Reliable Multicast Transport (RMT) Working Group
Internet-Draft
Expires: 17 March 2006

B. Adamson
NRL
C. Bormann
Universitaet Bremen TZI
M. Handley
UCL
J. Macker
NRL
October 2005

Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol draft-ietf-rmt-pi-norm-revised

Abstract

This document describes the messages and procedures of the Negative-acknowledgment (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 acknowledgment mechanism for transport reliability and offers additional protocol mechanisms to allow for operation with minimal "a priori" coordination among senders and receivers. A congestion control scheme is specified to allow the NORM protocol to 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) between the senders and 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.

Table of Contents

1. Introduction and Applicability

The Negative-acknowledgment (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 in 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 queuing 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].

Statement of Intent

This memo contains part of the definitions necessary to fully specify a Reliable Multicast Transport protocol in accordance with RFC 2357. As per RFC 2357, the use of any reliable multicast protocol in the Internet requires an adequate congestion control scheme.

While waiting for such a scheme to be available, or for an existing scheme to be proven adequate, the Reliable Multicast Transport working group (RMT) publishes this Request for Comments in the "Experimental" category.

It is the intent of RMT to re-submit this specification as an IETF Proposed Standard as soon as the above condition is met.

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) [7], Session Announcement Protocol (SAP) [8], etc.).

The NORM protocol design is principally driven by 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 independent of one another and receivers will maintain state as necessary for each 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 *NormTransportId*s 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.

To summarize, the NORM protocol provides 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. NORM senders provide for repair transmission of data and/or FEC 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 acknowledgment (NACK) based protocol schemes when feedback for reliability is required [9]. 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 [10]. FEC-based repair can be used to greatly reduce the quantity of reliable multicast repair requests and repair transmissions [11] in a NACK-oriented protocol. 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 [12]. 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 [13]. 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 each active sender. This may constrain the number of simultaneous senders in some uses of NORM.

1.3. Environmental Requirements and Considerations

All of the environmental requirements and considerations that apply to the RMT NORM Building Block [4] and the RMT FEC Building Block [5] also apply to the NORM protocol.

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. While the techniques utilized in NORM are principally applicable to "flat" end-to-end IP 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 [3], 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. 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 distinguish 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*. While 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 that can be leveraged for such application layer coordination.

As previously 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 stream service 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 possibly 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 symbols for transmission by the sender. In NORM, an FEC encoding symbol directly corresponds to the payload of `NORM_DATA` messages or "segment". Note that when systematic FEC codes are used, the payload of `NORM_DATA` messages sent for the first portion of a FEC encoding block are source symbols (actual segments of original user data), while the remaining symbols for the block consist of parity symbols generated by FEC encoding. These parity symbols are generally sent in response to repair requests, but some number may be sent proactively at the end each encoding block to increase the robustness of transmission. When non-systematic FEC codes are used, all symbols sent consist of FEC encoding parity content. In this case, the receiver must receive a sufficient number of symbols to reconstruct (via FEC decoding) the original user data for the given block. In this document, the terms "symbol" and "segment" are used interchangeably.

Transmitted *NormObjects* are temporarily yet uniquely identified within the *NormSession* context using the given sender's *NormNodeId*, *NormInstanceId*, and a temporary *NormObjectTransportId*. Depending upon the implementation, individual NORM senders may manage their *NormInstanceIds* independently, or a common *NormInstanceId* may be agreed upon for all participating nodes within a session if needed as a session identifier. NORM *NormObjectTransportId* 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. The individual segments or symbols of the *NormObject* are further identified with FEC payload identifiers which include coding block and symbol identifiers. These are discussed in detail later in this document.

2.1. Protocol Operation Overview

A NORM sender primarily generates messages of type `NORM_DATA`. These messages carry original data segments or FEC symbols and repair segments/symbols for the bulk data/file or stream *NormObjects* being transferred. By default, redundant FEC symbols are sent only in response to receiver repair requests (NACKs) and thus normally little or no additional transmission overhead is imposed due to FEC encoding. However, the NORM implementation MAY be optionally configured to proactively transmit some amount of redundant FEC symbols along with the original content to potentially enhance performance (e.g., improved delay) at the cost of additional transmission overhead. This option may be sensible for certain network conditions and can allow for robust, asymmetric multicast (e.g., unidirectional routing, satellite, cable) [15] 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 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. When the `NORM_INFO` message type is used, its transmission should precede transmission of any `NORM_DATA` message for the associated *NormObject*.

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 acknowledgment requests or application defined commands. The transmission of `NORM_CMD` messages from the sender is accomplished by one of three different procedures. These procedures 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 that can make use of information available to the underlying NORM protocol engine (e.g., round-trip timing, transmission rate, etc.).

All sender 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 that is compatible with coexistent TCP flows.

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 acknowledgment from receivers, other `NORM_ACK` messages are defined and available for use.

2.2. Protocol Building Blocks

The operation of the NORM protocol is based primarily upon the concepts presented in the Nack-Oriented Reliable Multicast (NORM) Building Block document [4]. This includes the basic NORM architecture and the data transmission, repair, and feedback strategies discussed in that document. Additional reliable multicast building blocks are applied in creating the full NORM protocol instantiation [16]. NORM also makes use of Forward Error Correction encoding techniques for repair messaging and optional transmission robustness as described in [10]. NORM uses the FEC Payload ID as specified by the FEC Building Block Document [5]. Additionally, for congestion control, this document includes a baseline congestion control mechanism (NORM-CC) based on the TCP-Friendly Multicast Congestion Control (TFMCC) scheme described in [19].

2.3. 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 when allowed more buffering resources than typical point-to-point transport protocols. This is because NORM feedback suppression is based upon randomly-delayed transmissions from the receiver set, rather than immediately transmitted feedback. 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. Conformance Statement

This Protocol Instantiation document, in conjunction with the RMT Building Block documents of [4] and [5], completely specifies a working reliable multicast transport protocol that conforms to the requirements described in RFC 2357 [17].

This document specifies the following message types and mechanisms which are REQUIRED in complying NORM protocol implementations:

Message Type	Purpose
NORM_DATA	Sender message for application data transmission. Implementations must support at least one of the NORM_OBJECT_DATA, NORM_OBJECT_FILE, or NORM_OBJECT_STREAM delivery services. The use of the NORM FEC Object Transmission Information header extension is OPTIONAL with NORM_DATA messages.
NORM_CMD(FLUSH)	Sender command to excite receivers for repair requests in lieu of ongoing NORM_DATA transmissions. Note the use of the NORM_CMD(FLUSH) for positive acknowledgment of data receipt is OPTIONAL.
NORM_CMD(SQUELCH)	Sender command to advertise its current valid repair window in response to invalid requests for repair.
NORM_CMD(REPAIR_ADV)	Sender command to advertise current repair (and congestion control state) to group when unicast feedback messages are detected. Used to control/suppress excessive receiver feedback in asymmetric multicast topologies.
NORM_CMD(CC)	Sender command used in collection of round trip timing and congestion control status from group (this may be OPTIONAL if alternative congestion control mechanism and round trip timing collection is used).
NORM_NACK	Receiver message used to request repair of missing transmitted content.
NORM_ACK	Receiver message used to proactively provide feedback for congestion control purposes. Also used with the OPTIONAL NORM Positive Acknowledgment Process.

This document also describes the following message types and associated mechanisms which are OPTIONAL for complying NORM protocol implementations:

Message Type	Purpose
NORM_INFO	Sender message for providing ancillary context information associated with NORM transport objects. The use of the NORM FEC Object Transmission Information header extension is OPTIONAL with NORM_INFO messages.
NORM_CMD(EOT)	Sender command to indicate it has reached end-of-transmission and will no longer respond to repair requests.
NORM_CMD(ACK_REQ)	Sender command to support application-defined, positively acknowledged commands sent outside of the context of the bulk data content being transmitted. The NORM Positive Acknowledgment Procedure associated with this message type is OPTIONAL.
NORM_CMD(APPLICATION)	Sender command containing application-defined commands sent outside of the context of the bulk data content being transmitted.
NORM_REPORT	Optional message type reserved for experimental implementations of the NORM protocol.

4. 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 auxiliary message type of NORM_REPORT is also provided for experimental purposes. This section describes 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. 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 and Extensions

There are some common message fields contained in all NORM message types. Additionally, a header extension mechanism is defined to expand the functionality of the NORM protocol without revision to this document. All NORM protocol messages begin with a common header with information fields as follows:

```
 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 |    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM Common Message Header Format

The "version" field is a 4-bit value indicating the protocol version number. NORM implementations SHOULD ignore received messages with version numbers different from their own. This number is intended to indicate and distinguish upgrades of the protocol which may be non-interoperable. The NORM version number for this specification is 1.

The message "type" field is a 4-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 8-bit "hdr_len" field indicates the number of 32-bit words that comprise the given message's header portion. This is used to facilitate header extensions that may be applied. The presence of header extensions are implied when the "hdr_len" value is greater than the base value for the given message "type".

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 a given destination address. A "sequence" field number space SHOULD be maintained for messages sent to the *NormSession* group address. This value can be monitored by receiving nodes to detect packet losses in the transmission from a sender and used in estimating raw packet loss for congestion control purposes. 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. With message authentication, the "sequence" field may also be leveraged for protection from message "replay" attacks, particularly of NORM_NACK or other feedback messages. In this case, the receiver node should maintain a monotonically increasing "sequence" field space for each destination to which it transmits (this may be multiple destinations when unicast feedback is used). 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 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 [18] 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.

NORM Header Extensions

When header extensions are applied, they follow the message type's base header and precede any payload portion. There are two formats for header extensions, both of which begin with an 8-bit "het" (header extension type) field. One format is provided for variable-length extensions with "het" values in the range from 0 through 127. The other format is for fixed length (one 32-bit word) extensions with "het" values in the range from 128 through 255. These formats are given here:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   het <=127   |      hel      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                    Header Extension Content                   |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM Variable Length Header Extension 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   het >=128   |    reserved   |    Header Extension Content   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM Fixed Length (32-bit) Header Extension Format

The "Header Extension Content" portion of these header extension format is defined for each header extension type defined for NORM messages. Some header extensions are defined within this document for NORM baseline FEC and congestion control operations.

4.2. 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 are used for various protocol control 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 format of `NORM_DATA` messages is comprised of three logical portions: 1) a fixed-format `NORM_DATA` header portion, 2) a FEC Payload ID portion with a format dependent upon the FEC encoding used, and 3) a payload portion containing source or encoded application data content. Note for objects of type `NORM_OBJECT_STREAM`, the payload portion contains additional fields used to appropriately recover stream content. NORM implementations MAY also extend the `NORM_DATA` header to include a FEC Object Transmission Information (EXT_FTI) header extension. This allows NORM receivers to automatically allocate resources and properly perform FEC decoding without the need for pre-configuration or out-of-band information.

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     flags     |    fec_id     |     object_transport_id       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         fec_payload_id                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                header_extensions (if applicable)              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       payload_msg_start*      |          payload_len*         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        payload_offset*                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         payload_data*                         |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_DATA Message Format

**NOTE: The "payload_msg_start", "payload_len" and "payload_offset" fields are present only for objects of type* `NORM_OBJECT_STREAM`.

The "payload_msg_start" field is used to mark the location (within the "payload_data") field of the start byte of an application-defined message boundary. Note that the "payload_msg_start" is the byte offset of the message boundary plus one. Thus, a value of "payload_msg_start" equal to ZERO denotes that no message boundary is present, while a "payload_msg_start" value of ONE indicates the message boundary is aligned with the beginning of the "payload_data" field. This allows NORM receiver applications to "synchronize" with NORM senders and to be able to properly interpret application layer data when joining a NORM session already in progress. The NORM sender implementation SHOULD provide a mechanism for the application to mark such message boundaries and set the "payload_msg_start" value accordingly. The "payload_msg_start" value will always be less than or equal to the "payload_len" value.

The "payload_len" and "payload_offset" fields allow senders to arbitrarily vary the size of `NORM_DATA` *payload segments for streams. This allows applications to flush transmitted streams as needed to meet unique streaming requirements. For objects of types* `NORM_OBJECT_FILE` *and* `NORM_OBJECT_DATA`*, these fields are unnecessary since the receiver can calculate the payload length and offset information from the "fec_payload_id" using the block partioning algorithm described in the FEC Building Block document [5]. When systematic FEC codes (e.g., "fec_id" = 129) are used, the "payload_msg_start", "payload_len" and "payload_offset" fields contain actual message start, length, and offset values for the associated application data segment (the "payload_data" field content) for those* `NORM_DATA` *messages containing source data symbols. In* `NORM_DATA` *messages that contain parity information, these fields do not contain values that can be* *directly interpreted, but instead are values computed from FEC encoding the "payload_msg_start", "payload_len" and "payload_offset" fields for the source data segments of the corresponding coding block.*

The "version", "type", "hdr_len", "sequence", and "source_id" fields form the NORM Common Message Header as described in Section 4.1. The value of the `NORM_DATA` "type" field is 2. The `NORM_DATA` _base_ "hdr_len" value is 4 (32-bit words) plus the size of the "fec_payload_id" field. The "fec_payload_id" field size depends upon the FEC encoding used for the referenced *NormObject*. The "fec_id" field is used to indicate the FEC coding type. For example, when small block, systematic codes are used, a "fec_id" value of 129 is indicated and the size of the "fec_payload_id" is two 32-bit words. In this case the `NORM_DATA` base "hdr_len" value is 6. The cumulative size of any header extensions applied is added into the "hdr_len" field.

The "instance_id" field contains a value generated by the sender to uniquely identify its current instance of participation in the *NormSession*. This allows receivers to detect when senders have perhaps left and rejoined a session in progress. When a sender (identified by its "source_id") is detected to have a new "instance_id", the NORM receivers SHOULD drop their previous state on the sender and begin reception anew, or at least treat this "instance" as a new, separate sender.

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 [19]). This value is used to control timing of the NACK repair process and other aspects of protocol operation as described in this document. Normally, the advertised "grtt" value will correspond to what the sender has measured based on feedback from the group, but, at low transmission rates, the advertised "grtt" SHOULD be set to MAX(grttMeasured, NormSegmentSize/senderRate) where the "NormSegmentSize" is sender's segment size in bytes and the "senderRate" is the sender's current transmission rate in bytes per second. The algorithm for encoding and decoding this field is described in the RMT NORM Building Block document [4].

The "backoff" field value is used by receivers to determine the maximum backoff timer value used in the timer-based NORM NACK feedback suppression. This 4-bit field supports values from 0-15 which is multiplied by the sender GRTT to determine the maximum backoff timeout. The "backoff" field informs the receiver set of the sender's backoff factor parameter "`Ksender`". Recommended values and their use are described in the NORM receiver NACK procedure description in Section 5.3. The "gsize" field contains a representation of the sender's current estimate of group size. This 4-bit field can roughly represent values from ten to 500 million where the most significant bit value of 0 or 1 represents a mantissa of 1 or 5, respectively and the three least significant bits incremented by one represent a base 10 exponent (order of magnitude). For examples, a field value of "0x0" represents 1.0e+01 (10), a value of "0x8" represents 5.0e+01 (50), a value of "0x1" represents 1.0e+02 (100), and a value of "0xf" represents 5.0e+08. For NORM feedback suppression purposes, the group size does not need to be represented with a high degree of precision. The group size may even be estimated somewhat conservatively (i.e., overestimated) to maintain low levels of feedback traffic. A default group size estimate of 10,000 ("gsize" = 0x4) is recommended for general purpose reliable multicast applications using the NORM protocol.

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 to meet 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.

NORM_FLAG_REPAIR 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. NORM_FLAG_EXPLICIT 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 could possibly be used by intermediate systems implementing functionality to control sub-casting of repair content to different legs of a reliable multicast topology with disparate repair needs. NORM_FLAG_INFO is set only when optional NORM_INFO content is actually available for the associated object. Thus, receivers will NACK for retransmission of NORM_INFO only when it is available for a given object. NORM_FLAG_UNRELIABLE is set when the sender wishes to transmit an object with only "best effort" delivery and will not supply repair transmissions for the object. NORM receivers SHOULD NOT execute repair requests for objects marked with the NORM_FLAG_UNRELIABLE flag. Note that receivers may inadvertently request repair of such objects when all segments (or info content) for those objects are not received (i.e., a gap in the "object_transport_id" sequence is noted). In this case, the sender should invoke the NORM_CMD(SQUELCH) process as described in Section 4.2.3. NORM_FLAG_FILE can be set as a "hint" from the sender that the associated object should be stored in non-volatile storage. NORM_FLAG_STREAM is set when the identified object is of type NORM_OBJECT_STREAM. The presence of NORM_FLAG_STREAM overrides that of NORM_FLAG_FILE with respect to interpretation of object size and the format of NORM_DATA messages.

The "fec_id" field corresponds to the FEC Encoding Identifier described in the FEC Building Block document [5]. The "fec_id" value implies the format of the "fec_payload_id" field and, coupled with FEC Object Transmission Information, the procedures to decode FEC encoded content. Small block, systematic codes ("fec_id" = 129) are expected to be used for most NORM purposes and the NORM_OBJECT_STREAM requires systematic FEC codes for most efficient performance.

The "object_transport_id" field is a monotonically and incrementally increasing value assigned by the sender to *NormObjects* 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 avoid 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 "source_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 "fec_payload_id" identifies the attached `NORM_DATA` "payload" content. The size and format of the "fec_payload_id" field depends upon the FEC type indicated by the "fec_id" field. These formats are given in the descriptions of specific FEC schemes as described in the IETF FEC Basic Schemes document [20]. or additional FEC Scheme documents that may be defined. As an example, the format of the "fec_payload_id" format for Small Block, Systematic codes ("fec_id" = 129) is given here:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      source_block_number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        source_block_len       |       encoding_symbol_id      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example FEC Payload ID Field ("fec_payload_id") Format
for Small Block, Systematic Codes ("fec_id" = 129)

In this example FEC payload identifier, the "source_block_number", "source_block_len", and "encoding_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 Basic Schemes document [20]. for the Small Block Systematic FEC Scheme identified by a "fec_id" value of 129. The "source_block_number" identifies the coding block's relative position with a *NormObject*. Note that, for *NormObjects* of type `NORM_OBJECT_STREAM`, the "source_block_number" may wrap for very long lived sessions. The "source_block_len" indicates the number of user data segments in the identified coding block. Given the "source_block_len" information of how many symbols of application data are contained in the block, the receiver can determine whether the attached segment is data or parity content and treat it appropriately. The "encoding_symbol_id" identifies which specific symbol (segment) within the coding block the attached payload conveys. Depending upon the value of the "encoding_symbol_id" and the associated "source_block_len" parameters for the block, the symbol (segment) referenced may be a user data or an FEC parity segment. For systematic codes, encoding symbols numbered less than the `source_block_len` contain original application data while segments greater than or equal to `source_block_len` contain parity symbols calculated for the block. The concatenation of `object_transport_id::fec_payload_id` can be viewed as a unique transport protocol data unit identifier for the attached segment with respect to the NORM sender's instance within a session.

Additional FEC Object Transmission Information (as described in the FEC Building Block document [5]) is required to properly receive and decode NORM transport objects. This information MAY be provided as out-of-band session information. However, in some cases, it may be useful for the sender to include this information "in-band" to facilitate receiver operation with minimal preconfiguration. For this purpose, the NORM FEC Object Transmission Information Header Extension (EXT_FTI) is defined. This header extension MAY be applied to NORM_DATA and NORM_INFO messages to provide this necessary information. The format of the EXT_FTI consists of two parts, a general part that contains the size of the associated transport object and a portion that depends upon the FEC scheme being used. The "fec_id" field in NORM_DATA and NORM_INFO messages identifies the FEC scheme. The format of the EXT_FTI general part is given here.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    het = 64   |    hel = 4    |       object_size (msb)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      object_size (lsb)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  FEC Scheme specific content ...              |
```

FEC Object Transmission Information Header Extension (EXT_FTI)

The header extension type "het" field value for the EXT_FTI header extension is 64. The header extension length "hel" value depends upon the format of the FTI for FEC code type identified by the "fec_id" field.

The 48-bit "object_size" field indicates the total length 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 by the sender to advertise 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.

As noted, the format of the extension depends upon the FEC code in use, but in general it SHOULD contain any required details on the FEC code in use (e.g., FEC Instance ID, etc.). As an example, the format of the EXT_FTI for small block systematic codes ("fec_id" = 129) is given here:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    het = 64   |    hel = 4    |       object_size (msb)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       object_size (lsb)                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       fec_instance_id         |          segment_size         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       fec_max_block_len       |        fec_num_parity         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example FEC Object Transmission Information Header Extension (EXT_FTI)
for Small Block, Systematic Codes ("fec_id" = 129)

In this example (for "fec_id" = 129), the "hel" field value is 4. The size of the EXT_FTI header extension may be different for other FEC schemes.

The 48-bit "object_size" serves the purpose described previously.

The "fec_instance_id" corresponds to the "FEC Instance ID" described in the FEC Building Block document [5]. In this case, the "fec_instance_id" is 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 [5]. 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. Typically, FEC parity symbol segments will be of this size.

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 "fec_num_parity" corresponds to the "maximum number 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 [5]. 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 payload portion of `NORM_DATA` messages includes source data or FEC encoded application content. Again, the content of this payload depends upon the FEC scheme being employed. Additionally, support for streaming using the `NORM_OBJECT_STREAM` type, necessitates some additional content in the payload.

The "payload_msg_start", "payload_len" and "payload_offset" fields are present ONLY for transport objects of type `NORM_OBJECT_STREAM`. For senders employing systematic FEC encoding, these fields contain values that can be interpreted directly for `NORM_DATA` messages containing original application source data content. But, for `NORM_DATA` messages containing calculated parity content, these fields will contain values computed by FEC encoding of the "payload_msg_start", "payload_len" and "payload_offset" values of the `NORM_DATA` data segments for the corresponding FEC coding block and cannot be interpreted directoly. The actual "payload_msg_start", "payload_len" and "payload_offset" values of missing data content can be determined upon decoding a FEC coding block. Note that these fields do NOT contribute to the value of the `NORM_DATA` "hdr_len" field. These fields are present only when the "flags" portion of the `NORM_DATA` message indicate the transport object is of type `NORM_OBJECT_STREAM`.

The "payload_msg_start" field, when set to a non-zero value, indicates that the associated "payload_data" content contains an application-defined message boundary (start-of-message). When such a message boundary is indicated, the first byte of an application-defined message, with respect to the "payload_data" field, will be found at an offset of "payload_msg_start - 1" bytes. Thus, if a `NORM_OBJECT_STREAM` `NORM_DATA` payload contains the start of an application message at the first byte of the "payload_data" field, the value of the "payload_msg_start" field will be '1'. Again, if the value of the "payload_msg_start" field is ZERO, no message boundary is indicated. It is RECOMMENDED that NORM implementations provide sender stream applications with a capability to mark message boundaries in this manner. Similarly, the NORM receiver SHOULD enable the application to recover such message boundary information. This enables NORM receivers to "synchronize" with transmitted message stream content in a meaningful way (i.e., meaningful to the application) at any time, whether joining the session late, or departing the session and returning.

The "payload_len" and "payload_offset" fields indicate the size and relative position (within the stream) of the source content contained in the "payload_data" field. Note that for long-lived streams, the "payload_offset" field may wrap.

The "payload_data" field contains the original application source or parity content for the symbol identified by the "fec_payload_id". The length of this field SHALL be limited to a maximum of the sender's *NormSegmentSize* bytes as given in the FTI for the object. Note the length of this field for messages containing parity content will always be of 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*. It is RECOMMENDED that a sender's *NormSegmentSize* generally be constant for the duration of a given sender's term of participation in the session, but may possibly vary on a per-object basis. 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 possibly leveraged at session startup to perform this configuration. The "payload_data" content may be delivered directly to the application for source symbols (when systematic FEC encoding is used) or upon decoding of the FEC block. For `NORM_OBJECT_FILE` and `NORM_OBJECT_STREAM` objects, the data segment length and offset can be calculated using the block paritioning algorithm described in the FEC Building Block document [5]. For `NORM_OBJECT_STREAM` objects, the length and offset is obtained from the segment's corresponding "payload_len" and "payload_offset" fields.

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*.

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     flags     |    fec_id     |     object_transport_id       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                header_extensions (if applicable)              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         payload_data                          |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_INFO Message Format

The "version", "type", "hdr_len", "sequence", and "source_id" fields form the NORM Common Message Header as described in Section 4.1. The value of "hdr_len" field when no header extensions are present is 4.

The "instance_id", "grtt", "backoff", "gsize", "flags", "fec_id", and "object_transport_id" 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.

As with NORM_DATA messages, the NORM FTI Header Extension (EXT_FTI) may be optionally applied to NORM_INFO messages. To conserve protocol overhead, some NORM implementations may wish to apply the EXT_FTI when used to NORM_INFO messages only and not to NORM_DATA messages.

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 Messages

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 messages begin with a common set of fields, after the usual NORM message common header. The standard NORM_CMD fields are:

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     flavor    |                                               |
+-+-+-+-+-+-+-+-+        NORM_CMD Content                       +
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD Standard Fields

The "version", "type", "hdr_len", "sequence", and "source_id" fields form the NORM Common Message Header as described in Section 4.1. The value of the "hdr_len" field for NORM_CMD messages without header extensions present depends upon the "flavor" field.

The "instance_id", "grtt", "backoff", 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"). NORM command flavors include:

Command	Flavor Value	Purpose
NORM_CMD(FLUSH)	1	Used to indicate sender temporary end-of-transmission. (Assists in robustly initiating outstanding repair requests from receivers). May also be optionally used to collect positive acknowledgment of reliable reception from subset of receivers.
NORM_CMD(EOT)	2	Used to indicate sender permanent end-of-transmission.
NORM_CMD(SQUELCH)	3	Used to advertise sender's current repair window in response to out-of-range NACKs from receivers.
NORM_CMD(CC)	4	Used for GRTT measurement and collection of congestion control feedback.
NORM_CMD(REPAIR_ADV)	5	Used to advertise sender's aggregated repair/feedback state for suppression of unicast feedback from receivers.
NORM_CMD(ACK_REQ)	6	Used to request application-defined positive acknowledgment from a list of receivers (OPTIONAL).
NORM_CMD(APPLICATION)	7	Used for application-defined purposes which may need to temporarily preempt data transmission (OPTIONAL).

4.2.3.1. 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 may indicate a temporary or permanent end of data transmission, but the sender is still willing to respond to repair requests. 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` unless a list of receivers from which explicit positive acknowledgment is expected ("acking_node_list") is given. In that case, the "acking_node_list" is updated as acknowledgments are received and the `NORM_CMD(FLUSH)` is repeated according to the mechanism described in Section 5.5.3. The greater the `NORM_ROBUST_FACTOR`, the greater the probability that all applicable receivers will be excited for acknowledgment or repair requests (NACKs) _and_ that the corresponding NACKs are delivered to the sender. A default value of `NORM_ROBUST_FACTOR` equal to 20 is RECOMMENDED. If a `NORM_NACK` message interrupts the flush process, the sender will re-initiate the flush process after any resulting repair transmissions are completed.

Note that receivers also employ a timeout mechanism to self-initiate NACKing (if there are outstanding repair needs) when no messages of any type 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 a terminal NACK process.

For finite-size transport objects such as NORM_OBJECT_DATA and NORM_OBJECT_FILE, the flush process (if there are no further pending objects) occurs at the end of these objects. Thus, FEC repair information is always 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 employed. 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.

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   flavor = 1  |    fec_id     |      object_transport_id      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         fec_payload_id                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                acking_node_list (if applicable)               |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(FLUSH) Message Format

The "version", "type", "hdr_len", "sequence", and "source_id" fields form the NORM Common Message Header as described in Section 4.1. 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 "fec_id" field indicates the FEC type used for the flushing "object_transport_id" and implies the size and format of the "fec_payload_id" field. Note the "hdr_len" value for the NORM_CMD(FLUSH) message is 4 plus the size of the "fec_payload_id" field when no header extensions are present.

The "object_transport_id" and "fec_payload_id" fields indicate the sender's current logical "transmit position". These fields are interpreted in the same manner as in the NORM_DATA message type. Upon receipt of 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 to the `NORM_CMD(FLUSH)` is generated.

For `NORM_OBJECT_STREAM` objects using systematic FEC codes, receivers MUST request "explicit-only" repair of the identified "source_block_number" if the given "encoding_symbol_id" is less than the "source_block_len". 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 "source_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 usual. Note that the use of a systematic FEC code is assumed here. Normal receiver NACK initiation and construction is discussed in detail in Section 5.3. The OPTIONAL "acking_node_list" field contains a list of *NormNodeIds* for receivers from which the sender is requesting explicit positive acknowledgment of reception up through the transmission point identified by the "object_transport_id" and "fec_payload_id" fields. The length of the list can be inferred from the length of the received `NORM_CMD(FLUSH)` message. When the "acking_node_list" is present, the lightweight positive acknowledgment process described in Section 5.5.3 SHALL be observed.

4.2.3.2. NORM_CMD(EOT) Message

The `NORM_CMD(EOT)` command is sent when the sender reaches permanent end-of-transmission with respect to the *NormSession* and will not respond to further repair requests. This allows receivers to gracefully reach closure of operation with this sender (without requiring any timeout) and free any resources that are no longer needed. The `NORM_CMD(EOT)` command SHOULD be sent with the same robust mechanism as used for `NORM_CMD(FLUSH)` commands to provide a high assurance of reception by the receiver set.

```
 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|    hdr_len    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   flavor = 2  |                  reserved                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(EOT) Message Format

The value of the "hdr_len" field for `NORM_CMD(EOT)` messages without header extensions present is 4. The "reserved" field is reserved for future use and MUST be set to an all ZERO value. Receivers MUST ignore the "reserved" field.

4.2.3.3. NORM_CMD(SQUELCH) Message

The `NORM_CMD(SQUELCH)` command is transmitted in response to outdated or 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 specific previously enqueued objects. The list also contains the identifiers for any objects within the repair window that 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:

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            source_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  flavor = 3   |    fec_id     |     object_transport_id       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         fec_payload_id                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        invalid_object_list                    |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(SQUELCH) Message Format

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 value of the "hdr_len" field when no extensions are present is 4 plus the size of the "fec_payload_id" field that is dependent upon the FEC scheme identified by the "fec_id" field.

The "object_transport_id" and "fec_payload_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). The "fec_id" field implies the size and format of the "fec_payload_id" field. This serves as an advertisement of a "synchronization point" for receivers to request repair. Note, that while an "encoding_symbol_id" may be included in the "fec_payload_id" field, the sender's repair window SHOULD be aligned on FEC coding block boundaries and thus the "encoding_symbol_id" SHOULD be zero.

The "invalid_object_list" is a list of 16-bit *NormTransportIds* that, although they are within the range of 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 can be determined from the packet's 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.

4.2.3.4. NORM_CMD(CC) Message

The `NORM_CMD(CC)` messages contains fields to enable sender-to-receiver group greatest round-trip time (GRTT) measurement and to excite the group for congestion control feedback. A baseline NORM congestion control scheme (NORM-CC), based on the TCP-Friendly Multicast Congestion Control (TFMCC) scheme of [19] is described in Section 5.5.2 of this document. The `NORM_CMD(CC)` message is usually transmitted as part of NORM-CC congestion control operation. A NORM header extension is defined below to be used with the `NORM_CMD(CC)` message to support NORM-CC operation. Different header extensions may be defined for the `NORM_CMD(CC)` (and/or other NORM messages as needed) to support alternative congestion control schemes in the future. If NORM is operated in a private network with congestion control operation disabled, the `NORM_CMD(CC)` message is then used for GRTT measurement only and may optionally be sent less frequently than with congestion control operation.

```
 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|    hdr_len    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   flavor = 4  |    reserved   |          cc_sequence          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         send_time_sec                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         send_time_usec                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               header extensions (if applicable)               |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  cc_node_list (if applicable)                 |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(CC) Message Format

The NORM common message header and standard `NORM_CMD` fields serve their usual purposes. The value of the "hdr_len" field when no header extensions are present is 6.

The "reserved" field is for potential future use and MUST be set to ZERO in this version of the NORM protocol and its baseline NORM-CC congestion control scheme. It may be possible that alternative congestion control schemes may use the `NORM_CMD(CC)` message defined here and leverage the "reserved" field for scheme-specific purposes.

The "cc_sequence" field is a sequence number applied by the sender. For NORM-CC operation, it is used to provide functionality equivalent to the "feedback round number" (`fb_nr`)described in [19]. The most recently received "cc_sequence" value is recorded by receivers and can be fed back to the sender in congestion control feedback generated by the receivers for that sender. The "cc_sequence" number can also be used in NORM implementations to assess how recently a receiver has received `NORM_CMD(CC)` probes from the sender. This can be useful instrumentation for complex or experimental multicast routing environments.

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 ("send_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 the time they received the `NORM_CMD(CC)` message to the time of their response as the "grtt_response" portion of `NORM_ACK` and `NORM_NACK` messages generated. This allows the sender to evaluate round-trip times to different receivers for congestion control and other (e.g., GRTT determination) purposes.

To facilitate the baseline NORM-CC scheme described in Section 5.5.2, a NORM-CC Rate header extension (EXT_RATE) is defined to inform the group of the sender's current transmission rate. This is used along with the loss detection "sequence" field of all NORM sender messages and the `NORM_CMD(CC)` GRTT collection process to support NORM-CC congestion control operation. The format of this header extension is as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    het = 128  |    reserved   |          send_rate            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM-CC Rate Header Extension Format (EXT_RATE)

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 portion and 4 bits of base 10 exponent (order of magnitude) information in the least significant portion. 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:

```
send_rate = (((int)(Value_mantissa * 4096.0 / 10.0 + 0.5)) << 4) | Value_exponent;
```

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:

```
send_rate = (((int)((3.2 * 4096.0 / 10.0) + 0.5)) << 4) | 4;

          = (0x51f << 4) | 0x4

          = 0x51f4
```

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

```
value = (send_rate >> 4) * 10.0 / 4096.0 * power(10.0, (send_rate & x000f))
```

Note the maximum transmission rate that can be represented by this scheme is approximately 9.99e+15 bytes per second.

When this extension is present, a "cc_node_list" may be attached as the payload of the `NORM_CMD(CC)` message. The presence of this header extension also implies that NORM receivers should respond according to the procedures described in Section 5.5.2. 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 that 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 maximum 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:

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Congestion Control Node List Item Format

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_FLAG_CC_CLR	0x01	Receiver is the current limiting receiver (CLR).
NORM_FLAG_CC_PLR	0x02	Receiver is a potential limiting receiver (PLR).
NORM_FLAG_CC_RTT	0x04	Receiver has measured RTT with respect to sender.
NORM_FLAG_CC_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 the receiver's actual measured receive rate).
NORM_FLAG_CC_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 RTT as measured by the sender with respect to the indicated receiver. This field is valid only if the NORM_FLAG_CC_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 [4]. 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.

4.2.3.5. NORM_CMD(REPAIR_ADV) Message

The NORM_CMD(REPAIR_ADV) message is used by the sender to "advertise" its aggregated repair state from 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(CC) (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 [13].

```
 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|    hdr_len    |           sequence            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  flavor = 5   |     flags     |           reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              header extensions (if applicable)                |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      repair_adv_payload                       |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(REPAIR_ADV) Message Format

The "instance_id", "grtt", "backoff", "gsize", and "flavor" fields serve the same purpose as in other `NORM_CMD` messages. The value of the "hdr_len" field when no extensions are present is 4.

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 fit its full current repair state into a single *NormSegmentSize*. If this flag is set, receivers should limit their NACK response to generating NACK content only up through the maximum ordinal transmission position *(objectId::fecPayloadId)* included in the "repair_adv_content".

When congestion control operation is enabled, a header extension may be applied to the `NORM_CMD(REPAIR_ADV)` representing the most limiting (in terms of congestion control feedback suppression) congestion control response. This allows the `NORM_CMD(REPAIR_ADV)` message to suppress receiver congestion control responses as well as NACK feedback messages. The field is defined as a header extension so that alternative congestion control schemes may be used with NORM without revision to this document. A NORM-CC Feedback Header Extension (EXT_CC) is defined to encapsulate congestion control feedback within `NORM_NACK`, `NORM_ACK`, and `NORM_CMD(REPAIR_ADV)` messages. If another congestion control technique (e.g., Pragmatic General Multicast Congestion Control (PGMCC) [21]) is used within a NORM implementation, an additional header extension MAY need to be defined encapsulate any required feedback content. The NORM-CC Feedback Header Extension format is:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     het = 3   |    hel = 3    |         cc_sequence           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    cc_flags   |     cc_rtt    |           cc_loss             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            cc_rate            |         cc_reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM-CC Feedback Header Extension (EXT_CC) Format

The "cc_sequence" field contains the current greatest "cc_sequence" value receivers have received in `NORM_CMD(CC)` messages from the sender. This information assists 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. For `NORM_CMD(REPAIR_ADV)` messages, the sender SHALL set the "cc_sequence" field value to the value set in the last `NORM_CMD(CC)` message sent.

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. These fields are used by receivers in controlling (suppressing as necessary) their congestion control feedback. For `NORM_CMD(REPAIR_ADV)` messages, the NORM_FLAG_CC_RTT should be set *only* when *all* feedback messages received by the sender have the flag set. Similarly, the NORM_FLAG_CC_CLR or NORM_FLAG_CC_PLR should be set only when *no* feedback has been received from non-CLR or non-PLR receivers. And the NORM_FLAG_CC_LEAVE should be set only when all feedback messages the sender has received have this flag set. These heuristics for setting the flags in `NORM_CMD(REPAIR_ADV)` ensure the most effective suppression of receivers providing unicast feedback messages.

The "cc_rtt" field SHALL be set to a default maximum value and the `NORM_FLAG_CC_RTT` flag SHALL be cleared when no receiver has yet received RTT measurement information. When a receiver has received RTT measurement information, it shall set the "cc_rtt" value accordingly and set the `NORM_FLAG_CC_RTT` flag in the "cc_flags" field. For `NORM_CMD(REPAIR_ADV)` messages, the sender SHALL set the "cc_rtt" field value to the largest non-CLR/non-PLR RTT it has measured from receivers for the current feedback round.

The "cc_loss" field represents 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
```

For `NORM_CMD(REPAIR_ADV)` messages, the sender SHALL set the "cc_loss" field value to the largest non-CLR/non-PLR loss estimate it has received from receivers for the current feedback round.

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_FLAG_CC_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. For `NORM_CMD(REPAIR_ADV)` messages, the sender SHALL set the "cc_loss" field value to the lowest non-CLR/non-PLR "cc_rate" report it has received from receivers for the current feedback round.

The "cc_reserved" field is reserved for future NORM protocol use. Currently, senders SHALL set this field to ZERO, and receivers SHALL ignore the content of this field.

The "repair_adv_payload" 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.

4.2.3.6. NORM_CMD(ACK_REQ) Message

The `NORM_CMD(ACK_REQ)` message is used by the sender to request acknowledgment from a specified list of receivers. This message is used in providing a lightweight positive acknowledgment mechanism that is OPTIONAL for use by the reliable multicast application. A range of acknowledgment 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 acknowledgment 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:

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  flavor = 6   |   reserved    |   ack_type    |    ack_id     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       acking_node_list                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(ACK_REQ) Message Format

The NORM common message header and standard `NORM_CMD` fields serve their usual purposes. The value of the "hdr_len" field for `NORM_CMD(ACK_REQ)` messages with no header extension present is 4.

The "ack_type" field indicates the type of acknowledgment 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_CC`	1	Used to identify `NORM_ACK` messages sent in response to `NORM_CMD(CC)` messages.
`NORM_ACK_FLUSH`	2	Used to identify `NORM_ACK` messages sent in response to `NORM_CMD(FLUSH)` messages.
`NORM_ACK_RESERVED`	3-15	Reserved for possible future NORM protocol use.
`NORM_ACK_APPLICATION`	16-255	Used at application's discretion.

The `NORM_ACK_CC` value is provided for use only in `NORM_ACKs` generated in response to the `NORM_CMD(CC)` messages used in congestion control operation. Similarly, the `NORM_ACK_FLUSH` is provided for use only in `NORM_ACKs` generated in response to applicable `NORM_CMD(FLUSH)` messages. `NORM_CMD`(ACK_REQ) messages with "ack_type" of `NORM_ACK_CC` or `NORM_ACK_FLUSH` SHALL NOT be generated by the sender.

The `NORM_ACK_RESERVED` range of "ack_type" values is provided for possible future NORM protocol use.

The `NORM_ACK_APPLICATION` range of "ack_type" values is provided so that NORM applications may implement application-defined, positively-acknowledged commands that are able to leverage internal transmission and round-trip timing information available to the NORM protocol implementation.

The "ack_id" provides a sequenced identifier for the given `NORM_CMD(ACK_REQ)` message. This "ack_id" is returned in `NORM_ACK` messages generated by the receivers so that the sender may associate the response with its corresponding request.

The "reserved" field is reserved for possible future protocol use and SHALL be set to ZERO by senders and ignored by receivers.

The "acking_node_list" field contains the *NormNodeIds* of the current NORM receivers that are desired to provide positive acknowledge (`NORM_ACK`) to this request. The packet payload length implies the length of the "acking_node_list" and its length is limited to the sender *NormSegmentSize*. The individual *NormNodeId* items are listed in network (Big Endian) byte order. If a receiver's *NormNodeId* is included in the "acking_node_list", it SHALL schedule transmission of a `NORM_ACK` message as described in Section 5.5.3.

4.2.3.7. 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 up to `NORM_ROBUST_FACTOR` times at a rate of once per `2*GRTT`. This rate of repetition allows the application to observe any response (if that is the application's purpose for the command) before it is repeated. Possible responses may include initiation of data transmission, other `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.

```
 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|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          instance_id          |     grtt      |backoff| gsize |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  flavor = 7   |                    reserved                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Application-Defined Content                  |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_CMD(APPLICATION) Message Format

The NORM common message header and `NORM_CMD` fields are interpreted as previously described. The value of the `NORM_CMD(APPLICATION)` "hdr_len" field when no header extensions are present is 4.

The "Application-Defined Content" area contains information in a format at the discretion of the application. The size of this payload SHALL be limited to a maximum of the sender's *NormSegmentSize* setting.

4.3. Receiver Messages

The NORM message types generated by participating 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 acknowledgment 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. `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:

```
 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=4|    hdr_len    |             sequence          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           server_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            instance_id        |             reserved          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_sec                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_usec                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                header extensions (if applicable)              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          nack_payload                         |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_NACK Message Format

The NORM common message header fields serve their usual purposes. The value of the "hdr_len" field for `NORM_NACK` messages without header extensions present is 6.

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

The "instance_id" field contains the current session identifier given by the sender identified by the "server_id" field in its sender messages. The sender SHOULD ignore feedback messages which contain an invalid "instance_id" value.

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 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.

For NORM-CC operation, the NORM-CC Feedback Header Extension, as described in the

`NORM_CMD(REPAIR_ADV}` message description, is added to `NORM_NACK` messages to provide feedback on the receivers current state with respect to congestion control operation. Note that alternative header extensions for congestion control feedback may be defined for alternative congestion control schemes for NORM use in the future.

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 up to the sender’s transmission position at the moment the receiver initiates the NACK Procedure as described in Section 5.3. A single NORM 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_payload" field of a `NORM_NACK` message. Note that a single NORM Repair Request can possibly include multiple "items", "ranges", or "erasure_counts". In turn, the "nack_payload" field may contain multiple repair requests. A single NORM Repair Request has the following 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             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      repair_request_items                     |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM Repair Request Format

The "form" field indicates the type of repair request items given in the "repair_request_items" list. Possible values for the "form" field include:

Form	Value
`NORM_NACK_ITEMS`	1
`NORM_NACK_RANGES`	2
`NORM_NACK_ERASURES`	3

A "form" value of `NORM_NACK_ITEMS` indicates each repair request item in the "repair_request_items" list is to be treated as an individual request. A value of `NORM_NACK_RANGES` indicates that the "repair_request_items" list consists of *pairs* of repair request items that correspond to inclusive ranges of repair needs. And the `NORM_NACK_ERASURES` "form" indicates that the repair request items are to be treated individually and that the "encoding_symbol_id" portion of the "fec_payload_id" field of the repair request item (see below) is to be interpreted as an "erasure count" for the FEC coding block identified by the repair request item’s "source_block_number".

The "flags" field is currently used to indicate the level of data content for which the repair request items apply (i.e., an individual segment, entire FEC coding block, or entire transport object). Possible flag values include:

Flag	Value	Purpose
NORM_NACK_SEGMENT	0x01	Indicates the listed segment(s) or range of segments are required as repair.
NORM_NACK_BLOCK	0x02	Indicates the listed block(s) or range of blocks in entirety are required as repair.
NORM_NACK_INFO	0x04	Indicates that NORM_INFO is required as repair for the listed object(s).
NORM_NACK_OBJECT	0x08	Indicates the listed object(s) or range of objects in entirety are required as repair.

When the NORM_NACK_SEGMENT flag is set, the "object_transport_id" and "fec_payload_id" fields are used to determine which sets or ranges of individual NORM_DATA segments are needed to repair content at the receiver. When the NORM_NACK_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. 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_payload_id" field is ignored when the flag NORM_NACK_OBJECT is set.

The "length" field value is the length in bytes of the "repair_request_items" field.

The "repair_request_items" field consists of a list of individual or range pairs of transport data unit identifiers in the following 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     fec_id    |   reserved    |      object_transport_id      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         fec_payload_id                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM Repair Request Item Format

The "fec_id" indicates the FEC type and can be used to determine the format of the "fec_payload_id" field. The "reserved" field is kept for possible future use and SHALL be set to a ZERO value and ignored by NORM nodes processing NACK content.

The "object_transport_id" corresponds to the *NormObject* for which repair is being requested and the "fec_payload_id" identifies the specific FEC coding block and/or segment being requested. When the `NORM_NACK_OBJECT` flag is set, the value of the "fec_payload_id" field is ignored. When the `NORM_NACK_BLOCK` flag is set, only the FEC code block identifier portion of the "fec_payload_id" is to be interpreted.

The format of the "fec_payload_id" field depends upon the "fec_id" field value.

When the receiver's repair needs dictate that different forms (mixed ranges and/or individual items) or types (mixed specific segments and/or blocks or objects in entirety) are required to complete reliable transmission, multiple NORM Repair Requests with different "form" and or "flags" values can be concatenated within a single `NORM_NACK` message. Additionally, NORM receivers SHALL construct `NORM_NACK` messages with their repair requests in ordinal order with respect to "object_transport_id" and "fec_payload_id" values. The "nack_payload" size SHALL NOT exceed the *NormSegmentSize* for the sender to which the `NORM_NACK` is destined.

NORM_NACK Content Examples:

In these examples, a small block, systematic FEC code ("fec_id" = 129) is assumed with a user data block length of 32 segments. In Example 1, a list of individual `NORM_NACK_ITEMS` repair requests is given. In Example 2, a list of `NORM_NACK_RANGES` requests _and_ a single `NORM_NACK_ITEMS` request are concatenated to illustrate the possible content of a `NORM_NACK` message. Note that FEC coding block erasure counts could also be provided in each case. However, 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 other FEC codes or for intermediate system purposes.

Example 1: `NORM_NACK` "nack_payload" 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 = 1    | flags = 0x01  |        length  = 36           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 12   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 3                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 2     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 12   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 3                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 5     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 12   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 3                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 8     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example 2: NORM_NACK "nack_payload" for: Object 18 Coding Block 6, Segments 5, 6, 7, 8, 9, 10; and Object 19 NORM_INFO and Coding Block 1, segment 3

```
 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 = 2    | flags = 0x01  |        length  = 24           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 18   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 6                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 5     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 18   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 6                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 10    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   form = 1    | flags = 0x05  |        length  = 12           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  fec_id = 129 |   reserved    |    object_transport_id = 19   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     source_block_number = 1                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    source_block_length = 32   |    encoding_symbol_id = 3     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.3.2. NORM_ACK Message

The NORM_ACK message is intended to be used primarily as part of NORM congestion control operation and round-trip timing measurement. As mentioned in the NORM_CMD(ACK_REQ) message description, the acknowledgment type NORM_ACK_CC is provided for this purpose. The generation of NORM_ACK(CC) messages for round-trip timing estimation and congestion-control operation is described in Sections 5.5.1 and 5.5.2, respectively. However, some multicast applications may benefit from some limited form of positive acknowledgment for certain functions. A simple, scalable positive acknowledgment scheme is defined in Section 5.5.3 that can be leveraged by protocol implementations when appropriate. The NORM_CMD(FLUSH) may be used for OPTIONAL collection of positive acknowledgment of reliable reception to a certain "watermark" transmission point from specific receivers using this mechanism. The NORM_ACK type NORM_ACK_FLUSH is provided for this purpose and the format of the "nack_payload" for this acknowledgment type is given below. Beyond that, a range of application-defined "ack_type" values is provided for use at the NORM application's discretion. Implementations making use of application-defined positive acknowledgments may also make use the "nack_payload" as needed, observing the constraint that the "nack_payload" field size be limited to a maximum of the *NormSegmentSize* for the sender to which the NORM_ACK is destined.

```
 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=5|    hdr_len    |          sequence             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           source_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           server_id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            instance_id        |    ack_type   |     ack_id    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_sec                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       grtt_response_usec                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                header extensions (if applicable)              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  ack_payload (if applicable)                  |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_ACK Message Format

The NORM common message header fields serve their usual purposes. The value of the "hdr_len" field when no header extensions are present is 6.

The "server_id", "instance_id", and "grtt_response" fields serve the same purpose as the corresponding fields in `NORM_NACK` messages. And header extensions may be applied to support congestion control feedback or other functions in the same manner.

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 to which this acknowledgment applies.

The "ack_id" field serves as a sequence number so that the sender can verify that a `NORM_ACK` message received actually applies to a current acknowledgment request. The "ack_id" field is not used in the case of the `NORM_ACK_CC` and `NORM_ACK_FLUSH` acknowledgment types.

The "ack_payload" 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_FLUSH`, a specific "ack_payload" format is defined:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     fec_id    |   reserved    |      object_transport_id      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         fec_payload_id                        |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

NORM_ACK_FLUSH "ack_payload" Format

The "object_transport_id" and "fec_payload_id" are used by the receiver to acknowledge applicable `NORM_CMD(FLUSH)` messages transmitted by the sender identified by the "server_id" field.

The "ack_payload" 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 Purpose Messages

Some additional message formats are defined for general purpose in NORM multicast sessions whether the participant is acting as a sender and/or receiver within the group.

4.4.1. NORM_REPORT Message

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. These report messages SHOULD be disabled for interoperability testing between different NORM implementations.

5. Detailed Protocol Operation

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

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` messages labeled with *NormTransportIds* and logically identified with FEC encoding block numbers and symbol identifiers. `NORM_INFO` messages may optionally precede the transmission of data content for NORM transport objects.

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 *objectId::fecPayloadId* transmit position and NACK _only_ for content ordinally prior to that transmit position. The receivers schedule 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 satisfied.

4) The sender aggregates repair requests from the receivers and logically "rewinds" its transmit position 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 (possibly 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 repairs.

5) The sender transmits `NORM_CMD(FLUSH)` messages when it reaches the end of enqueued transmit content and pending repairs. 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 transmissions are subject to rate control limits determined by congestion control mechanisms. In the baseline NORM-CC operation, each sender in a *NormSession* maintains its own independent congestion control state. Receivers provide congestion control feedback in `NORM_NACK` and `NORM_ACK` messages. `NORM_ACK` feedback for congestion control purposes is governed using a suppression mechanism similar to that for `NORM_NACK` messages.

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

5.1. Sender Initialization and Transmission

Upon startup, the NORM sender immediately begins sending `NORM_CMD(CC)` messages to collect round trip timing and other information from the potential group. If NORM-CC congestion control operation is enabled, the NORM-CC Rate header extension MUST be included in these messages. 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 use the `NORM_CMD(CC)` messaging to collect feedback from the group using the baseline NORM-CC feedback mechanisms. This proactive feedback collection can be used to establish a GRTT estimate prior to data transmission and potential NACK operation.

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. As noted, the periodic transmission of `NORM_CMD(CC)` messages may precede actual data transmission in order to have an initial GRTT estimate.

With inclusion of the OPTIONAL NORM FEC Object Transmission Information Header Extension (EXT_FTI), the NORM protocol sender message headers can contain all information necessary to prepare receivers for subsequent reliable reception. This includes FEC coding parameters, the sender *NormSegmentSize*, and other information. If this header extension is not used, it is presumed that receivers have received the FEC Object Transmission Information via other means. 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. These mechanisms allow for operation with minimal pre-coordination among the senders and receivers.

The NORM sender begins segmenting application-enqueued data into `NORM_DATA` segments and transmitting it to the group. For objects of type `NORM_OBJECT_DATA` and `NORM_OBJECT_FILE`, the segmentation algorithm described in FEC Building Block document. [5]. is RECOMMENDED. For objects of type `NORM_OBJECT_STREAM`, segmentation will typically be done into uniform FEC coding block sizes, with individual segment sizes controlled by the application, although in many cases, the application and NORM implementation should strive to produce full-sized (NormSegmentSize) segments when possible. The rate of transmission is controlled via congestion control mechanisms or is 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. The sender may attach an OPTIONAL "acking_node_list" to `NORM_CMD(FLUSH)` containing the *NormNodeIds* for receivers from which it expects explicit positive acknowledgment of reception. The `NORM_CMD(FLUSH)` message may be also used for this optional function any time prior to the end of data enqueued for transmission with the `NORM_CMD(FLUSH)` messages multiplexed with ongoing data transmissions. The OPTIONAL NORM positive acknowledgment procedure is described in Section 5.5.3.

5.1.1. Object Segmentation Algorithm

NORM senders and receivers MUST use a common algorithm for logically segmenting transport data into FEC encoding blocks and symbols so that appropriate NACKs can be constructed to request repair of missing data. NORM FEC coding blocks are comprised of multi-byte symbols (segments) that are transmitted in the payload of `NORM_DATA` messages. Each `NORM_DATA` message will contain one or more source or encoding symbol(s) identified by the "fec_payload_id" field and the *NormSegmentSize* sender parameter defines the maximum size (in bytes) of the "payload_data" field containing the content (a "segment"). The FEC encoding type and associated parameters govern the source block size (number of source symbols per coding block, etc.). NORM senders and receivers use these FEC parameters, along with the *NormSegmentSize* and transport object size to compute the source block structure for transport objects. These parameters are provided in the FEC Object Transmission Information for each object. The block partioning algorithm described in the FEC Building Block document [5] is RECOMMENDED for use to compute a source block structure such that all source blocks are as close to being equal length as possible. This helps avoid the performance disadvantages of "short" FEC blocks. Note this algorithm applies only to the statically-sized `NORM_OBJECT_DATA` and `NORM_OBJECT_FILE` transport object types where the object size is fixed and predetermined. For `NORM_OBJECT_STREAM` objects, the object is segmented according to the maximum source block length given in the FEC Transmission Information, unless the FEC Payload ID indicates an alternative size for a given block.

5.2. 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 limit or avoid repair requests for transport objects already 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. For general purpose operation, a default policy where receivers are allowed 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 objects of type `NORM_OBJECT_STREAM` it is RECOMMENDED that the join policy constrain receivers to start reliable reception at the current FEC coding block for which non-repair content is received.

For typical operation, it is expected that NORM receivers will join a specified multicast group and/or listen on an specific port number for sender transmissions. As the NORM receiver receives `NORM_DATA` messages it will provide content to its application as appropriate.

5.3. 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 FEC coding block boundaries, *NormObject* boundaries, upon receipt of a `NORM_CMD(FLUSH)` message, or upon an "inactivity" timeout when transmissions are no longer received from a previously active sender. The RECOMMENDED value of such an inactivity timeout is:

```
T_inactivity = NORM_ROBUST_FACTOR * MAX(GRTTsender, nominalRecvPacketInterval)
```

where the "`GRTTsender` value corresponds to the GRTT estimate advertised in the "grtt" field of NORM sender messages. The "`nominalRecvPacketInterval`" corresponds to a smoothed average (e.g., EWMA) of the interval between packets received from the sender (This prevents premature inactivity timeout when the `GRTTsender` value is small with respect to the sender application transmission rate). The NORM receiver SHOULD repeat NACKing initiated upon inactivity timeout for up to `NORM_ROBOUST_FACTOR` times or more depending upon the application's need for persistence by its receivers.

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 [4] using (`Ksender*GRTTsender`) for the "maxTime" parameter and the sender advertised group size (`GSIZEsender`) as the "groupSize" parameter. NORM senders provide values for `GRTTsender`, `Ksender` and `GSIZEsender` via the "grtt", "backoff", and "gsize" fields of transmitted messages. The `GRTTsender` value is determined by the sender based on feedback it has received from the group while the `Ksender` and `GSIZEsender` values may determined by application requirements and expectations or ancillary information. The backoff factor "`Ksender`" MUST be greater than one to provide for effective 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(Ksender*GRTTsender, GSIZEsender)
```

To avoid the possibility of NACK implosion in the case of sender or network failure during SSM operation, the receiver SHALL automatically suppress its NACK and immediately enter the "holdoff" period described below when `T_backoff` is greater than `(Ksender-1)*GRTTsender`. Otherwise, the backoff period is entered and the receiver MUST accumulate 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 *objectId::fecPayloadId*) 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 up to the sender transmission position at the time the NACK procedure (backoff timeout) was initiated.

If these conditions are met, the receiver immediately generates a `NORM_NACK` message when the backoff timeout expires. Otherwise, the receiver's NACK is considered to be "suppressed" and the message is not sent. At this time, the receiver begins a "holdoff" period during which it constrains itself to not reinitiate the NACKing process. The purpose of this timeout is to allow the sender worst-case time to respond to the repair needs before the receiver requests repair again. The value of this "holdoff" timeout (`T_rcvrHoldoff`) as described in [4] is:

```
T_rcvrHoldoff =(Ksender+2)*GRTTsender
```

The `NORM_NACK` message contains repair request content beginning with lowest ordinal repair position of the receiver up through the coding block prior to the most recently heard ordinal transmission position for the sender. If the size of the `NORM_NACK` content exceeds the sender's *NormSegmentSize*, the NACK content is truncated so that the receiver only generates a single `NORM_NACK` message per NACK cycle for a given sender. In summary, a single NACK message is generated containing the receiver's lowest ordinal repair needs.

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_ "encoding_symbol_id" (i.e., "encoding_symbol_id" = "source_block_len") and request the number of FEC 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 different, 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 plus the ordinally highest missing data segments required to satisfy its total 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* missed 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. Sender NACK Processing and Response

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" its logical transmission point 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 even when receivers experience uncorrelated data loss.

5.4.1. 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.

As described in [4], the period of time during which the sender aggregates `NORM_NACK` messages is equal to:

```
T_sndrAggregate = (Ksender+1)*GRTT
```

where "`Ksender`" 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. After pending repair transmissions are completed, the sender continues with new transmissions of any enqueued data. Also, at this point in time, the sender begins a "holdoff" timeout during which time the sender constrains itself from initiating a new repair aggregation cycle, even if `NORM_NACK` messages arrive. As described in [4], the value of this sender "holdoff" period is:

```
T_sndrHoldoff = (1*GRTT)
```

If additional `NORM_NACK` messages are received during this sender "holdoff" 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. Recall that receivers are not to initiate the NACK repair process until the sender's logical transmission position exceeds the lowest ordinal position of their repair needs. With the new NACK aggregation period, the sender repeats the same process of incorporating accumulated repair state into its transmission plan and subsequently "rewinding" to transmit the lowest ordinal repair data when the aggregation period expires. Again, this is conducted in concert with ongoing new data and/or pending repair transmissions.

5.4.2. 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 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 SHALL be flagged with the `NORM_FLAG_REPAIR` flag. This allows receivers to obey any policies that limit new receivers from joining the reliable transmission when only repair transmissions have been received. Additionally, the sender SHOULD additionally flag `NORM_DATA` transmissions sent as explicit repair with the `NORM_FLAG_EXPLICIT` flag.

Although NORM end system receivers do not make use of the `NORM_FLAG_EXPLICIT` flag, this message transmission status could be leveraged by intermediate systems wishing to "assist" NORM protocol performance. If such systems are properly positioned with respect to reciprocal reverse-path multicast routing, they need to sub-cast only a sufficient count of non-explicit parity repairs to satisfy a multicast routing sub-tree's erasure filling needs for a given FEC coding block. When the sender has resorted to explicit repair, then the intermediate systems should sub-cast all of the explicit repair packets to those portions of the routing tree still requiring repair for a given coding block. Note the intermediate systems 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 sub-casting. Additionally, the intermediate systems could perform additional `NORM_NACK` suppression/aggregation as it conducts this repair state accumulation for NORM repair cycles. The detail of this type of operation are beyond the scope of this document, but this information is provided for possible future consideration.

5.4.3. 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. 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 implicitly suppress requests for ordinally higher repair. For congestion control operation, the sender may also need to provide information so that dynamic congestion control feedback can be suppressed as needed among receivers. This document specifies the NORM-CC Feedback Header Extension that is applied for baseline NORM-CC operation. If other congestion control mechanisms are used within a NORM implementation, other header extensions may be defined. Whatever content format is used for this purpose should ensure that maximum possible suppression state is conveyed to the receiver set.

5.5. Additional 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. Greatest Round-trip Time 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 that 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 smoothes the values for a conservative estimate of the GRTT. The algorithm and methodology are described in the NORM Building Block document [4] 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 a minimum of the nominal inter-packet transmission time given the sender's current transmission rate and system clock granularity. 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 Rate header extension is present 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 proactive, but controlled, feedback from the group in the form of `NORM_ACK` messages. This provides for 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 without the NORM-CC Rate header extension. In this case, receivers will only provide GRTT measurement feedback when `NORM_NACK` messages are generated since no `NORM_ACK` messages are generated. In this case, the `NORM_CMD(CC)` messages may be sent less frequently, perhaps as little as once per minute, to conserve network capacity. Note that the NORM-CC Rate header extension may also be used proactively solicit RTT feedback from the receiver group per congestion control operation even though the sender may not be conducting congestion control rate adjustment. NORM operation without congestion control should be considered only in closed networks.

5.5.2. NORM Congestion Control Operation

This section describes baseline congestion control operation for the NORM protocol (NORM-CC). 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 [19]. This congestion control scheme is REQUIRED for operation within the general Internet unless the NORM implementation is adapted to use another IETF-sanctioned reliable multicast congestion control mechanism (e.g., PGMCC [21]). With this TFMCC-based approach, the transmissions of NORM senders are 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. 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-to-receiver packet loss estimates and RTTs 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 resulting calculated transmission rate is identified as the "current limiting receiver" (CLR). In the case of a "tie" (where candidate CLRs are within 10% of the same calculated rate), the receiver with the largest RTT value SHOULD be designated as the CLR.

As described in [22], a steady-state sender transmission rate, to be "friendly" with competing TCP flows can be 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. (In NORM, the "nominal" packet size can be 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. 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 current "worst path" in the group 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 measurement of RTTs, to inform the group of the congestion control CLR, and to provide feedback of individual RTT measurements to the receivers in the group. The NORM_CMD(CC) also provides for exciting feedback from OPTIONAL "potential limiting receiver" (PLR) nodes that may be determined administratively or possibly algorithmically based on congestion control feedback. PLR nodes are receivers that have been identified to have potential for (perhaps soon) becoming the CLR and thus immediate, up-to-date feedback is beneficial for congestion control performance. 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 transmitted 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) Until a CLR has been identified (based on previous receiver feedback) or when no data transmission is pending, the NORM_CMD(CC) interval is doubled up from its current interval to a maximum of once per 30 seconds. This results in a low duty cycle for NORM_CMD(CC) probing when no CLR is identified or there is no pending data to transmit. T}.sp 3) T{ When a CLR has been identified (based on receiver feedback) and data transmission is pending, the probing interval is set to the RTT between the sender and the CLR (RTT_clr).

4) Additionally, when the data transmission rate is low with respect to the RTT_clr interval used for probing, the implementation should ensure that no more than one NORM_CMD(CC) message is sent per NORM_DATA message when there is data pending transmission. 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 NORM-CC Rate Header Extension is applied to the NORM_CMD(CC) message and the sender advertises its current transmission rate in the "send_rate" field. The rate information is used by receivers 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_FLAG_CC_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 a RTT measurement get first priority. Of these, those with the greatest loss fraction receive precedence for list inclusion.

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

There are "cc_flag" values in addition to NORM_FLAG_CC_CLR that are used for other congestion control functions. The NORM_FLAG_CC_PLR flag value is used to mark additional receivers from that the sender would like to have immediate, non-suppressed feedback. These may be receivers that the sender algorithmically identified as potential future CLRs or that have been pre-configured as potential congestion control points in the network. The NORM_FLAG_CC_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" field does not contain a valid value when this flag is not set. Similarly, a value of ZERO for the "cc_rate" field here should be treated as an invalid value and be ignored for the purposes of feedback suppression, etc.

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. The goal of the congestion control feedback is to determine the receivers with the lowest congestion control rates. Receivers that are marked as CLR or PLR nodes in the NORM_CMD(CC) "cc_node_list" immediately 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 backoff timeout for the congestion control response is generated as follows:

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

The "RandomBackoff()" algorithm provides a truncated exponentially distributed random number and is described in the NORM Building Block document [4]. The same backoff factor K = Ksender MAY be used as with NORM_NACK suppression. However, in cases where the application purposefully specifies a very small Ksender backoff factor to minimize the NACK repair process latency (trading off group size scalability), it is RECOMMENDED that a larger backoff factor for congestion control feedback is maintained, since there may often be a larger volume of congestion control feedback than NACKs in many cases and some congestion control feedback latency may be tolerable where reliable delivery latency is not. As previously noted, a backoff factor 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 generates another feedback message (NORM_NACK or other NORM_ACK) before the congestion control feedback timeout expires (these messages will convey the current congestion control feedback information),

2) A NORM_CMD(CC) or other receiver feedback with an ordinally greater "cc_sequence" field value is received before the congestion control feedback timeout expires (this is similar to the TFMCC feedback round number),

3) When the T_backoff is greater than 1*GRTTsender. This prevents NACK implosion in the event of sender or network failure,

4) "Suppressing" congestion control feedback is heard from another receiver (in a NORM_ACK or NORM_NACK) or via a NORM_CMD(REPAIR_ADV) message from the sender. The local receiver's feedback is "suppressed" 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)
   ```

   Also note receivers that have not yet received an RTT measurement from the sender are suppressed only by other receivers that have not yet measured RTT. Additionally, receivers whose RTT estimate has "aged" considerably (i.e., they haven't been included in the NORM_CMD(CC) "cc_node_list" in a long time) may wish to compete as a receiver with no prior RTT measurement after some long term expiration period.

When the backoff timer expires, the receiver SHALL generate 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.

Whenever any feedback is generated (including this NORM_ACK(RTT) message), receivers include an adjusted version of the sender timestamp from the most recently received NORM_CMD(CC) message and the "cc_sequence" value from that command in the applicable NORM_ACK or NORM_NACK message fields. For NORM-CC operation, any generated feedback message SHALL also contain the NORM-CC Feedback header extension. The receiver provides its current "cc_rate" estimate, "cc_loss" estimate, "cc_rtt" if known, and any applicable "cc_flags" via this header extension.

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 from the equation-based value using its current loss event estimate and sender<->receiver RTT information. (The GRTT is used when the receiver has not yet measured its individual RTT).

The "cc_loss" field value reflects the receiver's current loss event estimate with respect to the sender in question.

When the receiver has a valid individual RTT measurement, it SHALL include this value in the "cc_rtt" field. The NORM_FLAG_CC_RTT MUST be set when the "cc_rtt" field is valid.

After a congestion control feedback message is generated or when the feedback is suppressed, a non-CLR receiver begins a "holdoff" timeout period during which it will restrain itself from providing congestion control feedback, 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 (T_ccHoldoff) period is:

```
T_ccHoldoff = (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. As noted in [19], 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.

The sender processes congestion control feedback from the receivers and selects the CLR based on the lowest rate receiver. Receiver rates are either determined directly from the *slow start* "cc_rate" provided by the receiver in the NORM-CC Feedback header extension or by performing the equation-based calculation using individual RTT and loss estimates ("cc_loss") as feedback is received.

The sender can calculate a current RTT for a receiver (RTT_rcvrNew) using the "grtt_response" timestamp included in feedback messages. When the "cc_rtt" value in a response is not valid, the sender simply uses this RTT_rcvrNew value as the receiver's current RTT (RTT_rcvr). For non-CLR and non-PLR receivers, the sender can use the "cc_rtt" value provided in the NORM-CC Feedback header extension as the receiver's previous RTT measurement (RTT_rcvrPrev) to smooth according to:

```
RTT_rcvr = 0.5 * RTT_rcvrPrev + 0.5 * RTT_rcvrNew
```

For CLR receivers where feedback is received more regularly, the sender SHOULD maintain a more smoothed RTT estimate 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 RTT_clr 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 CLR status, the smoothed estimate can be continued.

There are some additional periods besides steady-state operation that need to be considered in NORM-CC operation. These periods are:

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_FLAG_CC_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 any 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 (Seq_sender) to the last "cc_sequence" value received from the CLR (Seq_clr). As the "age" of the CLR feedback increases with no new feedback, the sender SHALL begin reducing its rate once per `RTT_clr` 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 = Seq_sender - Seq_clr;
if (Age > 4) Rsender = Rsender * 0.5;
```

This rate reduction is 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. If no receiver feedback is received, the sender MAY wish to withhold further transmissions of `NORM_DATA` segments and maintain `NORM_CMD(CC)` transmissions only until feedback is detected. 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 `RTT_clr` as:

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

If the minimum NORM rate is reached, the sender should set the NORM_FLAG_START flag in its `NORM_CMD(CC)` messages upon restart and the group should observer "slow start" congestion control procedures until any receiver experiences a new loss event.

5.5.3. NORM Positive Acknowledgment Procedure

NORM provides options for the source application to request positive acknowledgment (ACK) of `NORM_CMD(FLUSH)` and `NORM_CMD(ACK_REQ)` messages from members of the group. There are some specific acknowledgment requests defined for the NORM protocol and a range of acknowledgment request types that are left to be defined by the application. One predefined acknowledgment type is the `NORM_ACK_FLUSH` type. This acknowledgment is used to determine if receivers have achieved completion of reliable reception up through a specific logical transmission point with respect to the sender's sequence of transmission. The `NORM_ACK_FLUSH` acknowledgment may be used to assist in application flow control when the sender has information on a portion of the receiver set. Another predefined acknowledgment 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 for NORM-CC operation. Note the `NORM_ACK(CC)` response does NOT follow the positive acknowledgment procedure described here. The `NORM_CMD(ACK_REQ)` and `NORM_ACK` messages contain an "ack_type" field to identify the type of acknowledgment requested and provided. A range of "ack_type" values is provided for application-defined use. While the application is responsible for initiating the acknowledgment request and interprets application-defined "ack_type" values, the acknowledgment procedure SHOULD be conducted within the protocol implementation to take advantage of timing and transmission scheduling information available to the NORM transport.

The NORM positive acknowledgment 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. Either the `NORM_CMD(ACK_REQ)`, or when applicable, the `NORM_CMD(FLUSH)` message is used for polling and contains a list of *NormNodeIds* for receivers that should respond to the command. The list of receivers providing acknowledgment 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 that generates `NORM_CMD(FLUSH)` or `NORM_CMD(ACK_REQ)` messages in periodic "rounds". For `NORM_ACK_FLUSH` requests, the `NORM_CMD(FLUSH)` contain a "object_transport_id" and "fec_payload_id" denoting the watermark transmission point for which acknowledgment is requested. This watermark transmission point is "echoed" in the corresponding fields of the `NORM_ACK(FLUSH)` message sent by the receiver in response. `NORM_CMD(ACK_REQ)` messages contain an "ack_id" field which is similarly "echoed" in response so that the sender may match the response to the appropriate request.

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. (Note that `NORM_ACK(CC)` messages SHALL be multicast or unicast in the same manner as `NORM_NACK` messages).

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 acknowledgment is requested is limited to a maximum of the sender *NormSegmentSize* setting per round of the positive acknowledgment 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 NORM_ACK responses are received from the specified receiver set. As the sender receives valid responses (i.e., matching watermark point or "ack_id") from receivers, it SHALL eliminate those receivers from the subsequent NORM_CMD(ACK_REQ) message "acking_node_list" and add in any pending receiver *NormNodeIds* while 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 room for other potential receivers pending acknowledgment. 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) or NORM_CMD(FLUSH) messages to conduct the positive acknowledgment process is multiplexed with ongoing sender data transmissions. However, the NORM_CMD(FLUSH) positive acknowledgment process may be interrupted in response to negative acknowledgment repair requests (NACKs) received from receivers during the acknowledgment period. The NORM_CMD(FLUSH) positive acknowledgment process is restarted for receivers pending acknowledgment once any the repairs have been transmitted.

In the case of NORM_CMD(FLUSH) commands with an attached "acking_node_list", receivers will not ACK until they have received complete transmission of all data up to and including the given watermark transmission point. All receivers SHALL interpret the watermark point provided in the request NACK for repairs if needed as for NORM_CMD(FLUSH) commands with no attached "acking_node_list".

5.5.4. Group Size Estimate

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.

6. Security Considerations

The same security considerations that apply to the NORM, and FEC 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 NORM packets are cryptographically protected and 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 of NORM messages should be incremented with independent values for different destinations (e.g., group-addressed versus unicast-addressed messages versus "receiver" messages). Thus, the congestion control loss estimation function of the "sequence" field can be preserved for sender messages when receiver messages are unicast to the sender. The NORM protocol is compatible with the use of the IP security (IPsec) architecture described in [23]. It is important to note that 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.

7. IANA Considerations

No information in this specification is currently subject to IANA registration. However, several Header Extensions are defined within this document. If/when additional Header Extensions are developed, the first RFC MUST establish an IANA registry for them, with a "Specification Required" policy [6] and all Header Extensions, including those in the present document, MUST be registered thereafter. Additionally, building blocks components used by NORM may introduce additional IANA considerations. In particular, the FEC Building Block used by NORM does require IANA registration of the FEC codecs used. The registration instructions for FEC codecs are provided in [5].

8. 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.

9. Acknowledgments *(and these are not Negative)*

The authors would like to thank Rick Jones, Vincent Roca, Rod Walsh, Toni Paila, Michael Luby, and Joerg Widmer for their valuable input and comments on this document. The authors would also like to thank the RMT working group chairs, Roger Kermode and Lorenzo Vicisano, for their support in development of this specification, and Sally Floyd for her early input into this document.

10. References

10.1. Normative References

[1] Kermode, R. and L. Vicisano, "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] Deering, S., "Host Extensions for IP Multicasting", STD 5, RFC 1112, August 1989.

[4] Adamson, B., Bormann, C., Handley, M., and J. Macker, "Negative-Acknowledgement (NACK)-Oriented Reliable Multicast (NORM) Building Blocks", RFC 3941, November 2004.

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

[6] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

10.2. Informative References

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

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

[9] 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.

[10] 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.

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

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

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

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

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

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

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

[18] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

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

[20] Watson, M., "Basic Forward Error Correction (FEC) Schemes", Internet-Draft draft-ietf-rmt-bb-fec-basic-schemes-revised-00, July 2005.

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

[22] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", Proc ACM SIGCOMM 1998.

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

11. Authors' Addresses

Brian Adamson
Naval Research Laboratory
Washington, DC, USA, 20375

EMail: adamson@itd.nrl.navy.mil

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
D-28334 Bremen, Germany

EMail: cabo@tzi.org

Mark Handley
Department of Computer Science
University College London
Gower Street
London
WC1E 6BT
UK

EMail: M.Handley@cs.ucl.ac.uk

Joe Macker
Naval Research Laboratory
Washington, DC, USA, 20375

EMail: macker@itd.nrl.navy.mil

12. Full Copyright Statement

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the IETF's procedures with respect to rights in IETF Documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.