

The round-trip time for each receiver is fed into an algorithm that weights and smooths the values for a conservative estimate of the GRTT. The algorithm and methodology is described in the NORM Building Block document [11] in the section entitled "One-to-Many Sender GRTT Measurement". A conservative estimate helps feedback suppression at a small cost in overall protocol repair delay. The sender's current estimate of GRTT is advertised in the "grtt" field found in all NORM sender messages. The advertised GRTT is also limited to be at least as big as the nominal inter-packet transmission time given the sender's current transmission rate. The reason for this additional limit is to keep the receiver somewhat "event driven" by making sure the sender has had adequate time to generate any response to repair requests from receivers given transmit rate limitations due to congestion control or configuration.

When the NORM_CC_FLAG_ENABLE is set in `NORM_CMD`(CC) messages, the receivers respond to `NORM_CMD`(CC) messages as described in Section 5.5.2, "NORM Congestion Control Operation". The `NORM_CMD`(CC) messages are periodically generated by the sender as described for congestion control operation. This provides for active, but controlled, feedback from the group in the form of `NORM_ACK` messages and can provide GRTT feedback even if no `NORM_NACK` messages are being sent. If operating without congestion control in a closed network, the `NORM_CMD`(CC) messages may be sent periodically with the NORM_CC_FLAG_ENABLE flag cleared. In this case, receivers will only provide GRTT measurement feedback when `NORM_NACK` messages are generated as no `NORM_ACK` messages are generated in response to the `NORM_CMD`(CC). In this case, the `NORM_CMD`(CC) messages may be sent less frequently, as little as once per minute, to conserve network capacity. Note that the NORM_CC_FLAG_ENABLE can also be set to actively solicit RTT feedback from the receiver group per congestion control operation even though the sender may not be observing congestion control rate adjustment. NORM operation without congestion control should only be considered in closed networks.

5.5.2 NORM Congestion Control Operation

This section describes congestion control operation for the NORM protocol. The supporting NORM message formats and approach described here are an adaptation of the equation-based TCP-Friendly Multicast Congestion Control (TFMCC) approach described in [18] and [21]. With this TFMCC-based approach, the transmission rate of NORM senders is controlled in a rate-based manner as opposed to window-based congestion control algorithms as in TCP. However, it is possible that the NORM protocol message set may alternatively be used to support a window-based multicast congestion control scheme such as PGMCC [22]. The details of that alternative may be described separately or in a future revision of this document. In either case (rate-based TFMCC or window-based PGMCC), successful control of sender transmission depends upon collection of sender->receiver packet loss estimates and sender<->receiver RTT to identify the congestion control bottleneck path(s) within the multicast topology and adjust the sender rate accordingly. The receiver with loss and RTT estimates that correspond to the lowest result transmission rate is identified as the "current limiting receiver" (CLR).

The steady-state sender transmission rate, to be "friendly" with competing TCP flows is calculated as:

```
                                     S
Rsender = -------------------------------------------------------------
            tRTT * (sqrt((2/3)*p) + 12 * sqrt((3/8)*p) * p * (1 + 32*(p^2)))
```

where

S = Nominal transmitted packet size. (The "nominal" packet size is determined by the sender as an exponentially weighted moving average (EWMA) of transmitted packet sizes to account for variable message sizes).

tRTT = The RTT estimate of the current "current limiting receiver" (CLR).

p = The loss event fraction of the CLR.

To support congestion control feedback collection and operation, the NORM sender periodically transmits NORM_CMD(CC) command messages. The GRTT is determined from congestion control feedback included in NACKs and ACKs from the receiver set. The NORM_CMD(CC) messages are multiplexed with NORM data and repair transmissions and serve several purposes:

1) Stimulate explicit feedback from the general receiver set to collect congestion control information.

2) Communicate state to the receiver set on the sender's current congestion control status including details of the CLR.

3) Initiate rapid (immediate) feedback from the CLR in order to closely track the dynamics of congestion control for that "worst path" in the sender->receiver multicast topology.

The format of the NORM_CMD(CC) message is describe in Section 4.2.3 of this document. The NORM_CMD(CC) message contains information to allow for determination of sender<->receiver RTTs, to inform the group of the congestion control CLR, and to provide feedback of individual RTT information to receivers in the group. The NORM_CMD(CC) also provides for exciting feedback from a set of potential limiting receiver (PLR) nodes that may be determined administratively or possibly algorithmically based on congestion control feedback. The details of PLR selection are not discussed in this document.

5.5.2.1 NORM_CMD(CC) Transmission

The NORM_CMD(CC) message is tranmitted periodically by the sender along with its normal data transmission. Note that the repeated transmission of NORM_CMD(CC) messages may be initiated some time before transmission of user data content at session startup. This may be done to collect some estimation of the current state of the multicast topology with respect to group and individual RTT and congestion control state.

A NORM_CMD(CC) message is immediately transmitted at sender startup. The interval of subsequent NORM_CMD(CC) message transmission is determined as follows:

1) By default, the interval is set according to the current sender GRTT estimate. A startup GRTT of 0.5 seconds is recommended when no feedback has yet been received from the group.

2) If a CLR has been identified (based on previous receiver feedback), the interval is the sender<->receiver RTT for the CLR.

3) Additionally, if the interval of nominal data message transmission is greater than the GRTT or CLR RTT interval, the NORM_CMD(CC) interval is set to this greater value. This ensures that the transmission of this control message is not done to the exclusion of user data transmission.

The NORM_CMD(CC) "cc_sequence" field is incremented with each transmission of a NORM_CMD(CC) command. The greatest "cc_sequence" recently received by receivers is included in their feedback to the sender. This allows the sender to determine the "age" of feedback to assist in congestion avoidance.

The sender advertises its current transmission rate in the "send_rate" field of the NORM_CMD(CC) message. This rate information is used by receivers to bias the timing of explicit feedback and to initialize loss estimation during congestion control startup or restart.

The "cc_node_list" contains a list of entries identifying receivers and their current congestion control state (status "flags", "rtt" and "loss" estimates). The list may be empty if the sender has not yet received any feedback from the group. If the sender has received feedback, the list will minimally contain an entry identifying the CLR. A NORM_CC_FLAG_CLR flag value is provided for the "cc_flags" field to identify the CLR entry. It is recommended that the CLR entry be the first in the list for implementation efficiency. Additional entries in the list are used to provide sender-measured individual RTT estimates to receivers in the group. The number of additional entries in this list is dependent upon the percentage of control traffic the sender application is willing to send with respect to user data message transmissions. More entries in the list may allow the sender to be more responsive to congestion control dynamics. The length of the list may be dynamically determined according to the current transmission rate and scheduling of NORM_CMD(CC) messages. The maximum length of the list corresponds to the sender's "NormSegmentSize" parameter for the session. The inclusion of additional entries in the list based on receiver feedback are prioritized with following rules:

1) Receivers that have not yet been provided RTT feedback get first priority. Of these, those with the greatest loss fraction receive precedence for list inclusion.

2) Secondly, receivers that have previously been provided RTT are included with receivers yielding the lowest calculated congestion rate getting precedence.

There are also "cc_flag" values in addition to NORM_CC_FLAG_CLR that are used for other congestion control functions. The NORM_CC_FLAG_CLR flag value is used to mark additional receivers from which the sender would like to have immediate, non-suppressed feedback. These may be receivers which the sender algorithmically identified as potential, future CLRs or which have been pre-configured as potential congestion control points in the network. The NORM_CC_FLAG_RTT indicates the validity of the "cc_rtt" field for the associated receiver node. Normally, this flag will be set since the receivers in the list will typically be receivers from which the sender has received feedback. However, in the case that the NORM sender has been pre-configured with a set of PLR nodes, feedback from those receivers may not yet have been collected and thus the "cc_rtt" and "cc_rate" fields do not contain valid values.

5.5.2.2 NORM_CMD(CC) Feedback Response

Receivers explicitly respond to NORM_CMD(CC) messages in the form of a NORM_ACK(RTT) message. Receivers that are are marked as CLR or PLR nodes in the NORM_CMD(CC) "cc_node_list" immeditately provide feedback in the form of a NORM_ACK to this message. When a NORM_CMD(CC) is received, non-CLR or non-PLR nodes initiate random feedback backoff timeouts similar to that used when the receiver initiates a repair cycle (see Section 5.3) in response to detection of data loss. The goal of the congestion control feedback is to determine the receivers with the lowest congestion control rates. As described in [21], the receiver congestion control feedback (ACK) timeouts can be biased in favor of lower rate receivers (while maintaining effective feedback suppression). Such biasing is not necessarily possible with suppression of NORM_NACK messages since previous data and repair loss history may not be correlated with the current data loss.

The backoff timeout for the congestion control response is picked and biased as follows:

```
T_backoff = y*r*(K*GRTTsender) + (1 - y)*RandomBackoff(K*GRTTsender, GSIZEsender)
```

where

"y" is the fraction of (K*GRTT) used to offset the backoff with respect to the sender's current transmission rate. A value of y = 0.25 is recommended.

"r" is adjusted ratio of the local receiver's calculated rate to the sender's current rate. During steady-state congestion control operation, "r" is determined as:

```
r = (MAX(MIN((Rcalc / Rsender), 0.9), 0.5) - 0.5) / 0.4
```

During the "slow start" phase of congestion control operation, "r" is determined simply as:

```
r = Rrecv / Rsender
```

where "Rrecv" is the measured received rate. The receiver places a value equal to two times this "Rrecv" rate in the "cc_rate" field of its NORM_NACK or NORM_ACK feedback messages during the "slow start" phase of congestion control operation. If the sender chooses this rate as its congestion control rate, this prevents the sender from overshooting an appropriate rate by more than a factor of two during this "slow start" period when receivers have experienced no loss.

The RandomBackoff() algorithm provides a truncated exponentially distributed random number and is described in the NORM Building Block document [11]. The same backoff factor "K" used with the GRTT as for NORM_NACK suppression. As previously noted, a value of K = 4 is generally recommended for ASM operation and K = 6 for SSM operation. A receiver SHALL cancel the backoff timeout and thus its pending transmission of a NORM_ACK(RTT) message under the following conditions:

1) The receiver provides another feedback message (NORM_NACK or NORM_ACK) before the congestion control feedback timeout expires,

2) A "suppressing" NORM_ACK(RTT) message is heard from another receiver or via a NORM_CMD(REPAIR_ADV) message from the sender. The local receiver's feedback is canceled if the rate of the competing feedback (Rfb) is sufficiently close to or less than the local receiver's calculated rate (Rcalc). The local receiver's feedback is canceled when:

```
Rcalc > (0.9 * Rfb)
```

According to [21], this bias of suppression is recommended to help ensure that the receiver with the lowest rate reports, while still maintaining a low volume of feedback from the receiver set.

When the backoff timer expires, the receiver generates a NORM_ACK(RTT) message to provide feedback to the sender and group. This message may be multicast to the group for most effective suppression in ASM topologies or unicast to the sender depending upon how the NORM protocol is deployed and configured. In the congestion control feedback fields of any NORM_ACK or NORM_NACK messages, receivers will include an adjusted version of the sender timestamp from the most recently received NORM_CMD(CC) message and the greatest "cc_sequence" received. The receiver SHALL also set any applicable "cc_flags", its current "cc_rate", and its "cc_rtt" if known. The sender can use the receiver-provided previous "cc_rtt" value to smooth its RTT estimate when it is valid. As noted in [18], a smoothing constant of 0.5 is recommended for regular receivers and 0.9 for CLR (and PLR) receivers from which more rapid feedback is received.

During "slow start" (when the receiver has not yet detected loss from the sender), the receiver uses a value equal to two times its measured rate from the sender in the "cc_rate" field. For steady-state congestion control operation, the receiver "cc_rate" value is based on the equation based value using its current loss event estimate and sender<->receiver RTT information.

After a congestion control feedback message is generated or when the feedback is suppressed, the receiver begins a "holdoff" timeout period during which it will restrain itself from initiating another feedback cycle, even if NORM_CMD(CC) messages are received from the sender (unless the receive becomes marked as a CLR or PLR node). The value of this holdoff timeout period is:

```
T_holdoff = (K*GRTT)
```

Thus, non-CLR receivers are constrained to providing explicit congestion control feedback once per K*GRTT intervals. Note, however, that as the session progresses, different receivers will be responding to different NORM_CMD(CC) messages and there will be relatively continuous feedback of congestion control information while the sender is active.

5.5.2.3 Congestion Control Rate Adjustment

During steady-state operation, the sender will directly adjust its transmission rate to the rate indicated by the feedback from its currently selected CLR according to any limitations described in [18]. As noted there, the estimation of parameters (loss and RTT) for the CLR will generally constrain the rate changes possible within acceptable bounds. For rate increases, the sender SHALL observe a maximum rate of increase of one packet per RTT at all times during steady-state operation. Note that the sender SHALL maintain a smoothed RTT estimate for the CLR upon new feedback from the CLR where:

```
RTT_clr = 0.9 * RTT_clr + 0.1 * RTT_clrNew
```

"RTT_clrNew" is the new RTT calculated from the timestamp in the feedback message received from the CLR. The RTTclr is initialized to RTT_clrNew on the first feedback message received. Note that the same procedure is observed by the sender for PLR receivers and that if a PLR is "promoted" to PLR status, the smoothed estimate can be continued.

There are some additional periods besides steady-state operation which need to be considered in this protocol operation. These periods aare:

1) during session startup,
2) when no feedback is received from the CLR, and
3) when the sender has a break in data transmission.

During session startup, the congestion control operation SHALL observe a "slow start" procedure to quickly approach its fair bandwidth share. An initial sender startup rate is assumed where:

```
Rinitial = MIN(NormSegmentSize / GRTT, NormSegmentSize) bytes/second.
```

The rate is increased only when feedback is received from the receiver set. The "slow start" phase proceeds until any receiver provides feedback indicating that loss has occurred. Rate increase during "slow start" is applied as:

```
Rnew = Rrecv_min
```

where "Rrecv_min" is the minimum reported receiver rate in the "cc_rate" field of congestion control feedback messages received from the group. Note that during "slow start", receivers use two times their measured rate from the sender in the "cc_rate" field of their feedback. Rate increase adjustment is limited to once per GRTT during slow start.

If the CLR or any receiver intends to leave the group, it will set the NORM_CC_FLAG_LEAVE in its congestion control feedback message as an indication that the sender should not select it as the CLR. When the CLR changes to a lower rate receiver, the sender should immediately adjust to the new lower rate. The sender is limited to increasing its rate at one additional packet per RTT towards a new, higher CLR rate.

The sender should also track the "age" of the feedback it has received from the CLR by comparing its current "cc_sequence" value (Ssender) to the last "cc_sequence" value received from the CLR (Sclr). As the "age" of the CLR feedback increases with no new feedback, the sender SHALL begin reducing its rate once per CLR RTT as a congestion avoidance measure.

The following algorithm is used to determine the decrease in sender rate (Rsender bytes/sec) as the CLR feedback, unexpectedly, excessively ages:

```
Age = Ssender - Sclr;
rate1 = MAX((Rsender - NormSegmentSize), 0.0);  // bytes per sec
rate2 = Rsender * 0.5
if (Age > 4)
      Rsender = MIN(rate1, rate2);
else if (Age > 2)
      Rsender = MAX(rate1, rate2);
```

This rate reduction occurs limited to the lower bound on NORM transmission rate. After NORM_ROBUST_FACTOR consecutive NORM_CMD(CC) rounds without any feedback from the CLR, the sender SHOULD assume the CLR has left the group and pick the receiver with the next lowest rate as the new CLR. Note this assumes that the sender does not have explicit knowledge that the CLR intentionally left the group. After such a CLR timeout, the sender will be transmitting with a minimal rate and should return to slow start as described here for a break in data transmission.

When the sender has a break in its data transmission, it can continue to probe the group with NORM_CMD(CC) messages to maintain RTT collection from the group. This will enable the sender to quickly determine an appropriate CLR upon data transmission restart. However, the sender should exponentially reduce its target rate to be used for transmission restart as time since the break elapses. The target rate SHOULD be recalculated once per CLR RTT as:

```
Rsender = Rsender * 0.5;
```

Upon restart, the sender should set the NORM_FLAG_START flag in its NORM_CMD(CC) messages and the group should observe "slow start" congestion control procedures until any receiver experiences a new loss event.

5.5.3 NORM Positive Acknowledgment Procedure

NORM provides an option for the source application to request positive acknowledgment (ACK) of NORM_CMD(ACK_REQ) messages from members of the group. There are a few types of specific acknowledgement requests that are defined for the NORM protocol and a range of acknowledgment request types which left to be defined by the application. One predefined acknowledgement type is the NORM_ACK(WATERMARK) that is used to determine if receivers have acheived completion of reliable reception up through an identified transmission point with respect to the sender's logical sequence of transmission. The NORM_ACK(WATERMARK) acknowledgement may be used to assist in application flow control when the sender has information on a portion of the receiver set. Another predefined acknowledgement type is NORM_ACK(CC), which is used to explicitly provide congestion control feedback in response to NORM_CMD(CC) messages transmitted by the sender. Note the NORM_ACK(CC) response does NOT follow the positive acknowledgement procedure described here. The NORM_CMD(ACK_REQ) and NORM_ACK messages contain an "ack_type" field to identify the type of acknowledgement requested and provided. A range of "ack_type" values is provided for application-defined use. While the application initiates the acknowledgement request and interprets application-defined "ack_type" values, the acknowledgment request and response is conducted with the following procedure.

The NORM positive acknowledgement procedure uses polling by the sender to query the receiver group for response. Note this polling procedure is not intended to scale to very large receiver groups, but could be used in large group setting to query a critical subset of the group. The NORM_CMD(ACK_REQ) message is used for polling and contains a list of NormNodeIds for receivers that should respond to the command. The list of receivers providing acknowledgement is determined by the source application with "a priori" knowledge of participating nodes or via some other application-level mechanism.

The ACK process is initiated by the sender who generates NORM_CMD(ACK_REQ) messages in periodic "rounds". For NORM_ACK(WATERMARK), these requests contain the "object_transport_id", "fec_block_number", and "fec_symbol_id" denoting the watermark transmission point. For application-defined requests, the "ack_req_content" field of the NORM_CMD(ACK_REQ) is set and interpreted by the sender and receiver applications, respectively. In response to the NORM_CMD(ACK_REQ), the listed receivers randomly spread NORM_ACK messages uniformly in time over a window of (1*GRTT). These NORM_ACK messages are typically unicast to the sender.

The ACK process is self-limiting and avoids ACK implosion in that:

1) Only a single NORM_CMD(ACK_REQ) message is generated once per (2*GRTT), and

2) The size of the "acking_node_list" of NormNodeIds from which ACK is requested is limited to a maximum of the sender NormSegmentSize setting per round of the positive acknowledgement process.

Because the size of the included list is limited to the sender's NormSegmentSize setting, multiple NORM_CMD(ACK_REQ) rounds may be required to achieve responses from all receivers specified. The content of the attached NormNodeId list will be dynamically updated as this process progresses and ACKs are received from the specified receiver set. Thus, as the sender receives responses from receivers, it eliminates them from the subsequent NORM_CMD(ACK_REQ) message payload list and adds in any pending receiver NormNodeIds keeping within the NormSegmentSize limitation of the list size. Each receiver is queried a maximum number of times (NORM_ROBUST_FACTOR, by default). Receivers not responding within this number of repeated requests are removed from the payload list to make potential room for other receivers pending acknowledgement. The transmission of the NORM_CMD(ACK_REQ) is repeated until no further responses are required or until the repeat threshold is exceeded for all pending receivers. The transmission of NORM_CMD(ACK_REQ) messages to conduct the positive acknowledgment process is multiplexed with ongoing sender data transmissions. However, the positive acknowledgment process may be interrupted in response to negative acknowledgement repair requests (NACKs) received from receivers during the acknowledgment period. The ACK process is resumed once any pending repairs have been transmitted.

In the case of NORM_CMD(ACK_REQ(WATERMARK)) commands, receivers will not ACK until they have received complete transmission of all data up to and including the watermark transmission point. All receivers SHALL interpret the watermark point provided in the request in the same manner as the transmission point given in NORM_CMD(FLUSH) messages and NACK for repairs if needed.

5.5.4 Group Size Estimation

NORM sender messages contain a "gsize" field that is a representation of the group size and is used in scaling random backoff timer ranges. The use of the group size estimate within the NORM protocol does not require a precise estimation and works reasonably well if the estimate is within an order of magnitude of the actual group size. By default, the NORM sender group size estimate may be administratively configured. Also given the expected scalability of the NORM protocol for general use, a default value of 10,000 is recommended for use as the group size estimate.

It is possible that group size may be algorithmically approximated from the volume of congestion control feedback messages which follow the exponentially weighted random backoff. However, the specification of such an algorithm is currently beyond the scope of this document.

5.5.5 Operation with Generic Router Assist (GRA)

NORM packet formats will be extended to allow for operation with GRA reliable multicast functions. Additional NACK suppression and selective sub-casting of repair transmissions in the network will be possible with GRA. (Section 5.4.2 discusses some NORM mechanisms related to this). Additional details will be provide in future versions of this document as GRA specifications mature.

6.0 Security Considerations

The same security considerations that apply to the NORM, FEC, and TFMCC building blocks also apply to the NORM protocol. In addition to vulnerabilities that any IP and IP multicast protocol implementation may be generally subject to, the NACK based feedback of NORM may be exploited by replay attacks which force the NORM sender to unnecessarily transmit repair information. This MAY be addressed by network layer IP security implementations that guard against this potential security exploitation. It is RECOMMENDED that such IP security mechanisms be used when available. Another possible approach is for NORM senders to use the "sequence" field from the NORM Common Message Header to detect replay attacks. This can be accomplished if the sender is willing to maintain state on receivers which are NACKing. A cache of receiver state may provide some protection against replay attacks. Note that the "sequence" field should be incremented by NormNodes with independent values for "sender" messages versus "receiver" messages so that the congestion control loss estimation function of the "sequence" field can be preserved for sender messages when receiver messages are unicast to the sender.

While NORM does leverage FEC-based repair for scalability, this does not alone guarantee integrity of received data. Application-level integrity-checking of data content is highly RECOMMENDED.

The NORM protocol is compatible with the use of the IP security (IPSEC) architecture described in [23].

7.0 Suggested Use

The present NORM protocol is seen as useful tool for the reliable data transfer over generic IP multicast services. It is not the intention of the authors to suggest it is suitable for supporting all envisioned multicast reliability requirements. NORM provides a simple and flexible framework for multicast applications with a degree of concern for network traffic implosion and protocol overhead efficiency. NORM-like protocols have been successfully demonstrated within the MBone for bulk data dissemination applications, including weather satellite compressed imagery updates servicing a large group of receivers and a generic web content reliable "push" application.

In addition, this framework approach has some design features making it attractive for bulk transfer in asymmetric and wireless internetwork applications. NORM is capable of successfully operating independent of network structure and in environments with high packet loss, delay, and misordering. Hybrid proactive/reactive FEC-based repairing improve protocol performance in some multicast scenarios. A sender-only repair approach often makes additional engineering sense in asymmetric networks. NORM's unicast feedback capability may be suitable for use in asymmetric networks or in networks where only unidirectional multicast routing/delivery service exists. Asymmetric architectures supporting multicast delivery are likely to make up an important portion of the future Internet structure (e.g., DBS/cable/PSTN hybrids) and efficient, reliable bulk data transfer will be an important capability for servicing large groups of subscribed receivers.

8.0 References

[1] Kermode, R., Vicisano, L., "Author Guidelines for Reliable Multicast Transport (RMT) Building Blocks and Protocol Instantiation documents", RFC 3269, April 2002.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Mankin, A., Romanow, A., Bradner, S. and V. Paxson, "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.

[4] Whetten, B., Vicisano, L., Kermode, R., Handley, M., Floyd S. and Luby, M., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer", RFC 3048, January 2001.

[5] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[6] Handley, M., Perkins, C. and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[7] S. Pingali, D. Towsley, J. Kurose, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", In Proc. INFOCOM, San Francisco CA, October 1993.

[8] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M. and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, December 2002.

[9] J. Macker, R. Adamson, "The Multicast Dissemination Protocol (MDP) Toolkit", Proc. IEEE MILCOM 99, October 1999.

[10] J. Nonnenmacher and E. Biersack, "Optimal Multicast Feedback", Proc. IEEE INFOCOMM, p. 964, March/April 1998.

[11] J. Macker, R. Adamson, "Quantitative Prediction of Nack Oriented Reliable Multicast (NORM) Feedback", Proc. IEEE MILCOM 2002, October 2002.

[12] T. Speakman, L. Vicisano, "Reliable Multicast Transport Building Block Generic Roouter Assist - Signalling Protocol Specification", Internet Draft draft-ietf-rmt-bb-gra-signalling-01.txt, January 2003, work in progress. Citation for informational purposes only.

[13] Deering, S., "Host Extensions for IP Multicasting", STD 5, RFC 1112, August 1989.

[14] Holbrook, H. W., "A Channel Model for Multicast", Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California, August 2001.

[15] B. Adamson, C. Bormann, M. Handley, and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Protocol Building Blocks", Internet Draft draft-ietf-rmt-bb-norm-05.txt, March 2003, work in progress. Citation for informational purposes only.

[16] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, December 2002.

[17] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward Error Correction (FEC) Building BLock", RFC 3452, December 2002.

[18] J. Widmer, M. Handley, "TCP-Friendly Multicast Congestion Control (TFMCC) Protocol Specification", Internet Draft draft-ietf-rmt-bb-tfmcc-01.txt, November 2002, work in progress. Citation for informational purposes only.

[19] D. Gossink, J. Macker, "Reliable Multicast and Integrated Parity Retransmission with Channel Estimation", IEEE GLOBECOMM 98', September 1998.

[20] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[21] J. Widmer and M. Handley, "Extending Equation-Based Congestion Control to Multicast Applications", Proc ACM SIGCOMM 2001, San Diego, August 2001.

[22] L. Rizzo, "pgmcc: A TCP-Friendly Single-Rate Multicast Congestion Control Scheme", Proc ACM SIGCOMM 2000, Stockholm, August 2000.

[23] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

7.0 Authors' Addresses

Brian Adamson
adamson@itd.nrl.navy.mil
Naval Research Laboratory
Washington, DC, USA, 20375

Carsten Bormann
cabo@tellique.de
Tellique Kommunikationstechnik GmbH
Gustav-Meyer-Allee 25 Geb ude 12
D-13355 Berlin, Germany

Mark Handley
mjh@aciri.org
1947 Center Street, Suite 600
Berkeley, CA 94704

Joe Macker
macker@itd.nrl.navy.mil
Naval Research Laboratory
Washington, DC, USA, 20375