Network Working Group
Request for Comments: 5740
Obsoletes: 3940
Category: Standards Track

B. Adamson
Naval Research Laboratory
C. Bormann
Universitaet Bremen TZI
M. Handley
University College London
J. Macker
Naval Research Laboratory

November 2009

# NACK-Oriented Reliable Multicast (NORM) Transport Protocol

## Abstract

This document describes the messages and procedures of the Negative-ACKnowledgment (NACK) Oriented Reliable Multicast (NORM) protocol. This protocol can 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 (forward error correction) repair and other IETF Reliable Multicast Transport (RMT) building blocks in its design. This document obsoletes RFC 3940.

## Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## Copyright Notice

Table of Contents

## 1. Introduction and Applicability

The Negative-ACKnowledgment (NACK) Oriented Reliable Multicast (NORM) protocol can provide reliable transport of data from one or more senders 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 self-adapting to a wide range of dynamic network conditions with little or no pre-configuration. The protocol is tolerant of inaccurate timing estimations or lossy conditions that can occur in many networks including mobile and wireless. The protocol can also converge and maintain efficient operation even in situations of heavy packet loss and large queuing or transmission delays. This document obsoletes the Experimental RFC 3940 specification.

This document is a product of the IETF RMT working group and follows the guidelines provided in the Author Guidelines for Reliable Multicast Transport (RMT) Building Blocks and Protocol Instantiation documents [RFC3269].

Statement of Intent

This memo contains the definitions necessary to fully specify a Reliable Multicast Transport protocol in accordance with the criteria of IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols [RFC2357]. The NORM specification described in this document was previously published in the Experimental Category [RFC3940]. It was the stated intent of the RMT working group to re-submit this specifications as an IETF Proposed Standard in due course. This Proposed Standard specification is thus based on RFC 3940 and has been updated according to accumulated experience and growing protocol maturity since the publication of RFC 3940. Said experience applies both to this specification itself and to congestion control strategies related to the use of this specification. The differences between RFC 3940 and this document are listed in Section 10.

### 1.1. Requirements Language

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 RFC 2119 [RFC2119].

### 1.2. NORM Data 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 chosen via means outside the context of the given NormSession. Other existing IETF data format and protocol standards MAY be applied to describe and convey the necessary a priori information for a specific NormSession (e.g., Session Description Protocol (SDP) [RFC4566], Session Announcement Protocol (SAP) [RFC2974], 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 independently of one another and receivers will maintain state as necessary for each sender. In future versions of NORM, it is possible some aspects of protocol operation (e.g., round-trip time collection) will 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 to allocate 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 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 its message headers. Note the NORM_INFO out-of-band data mechanism can be leveraged by the application for this purpose if desired, or identification can 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. Participants, including senders, in NORM protocol sessions are also identified with unique identifiers (NormNodeIds). Each sender maintains its NormTransportId assignments independently and thus individual NormObjects can be uniquely identified during transport by concatenation of the session-unique sender identifier (NormNodeId) and the assigned NormTransportId. The NormTransportIds are assigned from a large, but fixed, numeric space in increasing order and will be reassigned during long-lived sessions. The NORM protocol provides mechanisms so the sender application can 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 reliable multicast application variants can realize 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.

### 1.3. NORM Scalability

Group communication scalability requirements lead to adaptation of NACK-based protocol schemes when feedback for reliability is needed [RmComparison]. 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 [RFC3453]. FEC-based repair can be used to greatly reduce the quantity of reliable multicast repair requests and repair transmissions [MdpToolkit] 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 [McastFeedback]. NORM dynamically measures the group's round-trip 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 relays the feedback state to the group to facilitate feedback suppression. In typical Internet environments, 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 [NormFeedback]. 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 for NORM to scale to larger group sizes. With respect to computer resource usage, the

NORM protocol does not need state to be kept on all receivers in the group. NORM senders maintain state only for receivers providing explicit congestion control feedback. However, NORM receivers need to maintain state for each active sender. This can constrain the number of simultaneous senders in some uses of NORM.

### 1.4. Environmental Requirements and Considerations

All of the environmental requirements and considerations that apply to the "Multicast Negative-Acknowledgment (NACK) Building Blocks" [RFC5401], "Forward Error Correction (FEC) Building Block" [RFC5052], and "TCP-Friendly Multicast Congestion Control (TFMCC) Protocol Specification" [RFC4654] 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 "Host Extensions for IP Multicasting" [RFC1112], but it SHALL also be capable of scalable operation in asymmetric topologies such as Source-Specific Multicast (SSM) [RFC4607] where only unicast routing service is available from the receivers to the sender(s).

NORM is compatible with IPv4 and IPv6. Additionally, NORM can be used with networks employing Network Address Translation (NAT) provided that 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 multiple possible senders and to distinguish feedback information from different receivers. There are two reserved NormNodeId values. A value of 0x00000000 is considered an invalid NormNodeId (NORM_NODE_NONE), and a value of 0xffffffff is a "wild card" NormNodeId (NORM_NODE_ANY).

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 provided by TCP for unicast data transport. The format of the stream content is application-defined and can 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, a 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 of 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 needs to receive a sufficient number of symbols to reconstruct (via FEC decoding) the original user data for the given block.

Transmitted NormObjects are temporarily, yet uniquely, identified within the NormSession context using the given sender's NormNodeId, NormInstanceId, and a temporary NormTransportId. Depending upon the

implementation, individual NORM senders can manage their NormInstanceIds independently, or a common NormInstanceId could be agreed upon for all participating nodes within a session, if needed, as a session identifier. NORM NormTransportId 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 NormTransportId field can wrap and previously used identifiers will be re-used. Note that globally unique identification of transported data content is not provided by NORM and, if necessary, is expected to be managed by the NORM application. The individual segments or symbols of the NormObject are further identified with FEC payload identifiers that 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 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 configuration is sensible for certain network conditions and can allow for robust, asymmetric multicast (e.g., unidirectional routing, satellite, cable) [FecHybrid] 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. The NORM_INFO message can 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, group round-trip time (GRTT) 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: 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.). A notable distinction between NORM_DATA message and some NORM_CMD message transmissions is that typically a receiver will need to allocate resources to manage reliable reception when NORM_DATA messages are received. However, some NORM_CMD messages are completely atomic and no specific reliability (buffering) state needs to be kept. Thus, for session management or other purposes, it is possible that even participants acting principally as data receivers MAY transmit NORM_CMD messages. However, it is RECOMMENDED that this is not done within the context of the NORM multicast session unless congestion control is addressed. For example, many receiver nodes transmitting NORM_CMD messages simultaneously can cause congestion for the destination(s).

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 is 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 are available for use.

## 2.2. Protocol Building Blocks

The operation of the NORM protocol is based primarily upon the concepts presented in the Multicast NACK Building Block [RFC5401] document. This includes the basic NORM architecture and the data transmission, repair, and feedback strategies discussed in that document. The reliable multicast building block approach, as described in "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer" [RFC3048], is applied in creating the full NORM protocol instantiation. NORM also makes use of the parity-based encoding techniques for repair messaging and added transmission robustness as described in "The Use of Forward Error Correction (FEC) in Reliable Multicast" [RFC3453]. NORM uses the FEC Payload ID as specified by the FEC Building Block document [RFC5052]. Additionally, for congestion control, this document fully specifies a baseline congestion control mechanism (NORM-CC) based on the TCP-Friendly Multicast Congestion Control (TFMCC) scheme [TfmccPaper], [RFC4654].

## 2.3. Design Trade-Offs

While the various features of NORM 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 trade-offs involved in reliable multicast transport design and this necessitates 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 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 trade-offs 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 RMT Protocol Instantiation document, in conjunction with the "Multicast Negative-Acknowledgment (NACK) Building Blocks" [RFC5401] and "Forward Error Correction (FEC) Building Block" [RFC5052] Building Blocks, completely specifies a working reliable multicast transport protocol that conforms to the requirements described in RFC 2357.

This document specifies the following message types and mechanisms that 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 is 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 that 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

There are two primary classes of NORM messages (see Section 2.1): 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. Sender messages SHALL be governed by congestion control for Internet use. For session management or other purposes, receivers can also employ NORM_CMD message transmissions. The principal rationale for distinguishing sender and receiver messages is that receivers will typically need to allocate resources to support reliable reception from sender(s) and NORM sender messages are subject to congestion control. NORM receivers MAY employ the NORM_CMD message type for application-defined purposes, but it is RECOMMENDED that congestion control and feedback implosion issues be addressed. Additionally, 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 compatible with the Maximum Transmission Unit (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:

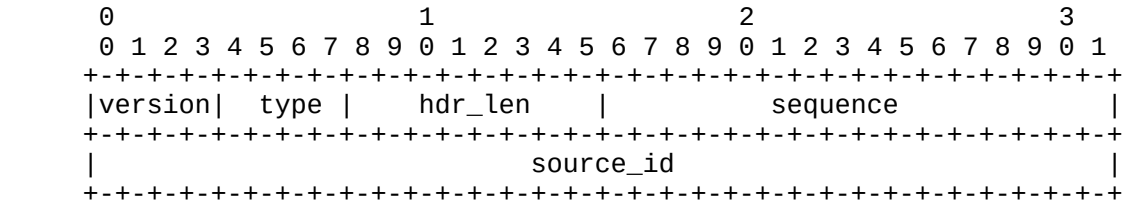



Figure 1: NORM Common Message Header Format

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 that are 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 the addition of header extensions. The presence of header extensions is 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. The "sequence" field serves two separate purposes, depending upon the message type:

1. NORM senders MUST set the "sequence" field of sender messages (NORM_INFO, NORM_DATA, and NORM_CMD) so that receivers can monitor the "sequence" value to maintain an estimate of packet loss that can be used for congestion control purposes (see Section 5.5.2 for a detailed description of NORM Congestion Control operation). A monotonically increasing sequence number space MUST be maintained to mark NORM sender messages in this way. Note that this "sequence" number is explicitly NOT used in

NORM as part of its reliability procedures. The NORM object and FEC payload identifiers are used to detect missing content for reliable transfer purposes.

2. NORM receivers SHOULD set the "sequence" field to support protection from message replay attacks of NORM_NACK or NORM_NACK messages. Note that, depending upon configuration, NORM feedback messages are sent to the session multicast address or the unicast address(es) of the active NORM sender(s). Thus, a separate, monotonically increasing sequence number space MUST be maintained for each destination address to which the NORM receiver is transmitting feedback messages.

Note that these two separate purposes necessitate the maintenance of separate sequence spaces to support the functions described here. And, in the case of NORM receivers, additional sequence spaces are needed when feedback messages are sent to the sender unicast address(es) instead of the session address.

The "source_id" field is a 32-bit value that uniquely identifies the node that sent the message within the context of a single NormSession. This value is termed the NORM node identifier (NormNodeId) and unique NormNodeIds MUST be assigned within a single NormSession. In some cases, use of the host IPv4 address or a hash of an address can suffice, but alternative methodologies for assignment and potential collision resolution of node identifiers within a multicast session SHOULD be considered. For example, the techniques for managing the 32-bit "synchronization source" (SSRC) identifiers defined in the Real-Time Protocol (RTP) specification [RFC3550] are applicable for use with NORM node identifiers when an ASM traffic model is observed. In most deployments of the NORM protocol to date, the NormNodeId assignments are administratively configured, and this form of NormNodeId assignment is RECOMMENDED for most purposes. NORM sender NormNodeId values MUST be unique within an ASM session so that NORM receiver feedback can be properly demultiplexed by senders, and NORM receiver NormNodeId values MUST also be unique for congestion control operation or when the OPTIONAL positive acknowledgment mechanism is used.

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.

For variable-length extensions, the value of the "hel" (header extension length) field is the length of the entire header extension, expressed in multiples of 32-bit words. The "hel" field MUST be present for variable-length extensions ("het" between 0 and 127) and MUST NOT be present for fixed-length extensions ("het" between 128 and 255).

The formats of the variable-length and fixed-length header extensions are given, respectively, here:

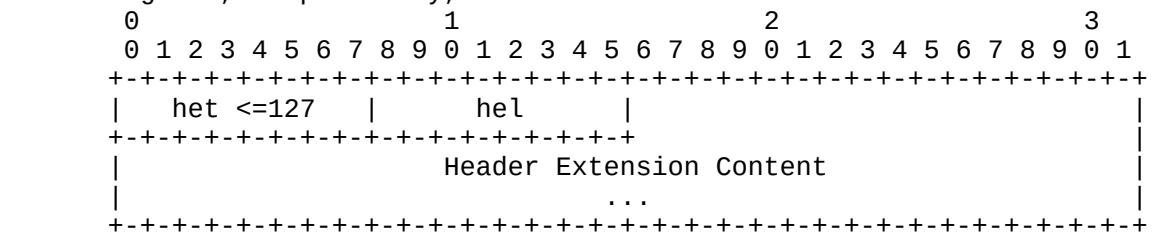



Figure 2: NORM Variable-Length Header Extension Format

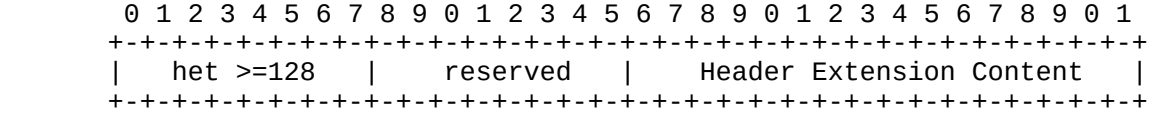



Figure 3: NORM Fixed-Length (32-bit) Header Extension Format

The "Header Extension Content" portion of the header extension is defined for each extension type. 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 generally 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 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.

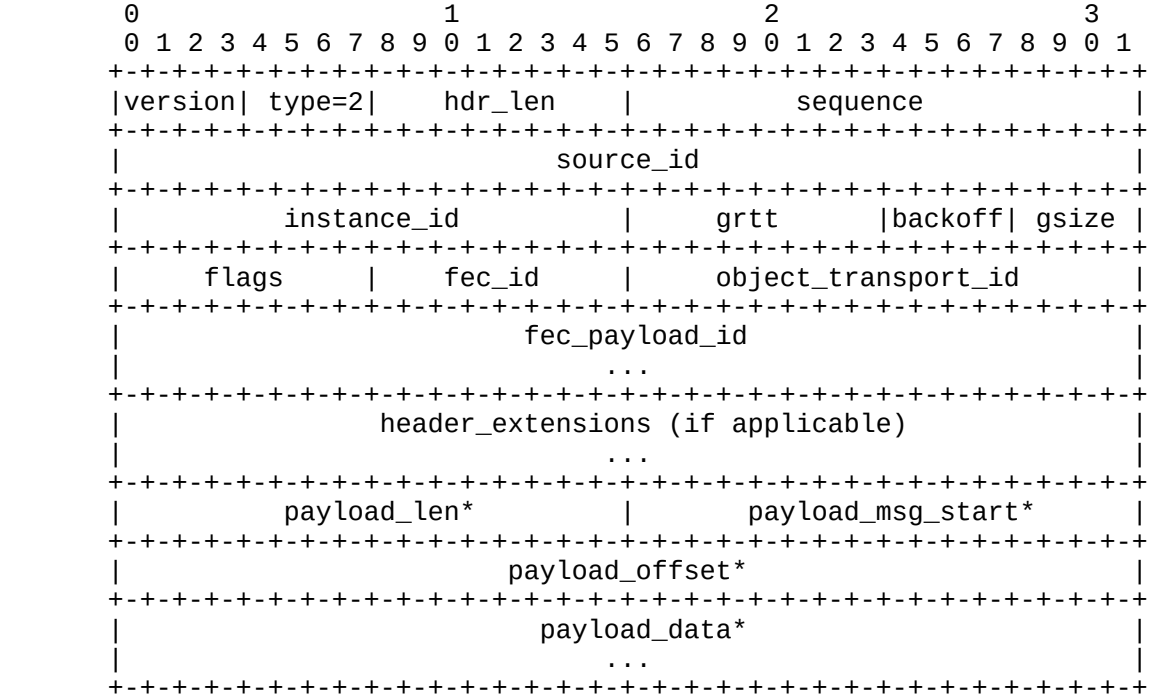



Figure 4: NORM_DATA Message Format

*IMPORTANT NOTE: The "payload_len", "payload_msg_start" and "payload_offset" fields are present only for objects of type NORM_OBJECT_STREAM. These fields, as with the entire payload, are subject to any FEC encoding used. Thus, when systematic FEC codes are used, these values can be directly interpreted only for packets containing source symbols while packets containing FEC parity content need decoding before these fields can be interpreted.

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 (i.e., four 32-bit words) plus the size of the "fec_payload_id" field. The "fec_payload_id" field size depends upon the FEC encoding type referenced by the "fec_id" field. 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_sender) (this is also referred to as R_max in [TfmccPaper]). 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" SHALL be set to MAX(grttMeasured, NormSegmentSize/senderRate) where the NormSegmentSize is the 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 Multicast NACK Building Block [RFC5401] document.

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 that are multiplied by GRTT_sender to determine the maximum backoff timeout. The "backoff" field informs the receivers of the sender's backoff factor parameter (K_sender). Recommended values and their uses 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 (GSIZE_sender). 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 example, 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" = 0x3) 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 for the receiver to appropriately 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 a 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" (not previously transmitted) 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. There are cases where receivers can 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 SHALL 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 [RFC5052]. 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 systematic FEC codes are RECOMMENDED for the most efficient performance of NORM_OBJECT_STREAM transport.

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 will wrap and be repeated, but it is presumed that the 16-bit field size provides a sufficient 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 such as those described in the FEC Basic Schemes [RFC5445] specification or in other FEC Schemes. As an example, the format of the "fec_payload_id" format for Small Block, Systematic codes ("fec_id" = 129) from the FEC Basic Schemes [RFC5445] specification is given here:

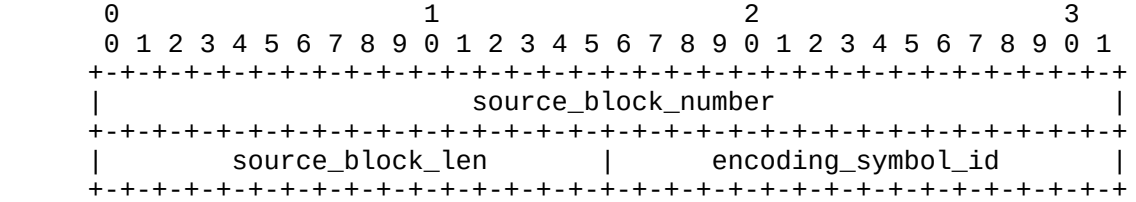



Figure 5: Example: FEC Payload Id Format for 'fec_id' = 129

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 for Small Block Systematic FEC Schemes identified by a "fec_id" value of 129 as specified by the FEC Basic Schemes [RFC5445] specification. 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" will 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. Applications MAY dynamically "shorten" code blocks when the pending information content is not predictable (e.g., real-time message streams). In that case, the "source_block_len" value given for an "encoding_symbol_id" that contains FEC parity content SHALL take precedence over the "source_block_len" value provided for any packets containing source symbols. Also, the "source_block_len" value given for an ordinally higher "encoding_symbol_id" SHALL take precedence over the "source_block_len" given for prior encoding symbols. The reason for this is that the sender will only know the maximum source block length at the time it is transmitting source symbols, but then subsequently "shorten" the code and then provide that last source symbol and/or encoding symbols with FEC parity content. 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 will be a user data or a 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 (FTI) (as described in the FEC Building Block [RFC5052]) document is needed to properly receive and decode NORM transport objects. This information MAY be provided as out-of-band session information. In some cases, it will be useful for the sender to include this information "in-band" to facilitate receiver operation with minimal pre-configuration. 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.

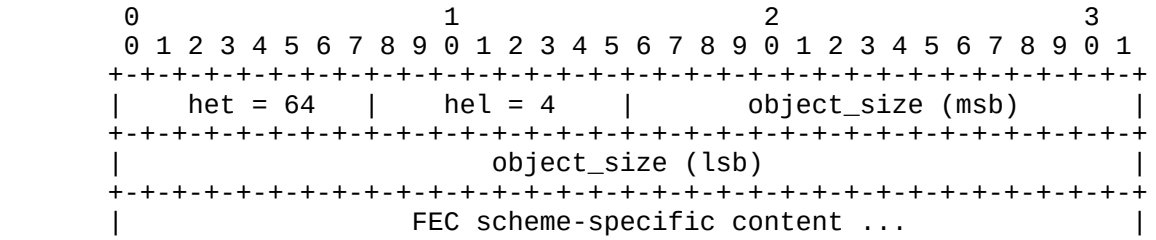



Figure 6: EXT_FTI Header Extension General Portion Format

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 encoding 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 might 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 contains any necessary details on the 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:

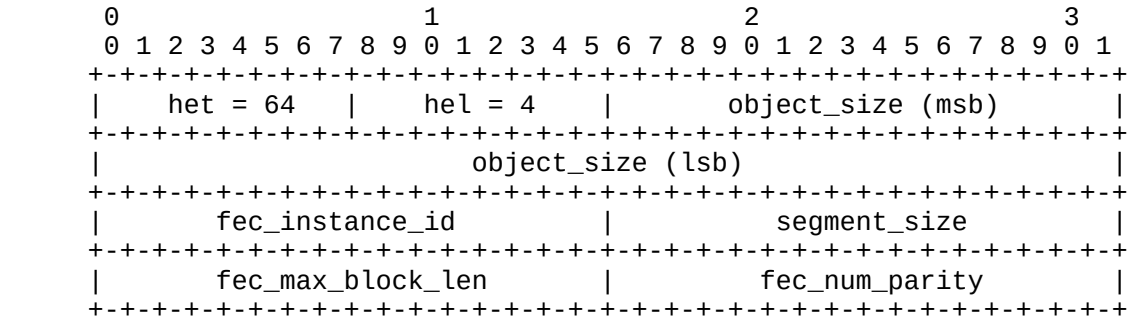



Figure 7: Example: EXT_FTI Header Extension Format for 'fec_id' = 129

In this example (for "fec_id" = 129), the "hel" field value is 4. The size of the EXT_FTI header extension will possibly 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 [RFC5052] document. 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 RFC 5052.

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 FEC Object Transmission Information for Small Block Systematic Codes as described in the FEC Building Block [RFC5052] document. For example, Reed-Solomon codes can 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. The content of this payload depends upon the FEC scheme being employed, and support for streaming using the NORM_OBJECT_STREAM type, when applicable, necessitates some additional content in the payload.

The "payload_len", "payload_msg_start", and "payload_offset" fields are present only for transport objects of type NORM_OBJECT_STREAM. These REQUIRED 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 REQUIRED block partitioning algorithm described in the FEC Building Block [RFC5052] document. When systematic FEC codes (e.g., "fec_id" = 129) are used, the "payload_len", "payload_msg_start", and "payload_offset" fields contain actual payload_data length, message start index (or stream control code), and byte offset values for the associated application stream data segment (the remainder of the "payload_data" field content) for those NORM_DATA messages containing source data symbols. In NORM_DATA messages that contain FEC parity content, these fields do not contain values that can be directly interpreted, but instead are values computed from FEC encoding the "payload_len", "payload_msg_start", and "payload_offset" fields for the source data segments of the corresponding coding block. 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_len" value, when non-zero, indicates the length (in bytes) of the source content contained in the associated "payload_data" field. However, when the "payload_len" value is equal to ZERO, this indicates that the "payload_msg_start" field be

alternatively interpreted as a "stream_control_code". The only "stream_control_code" value defined is NORM_STREAM_END = 0. The NORM_STREAM_END code indicates that the sender is terminating the transmission of stream content at the corresponding position in the stream and the receiver MUST NOT expect content (or request repair for any content) following that position in the stream. Additional specifications MAY extend the functionality of the NORM stream transport mode by defining additional stream control codes. These control codes are delivered to the recipient application reliably, in-order with respect to the streamed application data content.

The "payload_msg_start" field serves one of two exclusive purposes. When the "payload_len" value is non-zero, the "payload_msg_start" field, when also 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_DATA payload for a NORM_OBJECT_STREAM 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'. NORM implementations SHOULD provide sender stream applications with a capability to mark message boundaries in this manner. Similarly, the NORM receiver implementation SHOULD enable the application to recover such message boundary information. This enables NORM receivers to "synchronize" reliable reception of transmitted message stream content in a meaningful way (i.e., meaningful to the application) at any time, whether joining a session already in progress, or departing the session and returning. Note that if the value of the "payload_msg_start" field is ZERO, no message boundary is present. The "payload_msg_start" value will always be less than or equal to the "payload_len" value except for the special case of "payload_len = 0", which indicates the "payload_msg_start" field be instead interpreted as a "stream_control_code"

The "payload_offset" field indicates the relative byte position (from the sender stream transmission start) of the source content contained in the "payload_data" field. Note that for long-lived streams, the "payload_offset" field will 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 a 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 can possibly vary on a per-object basis. The NormSegmentSize SHOULD be configurable by the sender application prior to session participation as needed for network topology 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 partitioning algorithm described in the FEC Building Block [RFC5052] document. For NORM_OBJECT_STREAM objects, the length and offset is obtained from the segment's corresponding embedded "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 could then use the NORM_INFO content to make a decision as to whether to participate in reliable reception of the associated object. Each NormObject can have an independent unit of NORM_INFO with which it is associated. NORM_DATA messages contain a flag to indicate the availability of NORM_INFO for a given NormObject. NORM receivers will 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.

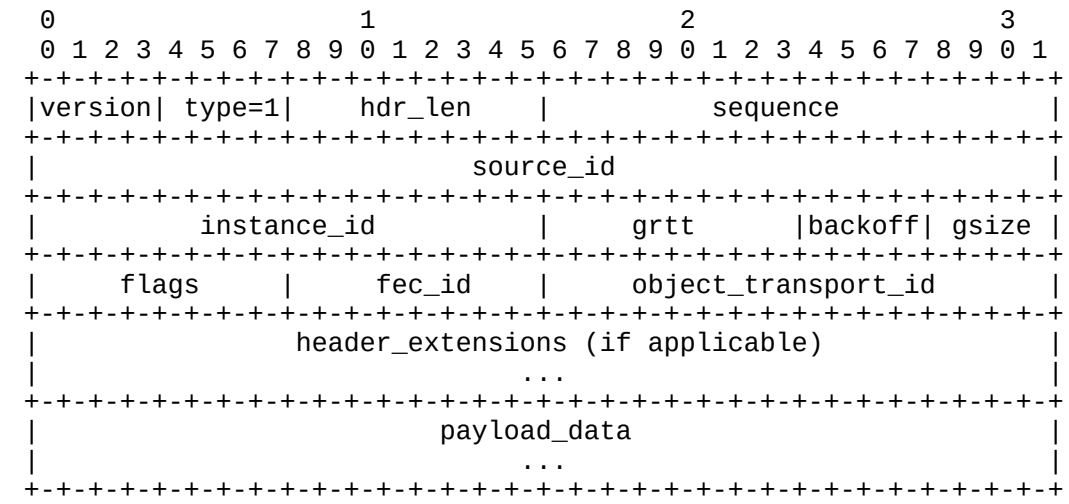



Figure 8: NORM_INFO Message Format

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 the "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 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, NORM implementations MAY 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 that 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 can have dynamic content attached. Any attached content will be limited to the maximum length of the sender NormSegmentSize to retain the atomic nature of the 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:

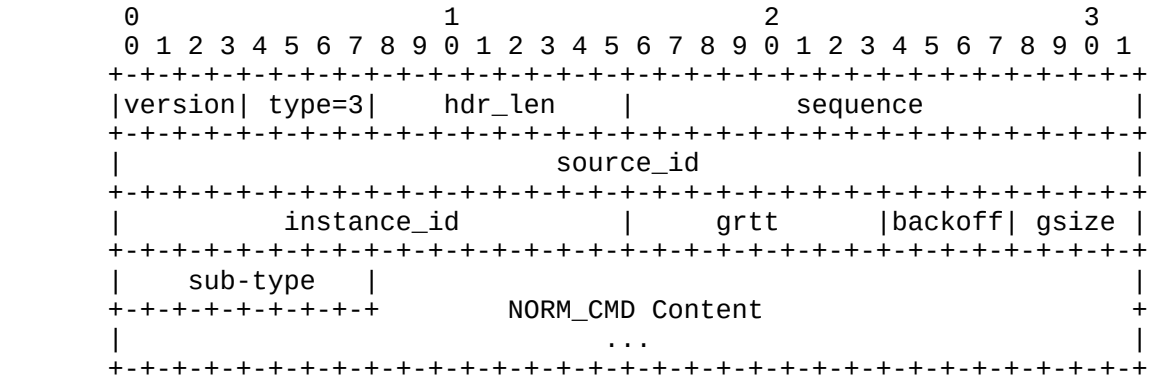



Figure 9: NORM_CMD Standard Fields

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 "sub-type" field.

The "instance_id", "grtt", "backoff", and "gsize" fields provide the same information and serve the same purpose as NORM_DATA and NORM_INFO messages. The "sub-type" field indicates the type of command to follow. The remainder of the NORM_CMD message is dependent upon the command sub-type. NORM command sub-types include:

| Command | Sub-type | 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 a 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 that need to temporarily preempt or supplement 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 can indicate either a temporary or permanent end-of-data transmission, but that the sender is still willing to respond to repair requests. This command is repeated once per 2*GRTT_sender 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 SHALL 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 the NORM_CMD(FLUSH) and NORM_ROBUST_FACTOR and is specified in Section 5.3. Receivers SHALL self-initiate the NACK repair process when the inactivity timeout has expired for a specific sender and the receiver has pending repairs needed from that sender. With a sufficiently large NORM_ROBUST_FACTOR value, data content is delivered with a high assurance of reliability. The penalty of a large NORM_ROBUST_FACTOR value is the potential transmission of excess NORM_CMD(FLUSH) messages and a longer inactivity timeout for receivers to self-initiate a terminal NACK process.

For finite-sized 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 can occur at any time, including in the middle of a FEC coding block if systematic FEC codes are employed. In this case, the sender will not yet be able to provide FEC parity content for the concurrent coding block and will be limited to explicitly repairing the stream with source 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 ongoing series of intermittent, relatively small messages 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.

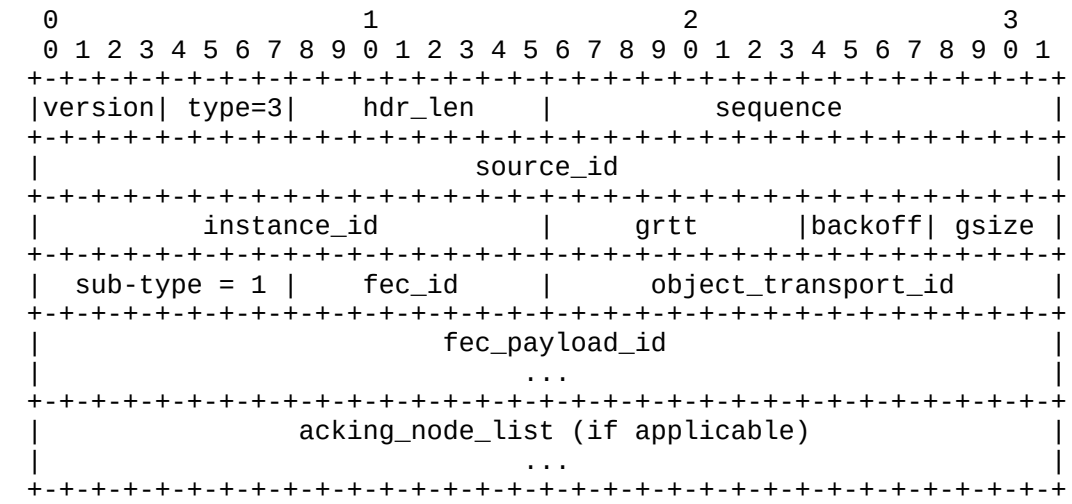



Figure 10: NORM_CMD(FLUSH) Message Format

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" that 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 a 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. Note that a sender has the option of arbitrarily shortening a given code block when such an application "flush" occurs. In this case, the receiver will request explicit repair, but the sender MAY provide FEC-based repair (parity segments) in response. These parity segments MUST contain the corrected "source_block_len" for the shortened block and that "source_block_len" associated with segments containing parity content SHALL override the previously advertised "source_block_len". Similarly, the "source_block_len" associated with the highest ordinal "encoding_symbol_id" SHALL take precedence over prior symbols when a difference (e.g., due to code shortening at the sender) occurs. 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.

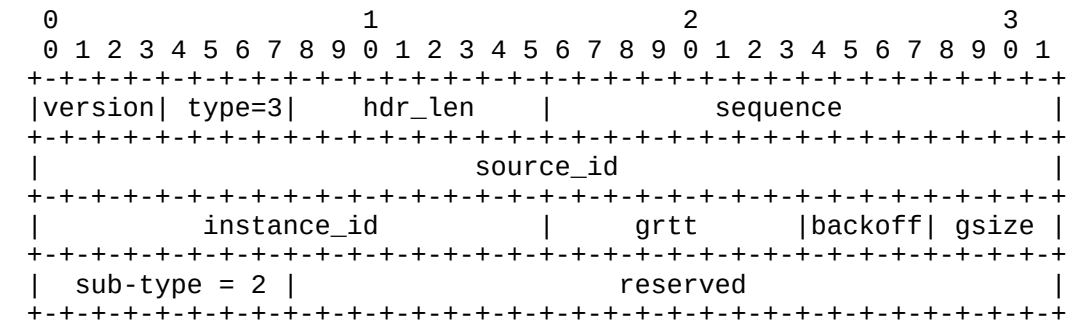



Figure 11: NORM_CMD(EOT) Message Format

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 that 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_sender 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, 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:

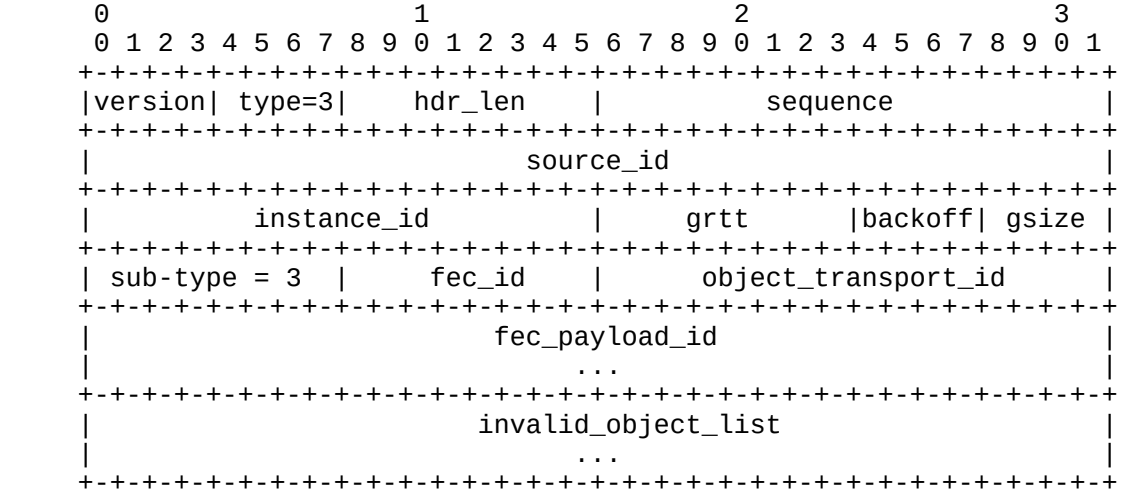



Figure 12: NORM_CMD(SQUELCH) Message Format

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 that initiated the NORM_CMD(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 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 cannot include the entire set of invalid objects in the repair window. In this case, the sender SHALL ensure that the list begins with a NormTransportId that is greater than or equal to the lowest ordinal invalid NormTransportId from the NACK message(s) that prompted the NORM_CMD(SQUELCH) generation. The NormTransportId in the "invalid_object_list" MUST be ordinally greater than the "object_transport_id" marking the beginning of the sender's repair window. This ensures 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 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) message contains fields to enable sender-to-group 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 RFC 4654 is fully specified in Section 5.5.2 of this document. The NORM_CMD(CC) message is usually transmitted as part of NORM-CC 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 network where resources are explicitly dedicated to the NORM session and therefore congestion control operation is disabled, the NORM_CMD(CC) message is then used solely for GRTT measurement and MAY be sent less frequently than with congestion control operation.

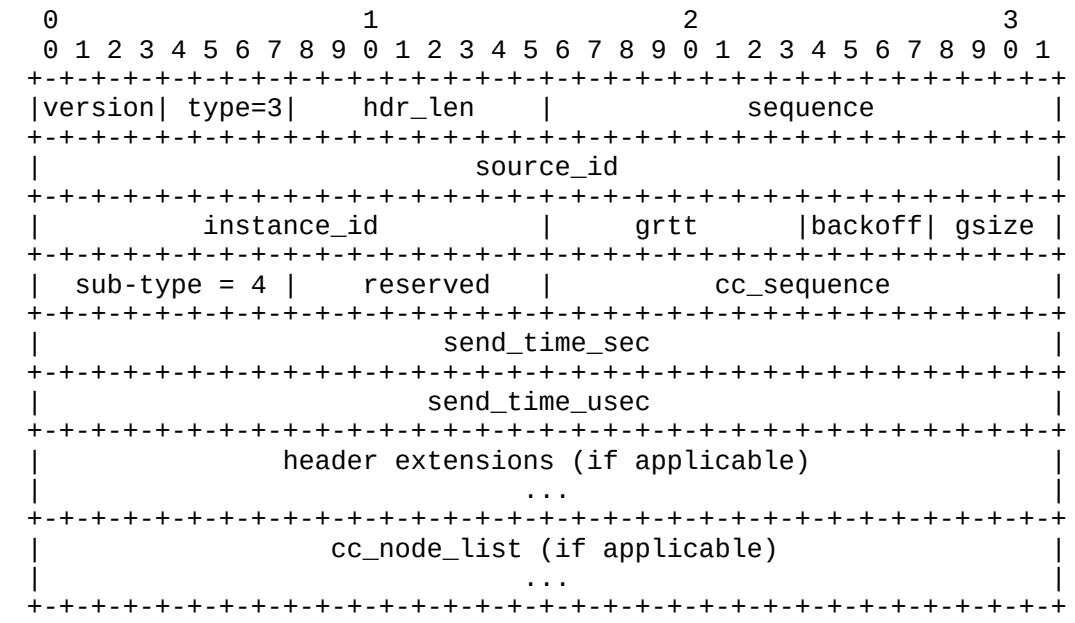



Figure 13: NORM_CMD(CC) Message Format

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.

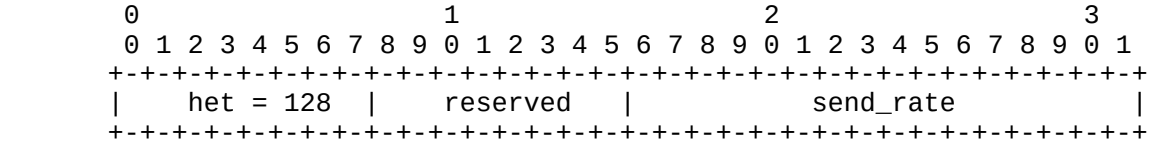
The "reserved" field is for potential future use and MUST be set to ZERO in this version of the NORM protocol and its baseline NORM-CC congestion control scheme. It is possible for alternative congestion control schemes to use the NORM_CMD(CC) message defined here and leverage the "reserved" field for 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 RFC 4654. 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:




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 integer exponent (E) information in the least significant portion. The 12-bit mantissa portion of the field is scaled such that a base 10 mantissa (M) floating point value of 0.0 corresponds to 0 and a value of 10.0 corresponds to 4096 in the upper 12 bits of the 16-bit "send_rate" field. Thus:

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

For example, to represent a transmission rate of 256 kbit/s (3.2e+04 bytes per second), the lower 4 bits of the 16-bit field contain a value of 0x04 to represent the exponent (E) while the upper 12 bits contain a value of 0x51f (M) 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) * (10/4096) * power(10, (send_rate & x000f))
```

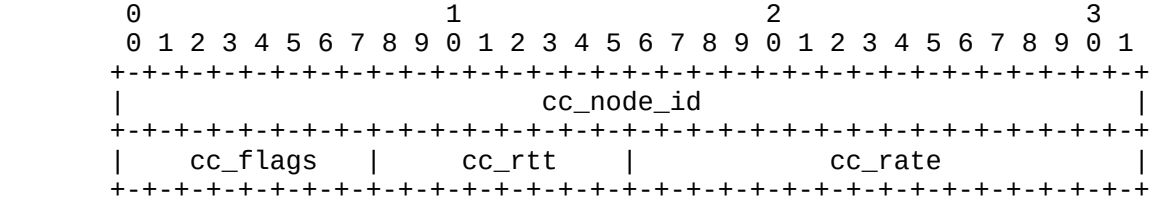
Note the maximum transmission rate that can be represented by this

scheme is approximately 9.99e+15 bytes per second.

When this extension is present, a "cc_node_list" might be attached as the payload of the NORM_CMD(CC) message. The presence of this header extension also implies that NORM receivers MUST 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 can be increased 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:




The "cc_node_id" is the NormNodeId of the receiver 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 Multicast NACK Building Block [RFC5401] document.

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 relaying 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 [NormFeedback].

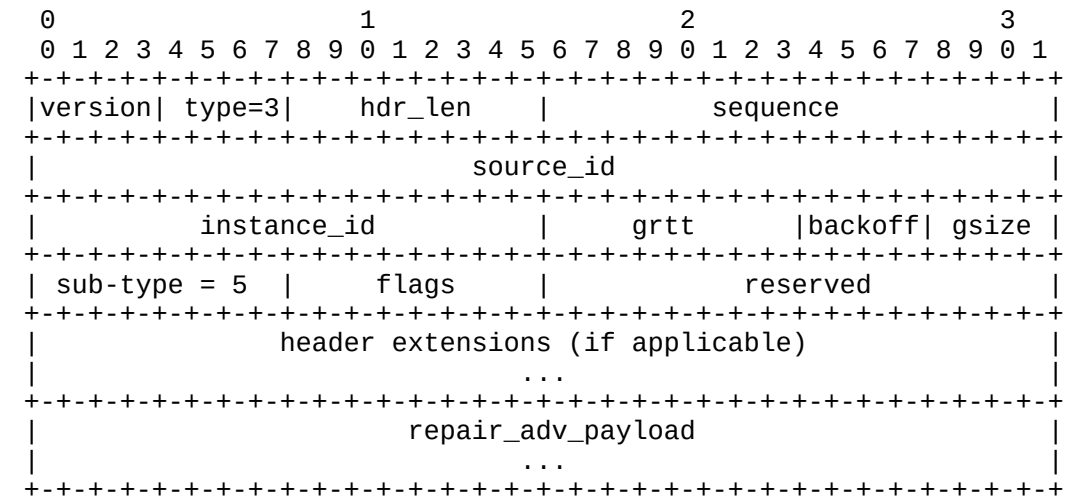



Figure 14: NORM_CMD(REPAIR_ADV) Message Format

The "instance_id", "grtt", "backoff", "gsize", and "sub-type" 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.

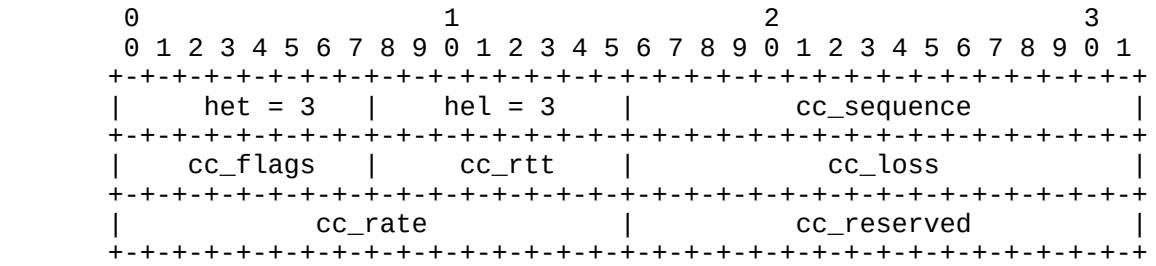
The "flags" field provides information on the NORM_CMD(REPAIR_ADV) content. There is currently one NORM_CMD(REPAIR_ADV) flag defined:

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 SHALL limit their NACK response to generating NACK content only up through the maximum ordinal transmission position (objectTransportId::fecPayloadId) included in the "repair_adv_content".

When congestion control operation is enabled, a header extension SHOULD 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 can be used for 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) [PgmccPaper]) is used

within a NORM implementation, an additional header extension MAY need to be defined to encapsulate any required feedback content. The NORM-CC Feedback Header Extension format is:




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 SHOULD 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 SHALL 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 SHALL be set only when no feedback has been received from non-CLR or non-PLR receivers. And the NORM_FLAG_CC_LEAVE SHALL 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" = floor(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 receiver's 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:

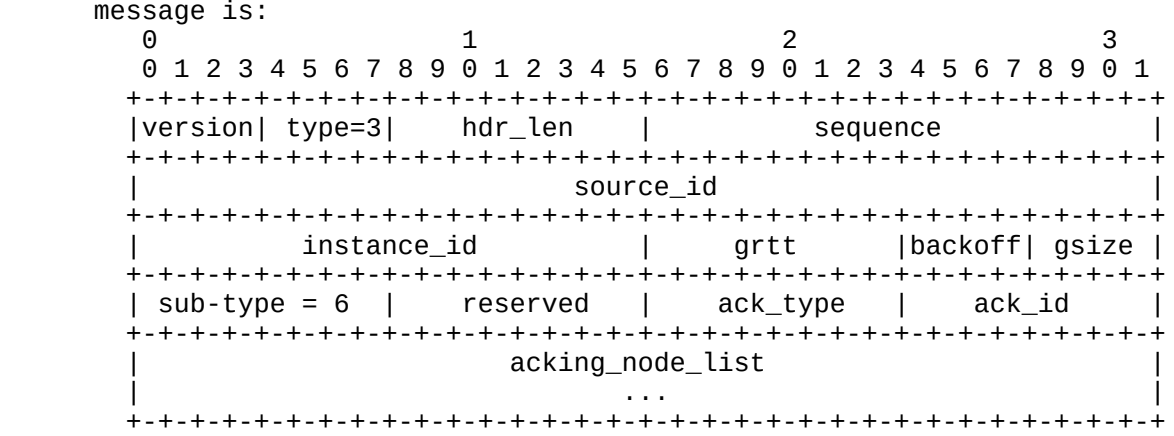



Figure 15: NORM_CMD(ACK_REQ) Message Format

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 can 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 can 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 acknowledgment (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_sender. 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 can include initiation of data transmission, other NORM_CMD(APPLICATION) messages, or even application-defined, positively acknowledged commands from other NormSession participants. The transmission of these commands will preempt data transmission when they are scheduled and can 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 needs while taking advantage of NORM transmission and round-trip timing information.

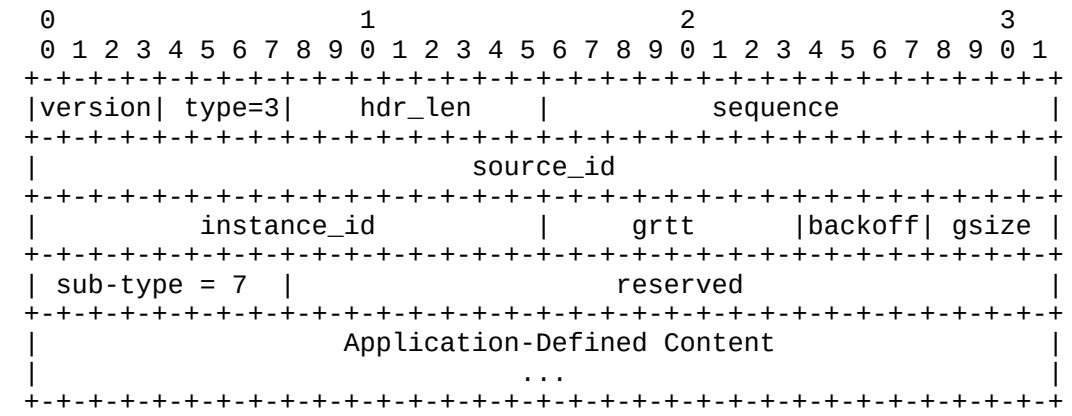



Figure 16: NORM_CMD(APPLICATION) Message Format

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. Upon reception, the NORM protocol implementation SHALL deliver the content to the receiver application. Note that any detection of duplicate reception of a NORM_CMD(APPLICATION) message is the responsibility of the application.

## 4.3. Receiver Messages

The NORM message types generated by participating receivers consist of the 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:

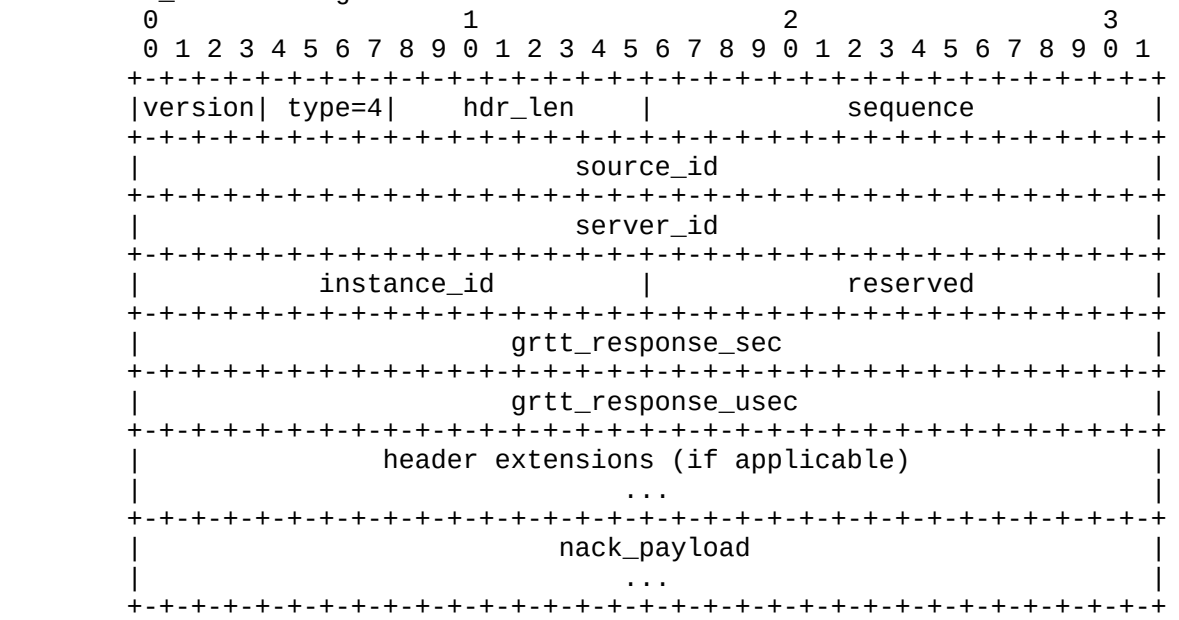



Figure 17: NORM_NACK Message Format

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 containing 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 delta from when the receiver received the NORM_CMD(CC) to when the NORM_NACK is transmitted in response to calculate the value in the "grtt_response" field. This is the "receive_to_response_delta"

value used in the following formula:
grtt_response = NORM_CMD(CC) send_time + receive_to_response_delta

The receiver SHALL set the "grtt_response" to a ZERO value, to indicate it has not yet received a NORM_CMD(CC) message from the indicated sender, and the sender MUST 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 receiver's current state with respect to congestion control operation. 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_payload" 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 needs from the sender 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 can be concatenated within the "nack_payload" field of a NORM_NACK message. 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:

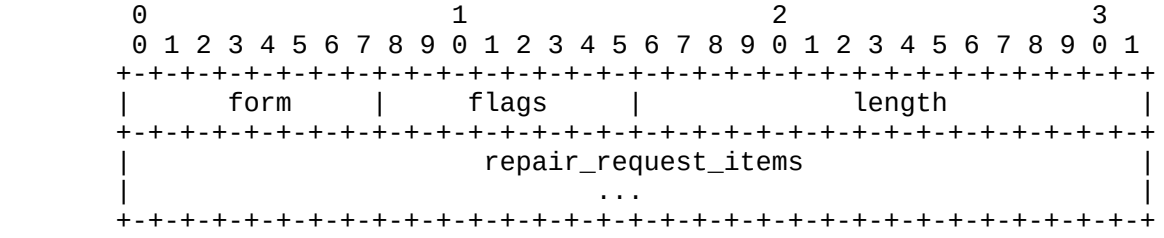



Figure 18: NORM Repair Request Format

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 the "repair_request_items" list consists of pairs of repair request items corresponding to the inclusive ranges of repair needs. The NORM_NACK_ERASURES "form" indicates the repair request items are to be treated individually and 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 needed as repair. |
| NORM_NACK_BLOCK | 0x02 | Indicates the listed block(s) or range of blocks in entirety that are needed as repair. |
| NORM_NACK_INFO | 0x04 | Indicates NORM_INFO is needed as repair for the listed object(s). |
| NORM_NACK_OBJECT | 0x08 | Indicates the listed object(s) or range of objects in entirety are needed 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 that transmissions sufficient to repair the indicated block(s) in their entirety are needed. 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 can be set

in combination with the NORM_NACK_BLOCK or NORM_NACK_SEGMENT flags, or can 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.

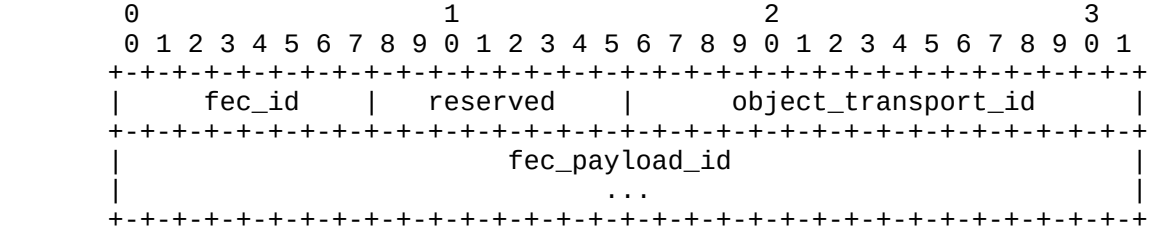



Figure 19: NORM Repair Request Item Format

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.

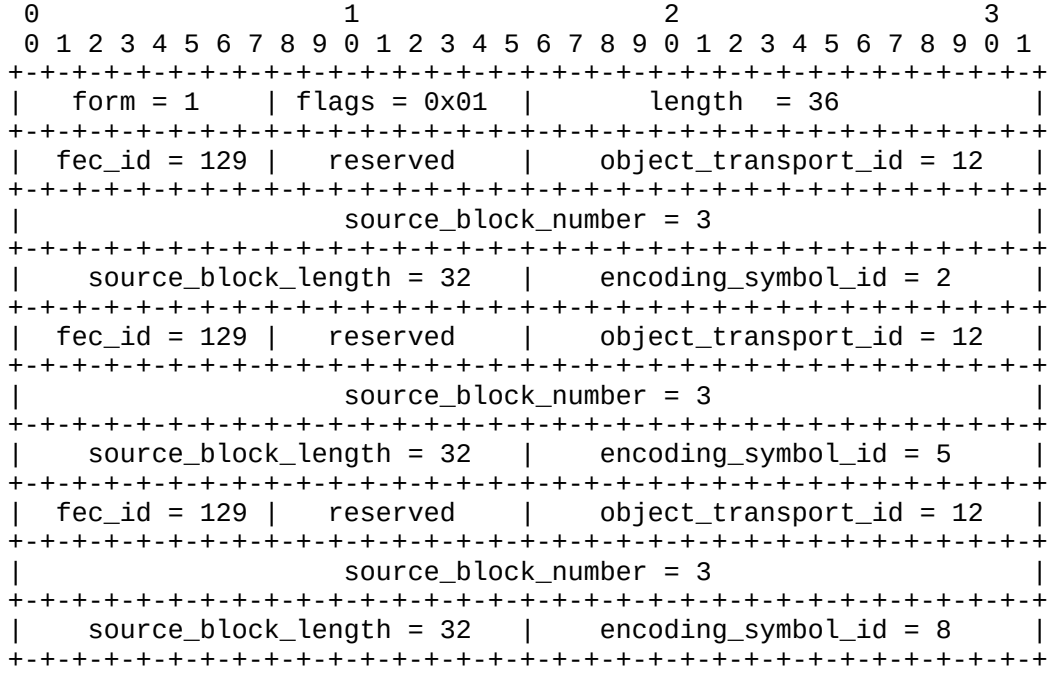
When the receiver’s repair needs dictate that different forms (mixed ranges and/or individual items) or types (mixed specific segments and/or blocks or objects in entirety) are needed to complete reliable transmission, multiple NORM Repair Requests with different "form" and or "flags" values can be concatenated 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.

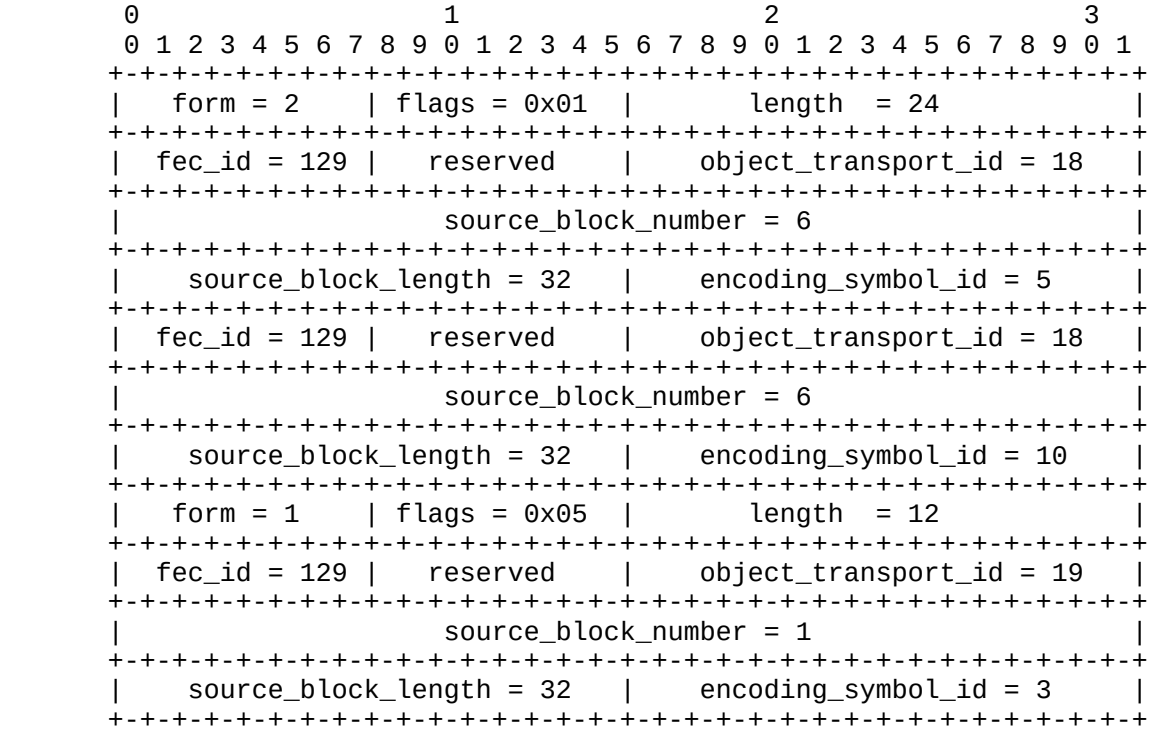
NORM_NACK Content Examples:

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 can 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, and 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




### 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. The acknowledgment type NORM_ACK(CC) is provided for this purpose as described in the NORM_CMD(ACK_REQ) message description. The generation of NORM_ACK(CC) messages for round-trip timing estimation and congestion control operation is described in Section 5.5.1 and Section 5.5.2, respectively. However, some multicast applications can benefit from some limited form of positive acknowledgment for certain functions. A simple, scalable positive acknowledgment scheme is defined in Section 5.5.3, which can be leveraged by protocol implementations when appropriate. The NORM_CMD(FLUSH) can also 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 of 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.

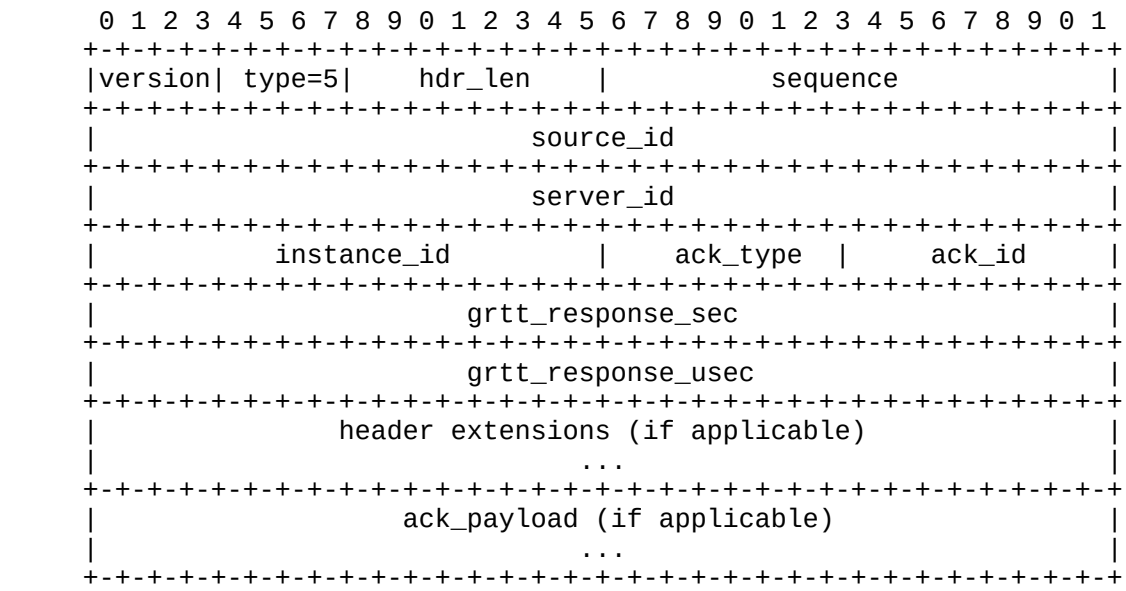



Figure 20: NORM_ACK Message Format

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.

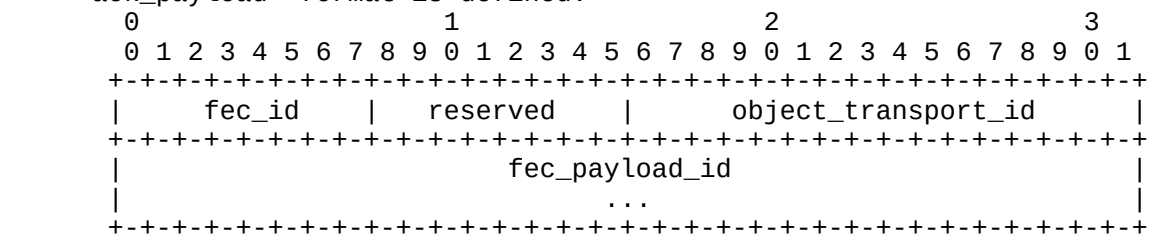
The "server_id", "instance_id", and "grtt_response" fields serve the same purpose as the corresponding fields in NORM_NACK messages. Header extensions can be applied to support congestion control feedback or other functions in the same manner.

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 the sender can verify a received NORM_ACK message 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:




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 of 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 can 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 compliant 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 the protocol operation is given here:

1. The sender periodically transmits NORM_CMD(CC) messages as needed to initialize and collect round-trip 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. When applicable, 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. The receivers track the sender's most recent objectTransportId::fecPayloadId transmit position and NACK only for content that is ordinally prior to that current 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. FEC parity content not previously transmitted 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 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 need 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 not operating using dedicated resources, like in the general Internet. Even if congestion control operation is disabled at the sender, it can 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 might need the sender to also proceed with data transmission immediately. In other cases, the sender might wish to defer data transmission until it has received some feedback or request from the receiver set indicating 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 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 [RFC5052] is RECOMMENDED. For objects of type NORM_OBJECT_STREAM, segmentation will typically be into uniform FEC coding block sizes, with individual segment sizes controlled by the application. In most 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_sender. Similar to the end of each transmitted FEC coding block during transmission, receivers SHALL respond to these NORM_CMD(FLUSH) messages with additional repair requests as needed. 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 purpose 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 appropriate NACKs can be constructed to request repair of missing data. NORM FEC coding blocks are comprised of multi-byte symbols (segments) transmitted in the payload of NORM_DATA messages. Each NORM_DATA message will contain one or more source or encoding symbols 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 partitioning algorithm described in the FEC Building Block [RFC5052] document is RECOMMENDED for use in computing 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 that 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

For typical operation, NORM receivers will join a specified multicast group and listen on a specific port number for sender transmissions. As the NORM receiver receives NORM_DATA messages, it will establish buffering state and provide content to its application as appropriate for the given data type. The NORM protocol allows receivers to join and leave the group at will, although some applications might need receivers to be members of the group prior to start of data transmission. Thus, different 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 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" might 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 the join policy constrain receivers to begin reliable reception at the current FEC coding block for which non-repair content is received.

In some deployments, different multicast receivers might have differing quality of network connectivity. Some receivers may suffer significantly poorer performance with very limited goodput due to low connection rate or substantial packet loss. Similar to the "join policies" described above, a NORM sender implementation MAY choose to enforce different "service policies" to perhaps exclude exceptionally poorly performing (or otherwise badly behaving) receivers from the group. The sender implementation could choose to ignore NACKs from such receivers and/or force advancement of its logical "repair window" (i.e., enforcing a minimal level of service) and use the NORM_CMD(SQUELCH) message to advise those poor performers of its advance. Note in some cases, the application may need to support the "weakest member" regardless of the time needed to achieve reliable delivery. When implemented, the protocol instantiation SHOULD expose controls to the set of "join" and/or "service" policies available to support the needs of different applications.

## 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 NORM_DATA or NORM_INFO transmissions are no longer received from a previously active sender. The RECOMMENDED value of such an inactivity timeout is:

T_inactivity = NORM_ROBUST_FACTOR * 2 * GRTT_sender

where the GRTT_sender value corresponds to the GRTT estimate advertised in the "grtt" field of NORM sender messages. A minimum T_inactivity value of 1 second is RECOMMENDED. The NORM receiver SHOULD reset this inactivity timer and repeat NACK initiation upon timeout for up to NORM_ROBUST_FACTOR times or more depending upon the application's need for persistence by its receivers. It is also important receivers rescale the T_inactivity timeout as the sender's advertised GRTT changes.

The NACKing procedure begins with a random backoff timeout. The duration of the backoff timeout is chosen using the "RandomBackoff" algorithm described in the Multicast NACK Building Block [RFC5401] document using (K_sender*GRTT_sender) for the maxTime parameter and the sender advertised group size (GSIZE_sender) as the groupSize parameter. NORM senders provide values for GRTT_sender, K_sender and GSIZE_sender via the "grtt", "backoff", and "gsize" fields of transmitted messages. The GRTT_sender value is determined by the sender based on feedback it has received from the group while the K_sender and GSIZE_sender values can be determined by application requirements and expectations or ancillary information. The backoff factor K_sender MUST be greater than one to provide for effective feedback suppression. A value of K_sender = 4 is RECOMMENDED for the Any Source Multicast (ASM) model, while a value of K_sender = 6 is RECOMMENDED for Single Source Multicast (SSM) operation.

Thus:

T_backoff = RandomBackoff(K_sender*GRTT_sender, GSIZE_sender)

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 (K_sender-1)*GRTT_sender. 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 objectTransportId::fecPayloadId) exceeds the earliest repair position of the receiver.

2. The repair state accumulated from NORM_NACK and NORM_CMD(REPAIR_ADV) messages does 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 re-initiate 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 [RFC5401] is:

T_rcvrHoldoff =(K_sender+2)*GRTT_sender

The NORM_NACK message contains repair request content beginning with the 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 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 the sender might have transmitted 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 needed 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 does not demand 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 principal goal of the sender is to make forward progress in the transmission of data its application has enqueued. However, the sender will need to 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 [RFC5401], the period of time during which the sender aggregates NORM_NACK messages is equal to:

$$T\_sndrAggregate = (K\_sender + 1) * GRTT\_sender$$

where K_sender is the backoff scaling value advertised to the receivers, and GRTT_sender is the sender's current estimate of the group's greatest round-trip time. Note, for NORM unicast sessions, the T_sndrAggregate time can be set to ZERO since there is only one receiver. Similarly, the K_sender value SHOULD be set to ZERO for NORM unicast sessions to minimize repair latency.

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 [RFC5401], the value of this sender "holdoff" period is:

T_sndrHoldoff = (1 * GRTT_sender)

If additional NORM_NACK messages are received during this sender "holdoff" period, the sender will immediately incorporate these late-arriving messages into its pending transmission state if, and 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 started (upon arrival of a NACK) in concert with the pending repair and new data transmission. Recall 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 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. The sender SHOULD NOT resort to explicit transmission of the receiver set's repair needs until after exhausting its supply of "fresh" (unsent) parity segments for a given coding block. 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 continue to operate 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 limiting new receivers from joining the reliable transmission when only repair transmissions have been received. Additionally, the sender SHOULD 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 need 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 NORM_NACK suppression/aggregation as it conducts this repair state accumulation for NORM repair cycles. The details 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_sender. 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. The list includes as many lower ordinal invalid "object_transport_ids" that can fit for the NORM_CMD(SQUELCH) payload size to less than or equal to 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. A NORM sender implementation MAY use a separate port number from the NormSession port number as the source port for its transmissions. Thus, NORM receivers can direct any unicast feedback messages to this separate sender port number, distinct from the NORM session (or destination) port number. Then, the NORM sender implementation can discriminate unicast feedback messages from multicast feedback messages when there is a mix of multicast and unicast feedback receivers. 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 that 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 is subject to the sender transmit rate limit and NormSegmentSize limitation. When the NORM_CMD(REPAIR_ADV) message is of maximum size (as indicated by the flag NORM_REPAIR_ADV_FLAG_LIMIT), 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 will also need to provide any information needed so dynamic congestion control feedback can be suppressed 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 to help NORM to adapt to network conditions and play fairly with other coexistent protocols.

#### 5.5.1. Group Round-Trip Time (GRTT) Collection

For NORM receivers to appropriately scale backoff timeouts and the senders to use proper corresponding timeouts, the participants need

to use a common timeout basis. Each NORM sender monitors the round-trip time of active receivers and determines the greatest group round-trip time. The sender advertises this GRTT estimate in every message it transmits so receivers have this value available for scaling their timers. To measure the current GRTT, the sender periodically sends NORM_CMD(CC) messages containing 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 assigns weights and smoothes the values for a conservative estimate of the GRTT. The algorithm and methodology are described in the Multicast NACK Building Block [RFC5401] document in the section entitled "One-to-Many Sender GRTT Measurement". A conservative estimate helps guarantee 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 the NORM-CC Rate header extension MAY also be used to proactively solicit RTT feedback from the receiver group per congestion control operation even when the sender is not 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 [RFC4654]. 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. 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 the NORM protocol message set MAY alternatively be used to support a window-based multicast congestion control scheme such as PGMCC. The details of such an 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 corresponding 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 [TcpModel], a steady-state sender transmission rate, to be "friendly" with competing TCP flows, can be calculated as:

$$Rsender = \frac{S}{T\_rtt*(sqrt((2/3)*p) + 12*sqrt((3/8)*p) * p * (1 + 32*(p^2)))}$$

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).

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

p = 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, they:

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 the current worst path in the group multicast topology.

The format of the NORM_CMD(CC) message is described 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 might be determined administratively or possibly algorithmically based upon 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 PLR list MAY be populated with a small number of receivers the sender identifies as approaching the CLR loss and delay conditions based on feedback from the group.

#### 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 the repeated transmission of NORM_CMD(CC) messages MAY be initiated some time before transmission of user data content at session startup. This can 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 initial value of GRTT_sender = 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.

3. 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 no more than one NORM_CMD(CC) message is sent per NORM_DATA message when there is data pending transmission. This ensures 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 will 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 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 will allow the sender to be more responsive to congestion control dynamics. The length of the list can 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 is prioritized with the following rules:

1. Receivers that have not yet been provided an 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 an 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 used for other congestion control functions. The NORM_FLAG_CC_PLR flag value is used to mark additional receivers from which the sender would like to have immediate, non-suppressed feedback. These can be receivers the sender algorithmically identified as potential future CLRs or 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 the NORM sender has been pre-configured with a set of PLR nodes, feedback from those receivers might not have yet 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 MUST 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 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 those 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_backoff * GRTT_sender, GSIZE_sender)
```

The RandomBackoff() algorithm provides a truncated exponentially distributed random number and is described in the Multicast NACK Building Block [RFC5401] document. The same backoff factor, K_backoff = K_sender, as used with NORM_NACK suppression is generally RECOMMENDED. However, in cases where the application purposefully

specifies a very small K_sender backoff factor to minimize the NACK repair process latency (trading off group size scalability), it is RECOMMENDED a larger backoff factor for congestion control feedback be maintained, since there can be a larger volume of congestion control feedback than NACKs in many cases and some congestion control feedback latency might be tolerable where reliable delivery latency is not. As previously noted, a backoff factor value of K_sender = 4 is generally RECOMMENDED for ASM operation and K_sender = 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*GRTT_sender. 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) might 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 its "cc_sequence" value in the corresponding 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_sender 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_sender * GRTT_sender)

Thus, non-CLR receivers are constrained to providing explicit congestion control feedback once per K_sender*GRTT_sender intervals. However, 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 [TfmccPaper], 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 determined either 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 SHOULD use the "cc_rtt" provided in the NORM-CC Feedback header extension as the receiver's previous RTT measurement (RTT_rcvrPrev) averaged with the current measurement ("RTT_rcvrNew") as the receiver's RTT value:

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 if a PLR is "promoted" to CLR status, the smoothed estimate can be continued.

There are some additional periods besides steady-state operation 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:

Rinit = MIN(NormSegmentSize/GRTT_sender, NormSegmentSize) bytes/sec

The rate is increased only when feedback is received from the receiver set. The "slow start" phase proceeds until any receiver

provides feedback indicating 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 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 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 rates. 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 the sender does not have explicit knowledge 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 observe 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 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) 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 a 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 of the receivers expected to 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 generating NORM_CMD(FLUSH) or NORM_CMD(ACK_REQ) messages in periodic rounds. For NORM_ACK(FLUSH) requests, the NORM_CMD(FLUSH) contains 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 that is similarly echoed in response so the sender can match the response to the appropriate request.

In response to the NORM_CMD(ACK_REQ), the listed receivers randomly, with a uniform distribution, transmit NORM_ACK messages over a time window of (1*GRTT_sender). These NORM_ACK messages are typically unicast to the sender. (Note 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 because:

1. Only a single NORM_CMD(ACK_REQ) message is generated once per (2*GRTT_sender), 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 will sometimes be necessary 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 needed 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 that is used in scaling random backoff timer ranges. The use of the group size estimate within the NORM protocol does not demand 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 also possible the group size MAY be algorithmically approximated from the volume of congestion control feedback messages based on the exponentially weighted random backoff. However, the specification of such an algorithm is currently beyond the scope of this document.

## 6. Configurable Elements

The NORM protocol supports a modest number of configurable parameters that control operation. Most of these need only be set at NORM sender(s) and the configuration information is communicated to the receiver set in NORM header and/or header extension fields. A notable exception to this is the NORM_ROBUST_FACTOR that is presumed to be a common value preset among senders and receivers for a given NORM session. The following table summarizes these configurable elements:

| Configurable Element | Purpose |
|---|---|
| Sender initial GRTT Estimate (GRTT_sender) | Sender's initial estimate of greatest group round-trip time. Affects timing of feedback suppression and sender command transmissions at sender startup. |
| Backoff Factor (K_sender) | Sender's scaling factor used for timer-based feedback suppression. |
| Group Size Estimate (GSIZE_sender) | Sender's rough estimate of receiver group size used in generation of random feedback backoff timeout. |
| NORM_ROBUST_FACTOR | Integer factor determining how persistently (i.e., robust) senders transmit repeated control messages and receivers self-initiate timeout-based NACKing in the absence of sender activity. |
| FEC Type ("fec_id") | Sender FEC encoding type. |
| Sender segment size (NormSegmentSize) | Maximum size (in bytes) of the payload portion of NORM_DATA and other messages. |
| NormNodeId | Unique identifiers pre-assigned to all NORM session participants. |

The sender-controlled GRTT estimate (referred to as GRTT_sender in this document) is used to set and scale various timers associated with NORM protocol operation. During steady-state operation, the sender probes the receiver set, adapts to the group round-trip timing state, and advertises its estimate to the receiver set in the "grtt" field of relevant NORM protocol messages. However, an initial value must be assumed at sender startup. A large initial estimate is conservative and safer with regard to preventing feedback implosion and starting up congestion control operation, but requires the sender and receivers to allocate more buffering resources for a given transmission rate (i.e., larger effective delay*bandwidth product) to maintain efficient operation. A default initial value of GRTT_sender = 0.5 seconds is RECOMMENDED.

The sender-controlled Backoff Factor (referred to a K_sender in this document) is used to scale protocol timers and contributes to the generation of the random backoff timeout value that facilitates timer-based feedback suppression. The sender advertises its configured Backoff Factor to the receiver set in the "backoff" field of applicable NORM messages and thus no receiver configuration is necessary. For ASM operation, a default value of K_sender = 4 is

RECOMMENDED; for SSM operation, a default value of K_sender = 6 is RECOMMENDED.

The sender estimate of session Group Size (referred to as GSIZE_sender in this document) also plays a role in the random selection of feedback suppression timeout values. The sender advertises its configured Group Size estimate to the receiver set in the "gsize" field of applicable NORM messages; thus, no receiver configuration is necessary. Only a rough estimate (i.e., "order-of-magnitude") is needed for effective feedback suppression and a default value of GSIZE_sender = 10,000 is RECOMMENDED as a conservative estimate for most uses.

The NORM_ROBUST_FACTOR is an integer parameter that determines how persistently NORM senders transmit control messages (NORM_CMD messages) such as end-of-transmission flushing, OPTIONAL positive acknowledgment requests, etc. Additionally, the receivers use their knowledge of NORM_ROBUST_FACTOR to determine when to consider a NORM sender inactive and MAY use the factor in determining how persistently to self-initiate repeated NACK repair requests upon such timeouts. This parameter is NOT communicated in NORM protocol message headers and is presumed to be preset to a consistent value among sender and receivers for a given NORM session. A default value of NORM_ROBUST_FACTOR = 20 is RECOMMENDED.

Another NORM sender configuration element is the FEC type used to encode NORM_DATA message content. The FEC type is communicated from the sender to the receiver set in the "fec_id" field of relevant NORM message headers. The "fec_id" value corresponds to an IANA-assigned value identifying the FEC encoding type as described in the FEC Building Block [RFC5052] document. Typically, a sender SHOULD use a consistent FEC encoding for its participation in a session to simplify receiver state allocation and maintenance, but its implementations MAY vary the FEC encoding type on a per-object basis if necessary.

The sender NormSegmentSize setting determines the maximum size of the payload portion of NORM_DATA and other messages that the sender transmits. Additionally, the payload size of feedback messages from receivers to a given sender is limited to that sender’s NormSegmentSize. The NormSegmentSize SHOULD be configured to be compatible with expected network MTU limitations, given the added overhead of NORM, UDP, and IP protocol message headers. Additionally, MTU Discovery MAY be employed by the sender to determine an appropriate NormSegmentSize. The NormSegmentSize for a given sender can be determined by receivers from the FEC Object Transmission Information (FTI) provided either in applied EXT_FTI header extensions or pre-configured session information.

Although it is not technically a configurable element, the receivers MUST have FEC Object Transmission Information for transmitted NormObjects to properly buffer, decode, and reassemble the original content. For loosely organized NORM protocol sessions, the sender MAY apply the EXT_FTI Header Extension to NORM_DATA and NORM_INFO (if applicable) messages so that receivers can get this information without prior coordination. An implementation MAY also apply the EXT_FTI only to NORM_INFO messages for reduced overhead. Finally, applications MAY also provide the FTI out-of-band prior to sender transmission.

Each participant in a NORM protocol session MUST be configured with a unique NormNodeId value. The NormNodeId value is used by receivers to identify the sender to which their NACK or other feedback messages are addressed, and senders use the NormNodeId to differentiate receivers for purposes of congestion control and OPTIONAL positive acknowledgment collection. Assignment of unique NormNodeId values can be done via a priori coordination and/or use of a deconfliction mechanism external to the NORM protocol itself. The values of NORM_NODE_NONE = 0x00000000 and NORM_NODE_ANY = 0xffffffff are reserved and MUST NOT be assigned to NORM participants.

## 7. Security Considerations

The same security considerations that apply to the Multicast NACK [RFC5401], TFMCC [RFC4654], and FEC [RFC5052] Building Blocks also apply to the NORM protocol. In addition to the vulnerabilities to which any IP and IP multicast protocol implementation is subject, malicious hosts might engage in excessive NACKing in an attempt to prevent the NORM sender(s) from making forward progress in reliable transmission. Receiver "join" and "service" policy enforcement as described in Section 5.2 can be applied if such activity is detected. The use of cryptographic peer authentication, integrity checks, and/or confidentiality mechanisms can be used to provide a more effective degree of protection from objectionable transmissions from unauthorized hosts. But in some cases, even with authentication and integrity checks, the NACK-based feedback of NORM can be exploited by replay attacks forcing the NORM sender to unnecessarily transmit repair information. This MAY be addressed in part with network-layer IP security implementations that guard against this potential security exploitation or alternatively with a security mechanism using the EXT_AUTH header extension for similar purposes. Such security mechanisms SHOULD be deployed and used when available. Use of security mechanisms will impose additional "a priori" configuration upon the NORM deployment depending upon the techniques used.

The NORM protocol is compatible with the use of IP security (IPsec)

[RFC4301], and the IPsec Encapsulating Security Payload (ESP) protocol or Authentication Header (AH) extension can be used to secure IP packets transmitted by NORM participants. A baseline approach to secure NORM operation using IPsec is described below. Compliant implementations of this specification are REQUIRED to be compatible with IPsec usage as described in Section 7.1. IPsec can be used to provide peer authentication, integrity protection, and/or encryption of packets containing NORM messages.

Additionally, the EXT_AUTH header extension (HET = 1) is reserved for use by security mechanisms to provide alternatives to IPsec for the security of NORM messages. The format of this header extension and its processing is outside the scope of this document and is to be communicated out-of-band as part of the session description. It is possible an EXT_AUTH implementation MAY also provide for encryption of NORM message payloads as well as peer authentication and integrity protection. The use of this approach as compared to IPsec can allow for header compression techniques to be applied jointly to IP and NORM protocol headers. In cases where security analysis deems encryption of NORM protocol header content to be beneficial or necessary, the aforementioned use of IPsec ESP might be more appropriate. Additionally, the EXT_AUTH header extension can be utilized when NORM is implemented in a network with Network Address Translation (NAT) systems that are incompatible with use of the IPsec AH extension. If EXT_AUTH is present, whatever packet authentication or integrity checks that can be performed immediately upon reception of the packet MUST be performed before accepting the packet and performing any congestion-control-related action on it. Some packet authentication schemes impose a delay of several seconds between when a packet is received and when the packet can be fully authenticated. Any appropriate congestion control related action MUST NOT be postponed by any such packet security mechanism (i.e., security mechanisms MUST NOT result in poor congestion control behavior).

Consideration MUST also be given to the potential for replay-attacks that would transplant authenticated packets from one NORM session to another to disrupt service. To avoid this potential, unique keys SHOULD be assigned on a per-session basis or NORM sender nodes SHOULD be configured to use unique "instance_id" identifiers managed as part of the security association for the sessions.

Note NORM implementations can use the "sequence" field from the NORM common message header to detect replay attacks. This can be accomplished if the NORM sender maintains state on actively NACKing receivers. A cache of such receiver state can be used to provide protection against NACK replay attacks. NORM receivers MUST also maintain similar state for protection against possible replay of other receiver messages in ASM operation as well. For example, a

receiver could be suppressed from providing NACK or congestion control feedback by replay of certain receiver messages. For these reasons, authentication of NORM messages (e.g., via IPsec) SHOULD be applied for protection against similar attacks that use fabricated messages. Also, encryption of messages to provide confidentiality of application data and protect privacy of users MAY also be applied using IPsec or similar mechanisms.

When applicable security measures are used, automated key management mechanisms such as those described in the Group Domain of Interpretation (GDOI) [RFC3547], Multimedia Internet KEYing (MIKEY) [RFC3830], or Group Secure Association Key Management Protocol (GSAKMP) [RFC4535] specifications SHOULD be applied.

While NORM does leverage FEC-based repair for scalability, this alone does not guarantee integrity of received data. Application-level integrity-checking of received data content is highly RECOMMENDED. This recommendation also applies when the IPsec security approach described below is used for added assurance in data content integrity given the shared use of IPsec Security Association information among the group.

### 7.1. Baseline Secure NORM Operation

This section describes a baseline mode of secure NORM protocol operation based on application of the IPsec security protocol. This approach is documented here to provide a baseline interoperable secure mode of operation. This particular approach represents one possible trade-off in the level of assurance that can be achieved and the scalability of multicast group-size given current IPsec mechanisms and the state required to support them. For example, this baseline approach specifies the use of a Security Association that is shared among the receiver set for feedback messages to the sender. This model requires that the receiver membership receiving the session keys is trusted and only provides protection from attacks that are external to the NORM group membership. More stateful and complex IPsec approaches and key management schemes may be applied for higher levels of assurance, but those are beyond the scope of this transport protocol specification. Additional approaches to NORM security, including other forms of IPsec application, MAY be specified in the future. For example, the use of the EXT_AUTH header extension could enable NORM-specific authentication or security encapsulation headers similar to those of IPsec to be specified and inserted into the NORM protocol message headers. This would allow header compression techniques to be applied to IP and NORM protocol headers when needed in a similar fashion to RTP [RFC3550] and as preserved in the specification for Secure Real Time Protocol (SRTP) [RFC3711].

The baseline approach described is applicable to NORM operation configured for SSM (or SSM-like) operation where there is a single sender and the receivers are providing unicast feedback. This form of NORM operation allows for IPsec to be used with a manageable number of security associations (SA).

#### 7.1.1. IPsec Approach

For NORM one-to-many SSM operation with unicast feedback from receivers, each node SHALL be configured with two transport mode IPsec security associations and corresponding Security Policy Database (SPD) entries. One entry will be used for sender-to-group multicast packet authentication and optionally encryption while the other entry will be used to provide security for the unicast feedback messaging from the receiver(s) to the sender. Note that this single SA for NORM receiver feedback messages is shared to protect traffic from possibly multiple receivers to the single sender.

For each NormSession, the NORM sender SHALL use an IPsec SA configured for ESP protocol [RFC4303] operation with the option for data origin authentication enabled. It is also RECOMMENDED this IPsec ESP SA be also configured to provide confidentiality protection for IP packets containing NORM protocol messages. This is suggested to make the realization of complex replay attacks much more difficult. The encryption key for this SA SHALL be preplaced at the sender and receiver(s) prior to NORM protocol operation. Use of automated key management is RECOMMENDED as a rekey SHALL be REQUIRED prior to expiration of the sequence space for the SA. This is necessary so receivers can use the built-in IPsec replay attack protection possible for an IPsec SA with a single source (the NORM sender). Thus, the receivers SHALL enable replay attack protection for this SA used to secure NORM sender traffic. An IPsec SPD entry MUST be configured to process outbound packets to the session (destination) address and UDP port number of the applicable (NormSession).

The NORM receiver(s) MUST be configured with the SA and SPD entry to properly process the IPsec-secured packets from the sender. The NORM receiver(s) SHALL also use a common, second IPsec SA (common Security Parameter Index (SPI) and encryption key) configured for ESP operation with the option for data origination authentication enabled. Similar to the NORM sender, is RECOMMENDED this IPsec ESP SA be also configured to provide confidentiality protection for IP packets containing NORM protocol messages. The receivers MUST have an IPsec SPD entry configured to process outbound NORM/UDP packets directed to the NORM sender source address and port number using this second SA. To support NORM unicast feedback, the sender’s transmission port number SHOULD be selected to be distinct from the

multicast session port number to allow discrimination between unicast and multicast feedback messages when access to the IP destination address is not possible (e.g., a user-space NORM implementation). For processing of packets from receivers, the NORM sender SHALL be configured with this common, second SA (and the corresponding SPD entry needed) in order to properly process messages from the receiver.

Multiple receivers using a common IPsec SA for traffic directed to the NORM sender (i.e., many-to-one) typically prevents the use of built-in IPsec replay attack protection by the NORM sender with current IPsec implementations. Thus the built-in IPsec replay attack protection for this second SA at the sender MUST be disabled unless the particular IPsec implementation manages its replay protection on a per-source basis (which is not typical of existing IPsec implementations). So, to support a fully secure mode of operation, the NORM sender implementation MUST provide replay attack protection based upon the "sequence" field of NORM protocol messages from receivers. This can be accomplished with a high assurance of security, even with the limited size (16-bits) of this field, because:

1. NORM receiver NACK and non-CLR ACK feedback messages are sparse.

2. The more frequent NORM_ACK feedback from CLR or PLR nodes is only a small set of receivers for which the sender needs to keep more persistent replay attack state.

3. NORM_NACK feedback messages preceding the sender’s current repair window do not significantly impact protocol operation (generation of NORM_CMD(SQUELCH) is limited) and could be in fact ignored. This means the sender can prune any replay attack state that precedes the current repair window.

4. NORM_ACK messages correspond to either a specific sender "ack_id", the sender "cc_sequence" for ACKs sent in response to NORM_CMD(CC), or the sender’s current repair window in the case of ACKs sent in response to NORM_CMD(FLUSH). Thus, the sender can prune any replay attack state for receivers that precede the current applicable sequence or repair window space.

The use of ESP confidentiality for secure NORM protocol operation makes it more difficult for adversaries to conduct any form of replay attacks. Additionally, a NORM sender implementation with access to the full ESP protocol header could also use the ESP sequence information to make replay attack protection even more robust by maintaining the per-source ESP sequence state that existing IPsec implementations typically do not provide. The design of this

baseline security approach for NORM intentionally places any more complex processing state or processing (e.g., replay attack protection given multiple receivers) at the NORM sender since NORM receiver implementations might often need to be less complex.

This baseline approach can be used for NORM protocol sessions with multiple senders if the SA pairs described are established for each sender. For small-sized groups, it is even possible many-to-many (ASM) IPsec configuration could be achieved where each participant uses a unique SA (with a unique SPI). In this case, the sender(s) would maintain an SA for each other participant rather than a single, shared SA for receiver feedback messages. This does not scale to larger group sizes given the complex set of SA and SPD entries each participant would need to maintain.

It is anticipated in early deployments of this baseline approach to NORM security that key management will be conducted out-of-band with respect to NORM protocol operation. In the case of one-to-many NORM operation, it is possible receivers will retrieve keying information from a central server as needed or otherwise conduct group key updates with a similar centralized approach. Alternatively, it is possible with some key management schemes for rekey messages to be transmitted to the group as a message or transport object within the NORM reliable transfer session. Similarly, for group-wise communication sessions, it is possible for potential group participants to request keying and/or rekeying as part of NORM communications. Additional specification is necessary to define an in-band key management scheme for NORM sessions perhaps using the mechanisms of the automated group key management specifications cited in this document. Additional specification outside of the scope of this document would be needed to provide an interoperable approach for key management in-band of a NORM reliable transport session.

#### 7.1.2. IPsec Requirements

In order to implement this secure mode of NORM protocol operation, the following IPsec capabilities are REQUIRED.

##### 7.1.2.1. Selectors

The implementation MUST be able to use the source address, destination address, protocol (UDP), and UDP port numbers as selectors in the SPD.

##### 7.1.2.2. Mode

IPsec in transport mode MUST be supported. The use of IPsec [RFC4301] processing for secure NORM traffic MUST be configured such

that unauthenticated packets are not received by the NORM protocol implementation.

#### 7.1.2.3. Key Management

An automated key management scheme for group key distribution and rekeying such as GDOI [RFC3547], GSAKMP [RFC4535], or MIKEY [RFC3830] is RECOMMENDED for use. Note it is possible for key update messages (e.g., the GDOI GROUPKEY-PUSH message) to be included as part of the NORM application reliable data transmission if appropriate interfaces are available between the NORM application and the key management daemon. Relatively short-lived NORM sessions MAY be able to use Manual Keying with a single, preplaced key, particularly if Extended Sequence Numbering (ESN) [RFC4303] is available in the IPsec implementation used. When manual keys are used, it is important that cryptographic algorithms suitable for manual key use are selected.

#### 7.1.2.4. Security Policy

Receivers MUST accept protocol messages only from the designated, authorized sender(s). Appropriate key management will provide authentication, integrity and/or encryption keys only to receivers authorized to participate in a designated session. The approach outlined here allows receiver sets to be controlled on a per-sender basis.

#### 7.1.2.5. Authentication and Encryption

Large NORM group sizes will necessitate some form of key management that does rely upon shared secrets. The GDOI and GSAKMP protocols mentioned here allow for certificate-based authentication. It is RECOMMENDED these certificates use IP addresses for authentication.

#### 7.1.2.6. Availability

The IPsec requirements profile outlined here is commonly available on many potential NORM hosts. Configuration and operation of IPsec typically requires privileged user authorization. Automated key management implementations are typically configured with the privileges necessary to affect system IPsec configuration.

## 8. IANA Considerations

Values of NORM Header Extension Types, Stream Control Codes, and NORM_CMD message sub-types are subject to IANA registration. They are in the registry named "Reliable Multicast Transport (RMT) NORM Protocol Parameters" available from http://www.iana.org.

Note the reliable multicast building block components used by this specification also have their respective IANA considerations, and those documents SHOULD be consulted accordingly. 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 RFC 5052. It is possible additional extensions of the NORM protocol might be specified in the future (e.g., additional NORM message types) and additional registries be established at that time with appropriate IETF standards action.

### 8.1. Explicit IANA Assignment Guidelines

This document introduces three registries for the NORM Header Extension Types, Stream Control Codes, and NORM_CMD Message sub-types. This section describes explicit IANA assignment guidelines for each of these.

#### 8.1.1. NORM Header Extension Types

This document defines a registry for NORM Header Extensions named "NORM Header Extension Types".

The NORM Header Extension Type field is an 8-bit value. The values of this field identify extended header content allowing the protocol functionality to be expanded to include additional features and operating modes. The values that can be assigned within the "NORM Header Extensions" registry are numeric indexes in the range {0, 255}, boundaries included. Values in the range {0,127} indicate variable-length extended header fields while values in the range {128,255} indicate extensions of a fixed 4-byte length. This specification registers the following NORM Header Extension Types:

| Value | Name | Reference |
|---|---|---|
| 1 | EXT_AUTH | This specification |
| 3 | EXT_CC | This specification |
| 64 | EXT_FTI | This specification |
| 128 | EXT_RATE | This specification |

Requests for assignment of additional NORM Header Extension Type values are granted on a "Specification Required" basis as defined by IANA Guidelines [RFC5226]. Any such header extension specifications MUST include a description of protocol actions to be taken when the extension type is encountered by a protocol implementation not supporting that specific option. For example, it is often possible for protocol implementations to ignore unknown header extensions.

### 8.1.2. NORM Stream Control Codes

This document defines a registry for NORM Stream Control Codes named "NORM Stream Control Codes".

NORM Stream Control Codes are 16-bit values that can be inserted within a NORM_OBJECT_STREAM delivery object to convey sequenced, out-of-band (with respect to the stream data) control signaling applicable to the referenced stream object. These control codes are to be delivered to the application or protocol implementation with reliable delivery, in-order with respect to the their inserted position within the stream. This specification registers the following NORM Stream Control Code:

| Value | Name | Reference |
|---|---|---|
| 0 | NORM_STREAM_END | This specification |

Additional NORM Stream Control Code value assignment requests are granted on a "Specification Required" basis as defined by IANA Guidelines [RFC5226]. The full 16-bit space outside of the value assigned in this specification are available for future assignment. In addition to describing the control code’s expected interpretation, such specifications MUST include a description of protocol actions to be taken when the control code is encountered by a protocol implementation not supporting that specific option.

### 8.1.3. NORM_CMD Message Sub-Types

This document defines a registry for NORM_CMD message sub-types named "NORM Command Message Sub-types".

The NORM_CMD message "sub-type" field is an 8-bit value with valid values in the range of 1-255. Note the value 0 is reserved to indicate an invalid NORM_CMD message sub-type. The current specification defines a number of NORM_CMD message sub-types senders can use to signal the receivers in various aspects of NORM protocol operation. This specification registers the following NORM_CMD Message Sub-types:

| Value | Name | Reference |
|---|---|---|
| 0 | reserved | This specification |
| 1 | NORM_CMD(FLUSH) | This specification |
| 2 | NORM_CMD(EOT) | This specification |
| 3 | NORM_CMD(SQUELCH) | This specification |
| 4 | NORM_CMD(CC) | This specification |
| 5 | NORM_CMD(REPAIR_ADV) | This specification |
| 6 | NORM_CMD(ACK_REQ) | This specification |
| 7 | NORM_CMD(APPLICATION) | This specification |

Future specifications extending NORM MAY define additional NORM_CMD messages to enhance protocol functionality. NORM_CMD message sub-type value assignment requests are granted on a "Specification Required" basis as defined by IANA Guidelines [RFC5226]. In addition to describing the command sub-type's expected interpretation, specifications MUST include a description of protocol actions to be taken when the command is encountered by a protocol implementation not supporting that specific option.

This specification already defines an "application-defined" NORM_CMD message sub-type for use at the discretion of individual applications using NORM for transport. These "application-defined" commands are suitable for many application-specific purposes and do not involve standards action. In any case, such additional messages SHALL be subject to the same congestion control constraints as the existing NORM sender message set.

## 9. Suggested Use

The present NORM protocol is seen as a 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 out-of-order delivery. 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 is 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., direct broadcast satellite (DBS) or cable and public-switched telephone network (PSTN) hybrids, etc.) and efficient, reliable bulk data transfer will be an important capability for servicing large groups of subscribed receivers.

## 10. Changes from RFC 3940

This section lists the changes between the Experimental version of this specification, RFC 3940, and this version:

1. Removal of the NORM_FLAG_MSG_START for NORM_OBJECT_STREAM, replacing it with the "payload_msg_start" field in the FEC-encoded preamble of the NORM_OBJECT_STREAM NORM_DATA payload.

2. Definition of IANA registry for header extension and other assignments.

3. Removal of file blocking scheme description now specified in the FEC Building Block document [RFC5052].

4. Removal of restriction of NORM receiver feedback message rate to local NORM sender rate (this caused congestion control failures in high speed operation. The extremely low feedback rate of the NORM protocol as compared to TCP avoids any resultant impact to the network as shown in [Mdpcc].)

5. Correction of errors in some message format descriptions.

6. Correction of inconsistency in specification of the inactivity timeout.

7. Addition of IPsec secure mode description with IPsec requirements.

8. Addition of the EXT_AUTH header extension definition.

9. Clarification of interpretation of "Source Block Length" when FEC codes are arbitrarily shortened by the sender.

## 11. 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.

## 12. References

### 12.1. Normative References

[RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.

[RFC4654] Widmer, J. and M. Handley, "TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification", RFC 4654, August 2006.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, August 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5401] Adamson, B., Bormann, C., Handley, M., and J. Macker, "Multicast Negative-Acknowledgment (NACK) Building Blocks", RFC 5401, November 2008.

## 12.2. Informative References

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

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

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

[Mdpcc] Adamson, B. and J. Macker, "A TCP-Friendly, Rate-based Mechanism for NACK-Oriented Reliable Multicast Congestion Control", Proc. IEEE GLOBECOMM, November 2001.

[NormFeedback] Adamson, B. and J. Macker, "Quantitative Prediction of NACK-Oriented Reliable Multicast (NORM) Feedback", IEEE MILCOM, October 2002.

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

[RFC2357] Mankin, A., Romanov, A., Bradner, S., and V. Paxson, "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC3048] 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.

[RFC3269] Kermode, R. and L. Vicisano, "Author Guidelines for Reliable Multicast Transport (RMT) Building Blocks and Protocol Instantiation documents", RFC 3269, April 2002.

[RFC3453] 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.

[RFC3547] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC3940] Adamson, B., Bormann, C., Handley, M., and J. Macker, "Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol", RFC 3940, November 2004.

[RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, June 2006.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5445] Watson, M., "Basic Forward Error Correction (FEC) Schemes", RFC 5445, March 2009.

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

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

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

Authors' Addresses

Brian Adamson
Naval Research Laboratory
Washington, DC 20375
USA

EMail: adamson@itd.nrl.navy.mil

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

EMail: cabo@tzi.org

Mark Handley
University College London
Gower Street
London WC1E 6BT
UK

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

Joe Macker
Naval Research Laboratory
Washington, DC 20375
USA

EMail: macker@itd.nrl.navy.mil