NACK-Oriented Reliable Multicast (NORM) Protocol Building Blocks

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This memo describes issues related to the creation of negative-acknowledgment (NACK) oriented reliable multicast (NORM) protocols. The general goals and assumptions for NORM are defined. The technical challenges related to NACK-oriented (and in some cases general) reliable multicast protocol design are identified. These challenges are resolved into a set of applicable "building blocks" which are described in further detail. It is anticipated that these building blocks (as they are further refined and defined in future revisions of this document) will be useful in generating different instantiations of reliable multicast protocols.
1.0 Background

Reliable multicast transport is a desirable technology for the efficient and reliable distribution of data to a group on the Internet. The complexities of group communication paradigms necessitate different protocol types and instantiations to meet the range of performance and scalability requirements of different potential reliable multicast applications and users [Mankin98]. Properly designed negative-acknowledgement (NACK) oriented reliable multicast (NORM) protocols offer scalability advantages for applications and/or network topologies where, for various reasons, it is prohibitive to construct a higher order delivery infrastructure above the basic Layer 3 IP multicast service (e.g. unicast or hybrid unicast/multicast data distribution trees). Additionally, the scalability property of NACK-oriented protocols [Pingali93, Levine96] may be applicable where broad "fanout" is expected for a single network hop (e.g. cable-TV data delivery, satellite, or other broadcast communication communication services). Furthermore, the simplicity of a protocol based on "flat" group-wide multicast distribution may offer advantages for a broad range of distributed services or dynamic networks and applications.

While different protocol instantiations may be required to meet specific application and network architecture demands [Clark90], there are a number of fundamental components which may be common to these different instantiations. This document describes the framework and common "building block" components relevant to multicast protocols based primarily on NACK operation for reliable transport.

2.0 Applicability

Each potential protocol instantiation using the building blocks presented here (and other applicable building block documents) will have specific criteria which will influence individual protocol design. To support the development of applicable building blocks, it is useful to identify and summarize driving general protocol design goals and assumptions. These are areas which each protocol instantiation will need to address in detail. Each building block description in this document will include a discussion of the impact of these design criteria. The categories of design criteria considered here include:

1) Data content model
2) Group membership dynamics
3) Sender/receiver relationships
4) Group size
5) Data delivery performance
6) Network topology
3) Router/intermediate system assistance

In some cases, a building block may be designed to address a wide range of assumptions while in other cases there will be trade-offs required to meet different application needs. Where necessary, building block features will designed to be parametric to meet different requirements. Of course, an underlying goal will be to minimize design complexity and to at least recommend default values for any such parameters which meet a general purpose "bulk data transfer" requirement in a typical Internet environment.

2.1 Data content model

The implicit goal of a reliable multicast protocol is the reliable delivery of "data" among a group of members communicating through IP multicast datagram service. However, the nature of the data content and the service the application is attempting to provide can impact design decisions. The service model may range from long-lived transfer sessions of bulk quantities of data (file broadcast) to more interactive exchanges of small messages (e.g. white-boarding, text chat). And within those different models there are other issues such as the sender’s ability to cache transmitted data (or state referencing it) for retransmission or repair. The needs for ordering and/or causality in the sequence of transmissions and receptions among members in the group may be different depending upon data content. The group communication paradigm differs significantly from the point-to-point model in that, depending upon the data content type, some receivers may complete reception of a portion of data content and be able to act upon it before other members have received the content. This may be acceptable (or even desirable) for some applications but not for others. These varying requirements drives the need for a number of different protocol instantiation designs.

A significant challenge in developing generally useful building block mechanisms is accommodating even a limited range of these capabilities without defining specific application-level details. Note that this particular building block "guideline" may be generally applicable beyond the realm of NACK-oriented reliable multicast.

2.2 Group membership dynamics

Group communication can differ from point-to-point communications with respect to the fact that even if the composition of the group changes, the "thread" of communication can still exist. This contrasts with the point-to-point communication model where, if either of the two parties leave, the communication process (exchange of
data) is terminated (or at least paused). Depending upon application goals, senders and receivers participating in a reliable multicast transport "session" may be able to join late, leave, and/or potentially rejoin while the ongoing group communication "thread" still remains functional and useful.

Also note that this can impact protocol message content. If "late joiners" are supported, some amount of additional information may be placed in message headers to accommodate this functionality. Alternatively, the information may be sent in its own message (on demand or intermittently) if the impact to the overhead of typical message transmissions is deemed too great. Group dynamics can also impact other protocol mechanisms such as NACK or other timing, congestion control operation, etc.

2.3 Sender/receiver relationships

The relationship of senders and receivers among group members requires consideration. In some applications, there may be a single sender multicasting to a group of receivers. In other cases, there may be more than one sender or the potential for everyone in the group to be a sender _and_ receiver of data may exist.

2.4 Group size

Native IP multicast [Deering89] may scale to extremely large group sizes. It may be desirable for some applications to be able to scale along with the multicast infrastructure’s ability to scale. In its simplest form, there are limits to the group size to which a NACK-oriented protocol can apply without NACK implosion problems. However, the potential for router assistance or other non-linear NACK suppression mechanisms may enable these protocols to scale to very large group sizes. In large scale cases, it may be prohibitive for members to maintain state on all other members (in particular, other receivers) in the group. The impact of group size needs to be considered in the development of generally applicable building blocks.

2.5 Data delivery performance

There is a trade-off between scalability and data delivery latency when designing NACK-oriented protocols. If NACK suppression is to be used, there will be some delays built into the NACK generation and repair transmission process to allow suppression to occur and for the sender of data to identify appropriate content for efficient repair transmission. For example, backoff timeouts can be used to ensure efficient NACK suppression and repair transmission, but this comes at a cost of increased delivery latency and
increased buffering requirements for both senders and receivers. The building blocks should allow applications to establish bounds for data delivery performance. Note that application designers must be aware of the scalability trade-off which is made when such bounds are applied.

2.6 Network topology

The Internet Protocol has historically assumed a role of providing service across heterogeneous network topologies. It is desirable that a reliable multicast protocol be capable of effectively operating across a wide range of the networks to which general purpose IP service applies. The bandwidth available on the links between the members of a single group today may vary between low numbers of kbit/s for wireless links and multiple Gbit/s for high speed LAN connections, with varying degrees of contention from other flows. Recently, a number of asymmetric network services including 56K/ADSL modems, CATV Internet service, satellite and other wireless communication services have begun to proliferate. Many of these are inherently broadcast media with potentially large "fanouts" to which IP multicast service is highly applicable.

Additionally, policy and/or technical issues may result in topologies where multicast connectivity is limited to a single logical direction from a specific source or set of sources to the group at large. Receivers in the group may be restricted to unicast feedback for NACKs and other messages. Consideration must be given, in building block development and protocol design, to the nature of the underlying networks over which the protocols may be operating.

2.7 Router/Intermediate System Assistance

While intermediate assistance from devices/systems with direct knowledge of the underlying network topology may used to leverage the performance and scalability of reliable multicast protocols, there will continue to be a number of instances where this is not available or practical. Any building block components for NACK-oriented reliable multicast should be capable of operating without such assistance but should also be capable of utilizing these features when available.

3.0 Building Block Areas

The previous section has presented in general what building blocks are intended to be and some of the criteria which may affect building block identification/design. This section describes different building block areas applicable to NACK-oriented reliable multicast protocols. Some of these areas are specific to NACK-oriented
protocols. Detailed descriptions of such areas are provided below. In other cases, the areas (e.g. node identifiers, FEC, etc) may be generally applicable to other forms of reliable multicast. In those cases, the discussion below describes requirements placed on those other general building block areas from the standpoint of NACK-oriented reliable multicast.

For the individual building blocks to be incorporated as part of a specific protocol instantiation, it is expected that a description of some notional "interface" to the building blocks' functionality be provided. For example, a building block component may require some form of "input" from another building block component or other source in order to perform its function. Any "inputs" required by each building block component and/or any resultant "output" provided by the building block will be defined and described as the building block component's interface definition.

The following building block areas are described below:

(NACK-Oriented Components)
1) Sender transmission
2) NACK-oriented Repair Process
3) "Late-joining" Receiver Policies and Mechanisms

(Generally-applicable Components)
4) Node (member) Identification
5) Data Content Identification
6) Forward Error Correction
7) Round-trip Timing Collection
8) Group Size Determination/Estimation
9) Congestion Control Operation
10) Router/Intermediate System Assistance
11) Additional Protocol Mechanisms

Figure 1 provides an pictorial overview of these building block areas and their relationships. For example, the content of the data messages that sender initially transmits depends upon the "Node Identification", Data Content Identification", "FEC", and "Congestion Control components (Note that the rate of message transmission will generally depend upon the "Congestion Control" component). Subsequently, the receivers response to these transmissions (e.g. NACKing for repair) will depend upon the content of these transmissions and inputs from other building block components. Then, the sender's processing of these responses will feed back into its transmission strategy.
The components on the left side of this figure represent the components which may be generally applicable beyond NORM and those on the right are specific to NORM protocols. Some components (e.g. "Security") will impact many aspects of the protocol and others such as "Router Assistance" may be more transparent to the core protocol processing. The sections below discuss issues with regards to these building block components and their relationships. Where applicable, specific technical recommendations are made for mechanisms which will properly satisfy the goals of reliable multicast bulk transport for the Internet.

3.1 Sender transmission
Senders will transmit data content to the multicast session. The data content will be application dependent. The sender will transmit data content at a rate and with packet sizes determined by application and/or network architecture requirements. When congestion control mechanisms are used (recommended), the transmission rate will be controlled by the congestion control mechanism. It is recommended that all data transmissions from senders be subject to rate limitations determined by the application or congestion control algorithm. The sender’s transmissions should make good utilization of the available capacity (which may be limited by the application and/or by congestion control). As a result, it is expected there will be overlap and multiplexing of new data content transmission with repair content.

In addition to data content, other sender messages or commands may be employed as part of protocol operation. For NACK-oriented operation, the reliability of any such commands may depend upon redundant transmission. Other factors related to NACK-oriented operation may determine sender transmission formats and methods. Some consideration needs to be given to the sender’s behavior during intermittent idle periods when it has no data to transmit. While many aspects may be protocol-specific, there are techniques which may be generally applicable to NACK-oriented reliable multicast. For example, the periodicity of redundant transmission of command messages issued by a sender should be dependent upon the greatest round trip timing estimate and the resultant NACK operation. More specifically, a command message might be redundantly transmitted by a sender to indicate that it is temporarily (or permanently) halting transmission. At this time, it may be appropriate for receivers to respond with NACKs for any outstanding repairs they require following the rules of the NACK process described below. For efficiency, the sender should allow sufficient time between redundant transmissions to receive any NACK-oriented responses from the receiver set to this command. Other protocol components may benefit from a similar approach.

Inputs:

1) Data content
2) Segmentation size
3) Transmission rate
4) Application controls

Outputs:

1) Rate-controlled stream of packets with headers uniquely identifying data or repair content within the context of the reliable multicast session.
2) Commands indicating sender’s status or other transport control actions to be taken.
3.2 NACK-oriented repair process

The most critical component of the NACK-oriented reliable multicast protocol building block is the NACK repair process. There are four primary elements of a general NACK repair process:

1) Method for determining when receivers will initiate the NACK process in response to sender transmission for which they need repair.
2) NACK message content.
3) NACK suppression mechanisms to promote scalability of the protocol.
4) Sender NACK reception, aggregation, and repair response.

3.2.1 NACK Process Initiation

The NACK process (cycle) will be initiated by receivers who detect they require repair transmissions from a specific sender at defined opportunities. When FEC is applied, a NACK cycle should only be initiated when it is known by the receiver that its repair requirements exceed the amount of FEC pending transmission for a given coding block of packets. This may be determined by the receiver if the sender’s transmission advertises the quantity of repair packets it is already planning to send for a block, and/or at the end of the current transmission block (if it is indicated) or at the start of subsequent coding block for packets transmitted within the context of a designed data content set (See object/stream data content identifier discussion below).

Allowing receivers to initiate NACK cycles at any time they detect their repair needs have exceeded pending repair transmissions may result in slightly quicker repair cycles. However, it may be useful to limit the initiation opportunities to specific events such as at the end-of-transmission of an FEC coding block (or alternatively at detection of subsequent coding blocks). This can allow receivers to aggregate NACK content into a smaller number of NACK messages. In either case, the NACK cycle should begin with receivers observing backoff timeouts to facilitate NACK suppression as described below.

Interface Description

Inputs:

1) Object/stream data content and sequencing identifiers from sender transmissions.

Outputs:
1) NACK Process Initiation Decision

3.2.2 NACK Content

The content of NACK messages generated by reliable multicast receivers will include information detailing the current repair needs of each receiver. The specific information depends on the use and type of FEC in the NORM repair process. The identification of repair needs is dependent upon the data content identification (See Section 3.5 below). At the highest level the NACK content will identify the data transport object (or stream) within the multicast session which needs repair. For the indicated transport entity, the NACK content will then identify the specific coding blocks and/or segments of coding blocks it needs to reconstruct the transmitted data. This content may consist of FEC block erasure counts and/or explicit indication of missing blocks or segments of data and FEC content.

3.2.2 NACK Content Strategies

Where FEC-based repair is used, the NACK message content will minimally need to identify the coding block(s) for which repair is needed and a count of erasures (missing packets) for the coding block. Note that this assumes the FEC algorithm is capable of repairing _any_ loss combination within the coding block and that the quantity of unique FEC parity packets the server has available to transmit is essentially unlimited (i.e. the server will always be able to provide new parity packets in response to any subsequent repair requests for the same coding block). In other cases, the NACK content will need to also _explicitly_ identify which packets (information and/or parity) the receiver requires to successfully reconstruct the content of the coding block. This will be true of many applicable small to medium size block codes (e.g. Reed Solomon).

When FEC is not used as part of the repair process or the protocol instantiation is required to provide reliability even when the sender has transmitted all available parity for a given coding block (or the sender’s ability to buffer transmission history is exceeded by the delay*bandwidth*loss characteristics of the network topology), the NACK content will need to contain _explicit_ coding block and/or packet loss information so that the sender can provide appropriate repair packets and/or data retransmissions. Explicit loss information in NACK content may also potentially serve other purposes. For example, it may be useful for decorrelating loss characteristics among a group of receivers to help differentiate candidate congestion control bottlenecks among the receiver set.
For cases where the amount of loss is very small with respect to the coding block size, it may be efficient to simply provide a list of coding block vector identifiers. However, in many cases, a bit mask marking the locations of missing packets may be significantly more efficient for communicating receiver repair needs. And since the data content is logically divided into coding blocks, a system of hierarchical bit masks can be used to encode missing content at the object/stream, FEC block, and individual packet levels. Hierarchical bit mask encoding can provide compact NACK messages even in high delay*bandwidth*loss conditions. Bit mask based NACK content can also be efficiently processed with logical operations during protocol operation.

When FEC is used and NACK content is designed to contain explicit repair requests, there is a strategy where the receivers can NACK for specific content which will help facilitate NACK suppression and repair efficiency. The assumptions for this strategy are that sender may potentially exhaust its supply of new, unique parity packets available for a given coding block and be required to explicitly retransmit some data or parity segments to complete reliable transfer. Another assumption is that an FEC algorithm where any parity packet can fill any erasure within the coding block is used. The goal of this strategy is to make maximum use of the available parity and provide the minimal amount of data and repair transmissions during reliable transfer of a coding block to the group.

In this approach, the sender transmits the data content of the coding block (and optionally some quantity of parity packets) in its initial transmission. Note that a coding block is considered to be logically made up of the contiguous set of data vectors plus parity vectors for the given FEC algorithm used.

Receivers construct NACK messages requesting sufficient parity content to satisfy their repair needs. For example, if the receiver has three erasures in the received coding block, it requests transmission of the three lowest ordinal parity vectors in the coding block. (In the case of a code where the loss exceeds TBD TBD TBD)

In response to repair request, the sender transmits parity vectors beginning from the lowest ordinal part of the parity portion of the coding block.

When the receivers construct explicit NACK content, they request transmission of only the _upper_ ordinal portion, corresponding to the number of _data_ vectors, of the logical coding block required to fill their packet erasures (Note this _may_ include data vectors if there is a smaller number of parity vectors than data vectors.
for the selected code, but generally will consist solely of parity vectors). The receivers can also provide a count of erasures as a convenience (saves processing time) to the sender. Upon receipt of the NACK message, the sender will schedule transmission of fresh, new parity vectors based on the erasure count _if_ it has a sufficient quantity of vectors which were _not_ previously transmitted and ignore the explicit content requested. Otherwise, the sender will resort to transmitting the set of repair vectors requested.  With this approach, the sender needs to maintain very little state on requests it has received from the group without need for synchronization of repair requests from the group. Since all receivers use this same algorithm to express their explicit repair needs, NACK suppression among receivers is simplified over the course of multiple repair cycles. The receivers can simply compare NACKs heard from other receivers against their own calculated repair needs to determine whether they should transmit or suppress their pending NACK messages.

3.2.3 NACK Content Encapsulation

The format of NACK content will depend on the protocol’s data service model and the format of data content identification the protocol uses. This is also dependent upon the type of FEC encoding (if any) is used. Figure 2 illustrates a general logical hierarchy of transmission content identification, denoting that the notion of objects (or streams) and/or FEC blocking is optional at the protocol instantiation’s discretion. Since the notion of session "streams" and "blocks" are optional, the framework degenerates to that of typical transport data segmentation and reassembly in its simplest form.

```
  Session
     /  \
  [Object/Stream(s)]
     /  \
  [FEC Blocks]
    /  \ Segments
```

Figure 2: Hierarchy of Reliable Data Transfer Content

The format of NACK messages should meet the following goals:

1) Describe a basic unit to identify transport data unit transmissions required to repair a portion of the received content, whether it is an entire missing object/stream (or range), entire FEC coding block(s), or sets of segments,
1) Be simple to process for NACK aggregation and suppression,

2) Be capable of including NACKs for multiple objects, fec coding blocks and/or symbols in a single message. FEC erasure counts may also be desirable.

3) Have a compact format, and

4) Be capable of working with the Generic Router Assist (GRA) building block.

The concatenation of "<objectId><blockId><segmentId>" comprises a basic transport protocol data unit (TPDU) identifier of segments transmitted from a given source. NACK content can be composed of lists and/or ranges of these TPDU identifiers to build up NACK messages to describe the receivers repair needs. If no hierarchical object delineation or FEC blocking is used, the TPDU is a simple linear representation of the data segments transmitted by the sender. When the TPDU represents a hierarchy for purposes of object/stream delineation and/or FEC blocking, the NACK content unit may require flags to indicate which portion of the TPDU is applicable. For example, if an entire "object" (or range of objects) is missing in the received data, the receiver will not necessarily know the appropriate range of <blockIds> or <segmentIds> for which to request repair and thus requires some mechanism to request repair (or retransmission) of the entire unit represented by an <objectId>. The same is true if entire FEC coding blocks represented by one or a range of <blockIds> is missing for a receiver.

3.2.4 NACK Suppression Mechanisms

A primary NACK suppression mechanism is the use of initial backoff timeouts by receivers wishing to transmit NACK messages[Floyd95]. Upon expiration of the timeout, a receiver will transmit a NACK unless the content of the pending repair request is completely superseded by NACK messages heard from other receivers (when receivers are multicasting NACKs) or from some indicator from the sender. (Note: When receivers are unicasting NACK messages, the sender may facilitate NACK suppression by forwarding appropriate NACKs it has received to the group at large or provide some other indicator of the repair information it will be subsequently transmitting).

The backoff timeout periods used by receivers should be independently, randomly picked by receivers with an exponential
distribution [Nonnenmacher98]. This results in the bulk of the receiver set holding off transmission of NACK messages under the assumption that the smaller number of "early NACKers" will supersede the repair needs of the remainder of the group. The mean of the distribution should be determined as a function of the current estimate of sender<->group greatest round trip time (GRTT) and a group size estimate which determined by other mechanisms within the protocol (See section below) or preset by the multicast application.

A simple algorithm can be constructed to generate random backoff timeouts with the appropriate distribution. Additionally, the algorithm may be designed to optimize the backoff distribution given the number of receivers (R) potentially generating feedback. This "optimization" minimizes the number of feedback messages (e.g. NACK) in the worst-case situation where all receivers generate a NACK. The maximum backoff timeout (T) can also be controlled to allow the application or protocol tradeoff NACK latency versus volume of feedback traffic. A larger value of T will result in a lower density of feedback traffic for a given repair cycle. A smaller value of T results in shorter latency which reduces the buffering requirements of senders and receivers for reliable transport.

Given the receiver group size (R), and maximum allowed backoff timeout (T), a truncated exponential backoff timeout (t') can be picked with the following algorithm:

1) Establish an optimal mean (L) for the exponential backoff based on the group size:

   \[ L = \ln(R) + 1 \]

2) Pick a random number (x) from a uniform distribution over a range of:

   \[ \frac{L}{T \times (\exp(L) - 1)} \quad \text{to} \quad \frac{L}{T \times (\exp(L) - 1)} + \frac{L}{T} \]

3) Transform this random variate to generate the desired random backoff time (t) with the following equation:

   \[ t' = \frac{T}{L} \times \ln(x \times (\exp(L) - 1) \times (T/L)) \]

This is a C language function which can be used to perform this function:
double RandomBackoff(double maxTime, double groupSize)
{
    double lambda = log(groupSize) + 1;
    double x = UniformRand(lambda/maxTime) +
        lambda / (maxTime*(exp(lambda)-1));
    return ((maxTime/lambda) * 
        log(x*(exp(lambda)-1)*(maxTime/lambda)));
}  // end RandomBackoff()

where UniformRand(double max) returns random numbers with a uniform
distribution from the range of 0..max. For example, based on the
POSIX "rand()" function, the following code can be used:

double UniformRand(double max)
{
    return (max * ((double)rand()/(double)RAND_MAX));
}

The number of expected NACKs generated (N) within the first round
trip time for a single loss event can be approximately expected to
be:

\[ N = \exp(1.2 \times \frac{L}{2 \times \text{GRTT}}) \]

Thus the maximum backoff time (T) can be adjusted to tradeoff
worst-case NACK feedback volume versus latency. This is derived
from [Nonnenmacher98] and assumes \( T \geq \text{GRTT} \), and \( L \) is the mean of
the distribution optimized for the given group size as shown in the
algorithm above. Note that other mechanisms within protocol may
work to reduce redundant NACK generation further.

There are some secondary NACK suppression mechanisms which can also
be considered. For example, the sender’s transmissions may follow
an ordinal sequence of transmission (observed through data/repair
content <objectIds> and/or <segmentIds>) which is "rewound" during
repair transmissions. Receivers may wish to limit transmission of
their NACKs only when the sender’s current sequenced position of
transmission passes the point at which the receiver has incomplete
transmissions, thus reducing premature requests for repair of data
the sender may be providing in response to other receiver requests.
This mechanism can be very effective for protocol convergence in
high loss conditions when transmissions of NACKs from other
receivers (or indicators from the sender) are lost. Another mecha-
nism (particularly applicable when FEC is used) is for the sender
to embed an indication of impending repair transmissions in current
packets sent. For example, the indication may be as simple as an
advertisement of the number of FEC packets to be sent for the cur-
rent applicable coding block. Finally, some consideration might be
given to using the NACKing history of receivers to weight their selection of NACK backoff timeout intervals. For example, if a receiver has historically been experiencing the greatest degree of loss, it may promote itself to statistically NACK sooner than other receivers. Note this assumes there is some degree of correlation over successive intervals of time in the loss experienced by receivers. This adjustment of backoff timeout selection may require the creation of an "early NACK" slot for these historical NACKers. This additional slot in the NACK backoff window will result in a longer repair cycle process which may not be desirable for some applications. The resolution of these trade-offs may be dependent upon the protocol’s target application set or network.

Interface Description

Inputs:

1) Group greatest round trip time estimate (GRTT).
2) Group size estimate.
3) Application-defined bound on backoff timeout period.
4) NACKs from other receivers.
5) Pending repair indication from sender (may be forwarded NACKs).
6) Current sender transmission sequence position.

Outputs:

1) Yes/no decision to generate NACK message upon backoff timer expiration.

3.2.5 Sender NACK Processing and Repair Response

Upon reception of a repair request from a receiver in the group, the sender will initiate a repair response procedure. The sender may wish to delay transmission of repair content until it has had sufficient time to accumulate potentially multiple NACKs from the receiver set. This allows the sender to determine the most efficient repair strategy for a given transport stream/object or FEC coding block. Depending upon the approach used, some protocols may find it beneficial for the sender to provide an indicator of pending repair transmissions as part of the its current transmitted message content. This can aid some NACK suppression mechanisms. Alternatively, in the case of unicast feedback from the receiver set, it may be useful for the sender to forward (via multicast) superseding NACK messages to the group to allow for NACK suppression when there is not multicast connectivity among the receiver set.
When FEC is used, it is beneficial that the sender transmit previously untransmitted parity content whenever possible. This maximizes the receiving nodes’ ability to reconstruct the entire transmitted content from their individual subsets of received messages.

The transmitted object and/or stream content will be marked with monotonically increasing sequence numbers (within a reasonably large ordinal space). If the sender observes the discipline of transmitting repair for the earliest content first, the receivers can use a strategy of withholding repair requests for later content until the sender once again returns to that point in the object/stream transmission sequence. This can increase overall message efficiency among the group and help work to keep repair cycles relatively synchronized without dependence upon strict timing. This also helps minimize the buffering requirements of receivers and senders and reduces redundant transmission of data to the group at large.

Interface Description

Inputs:

1) Receiver NACKs
1) Group timing information

Outputs:

1) Repair messages (FEC and/or Data content retransmission)

3.3 Group "Join" Policies/ Procedures

Consideration should be given to how new receivers join a group (perhaps where reliable transmission is already in progress) and begin NACKing for any repair needs. If this is unconstrained, the dynamics of group membership may impede the application’s ability to meet its goals progressing the transmission of data. Policies limiting the opportunities at which receivers begin participating in the NACK process may be used to achieve the desired behavior. For example, it may be beneficial if receivers only attempt reliable reception from a newly-heard sender when upon non-repair transmissions of data in the first FEC block of an object or logical portion of a stream. The sender may also implement policies limiting which receivers from which it will accept NACK requests, but this may be prohibitive for scalability reasons in some situations. In some types of bulk transfer applications (and for potential interactive applications), it may alternatively desirable to have a looser transport synchronization policy and rely upon session management mechanisms to control group dynamics which may
result in poor behavior.

Inputs:

1) Current object/stream data/repair content and sequencing identifiers from sender transmissions.

Outputs:

1) Receiver yes/no decision to begin receiving and NACKing for reliable reception of data

### 3.4 Reliable Multicast Member Identification

In a NORM protocol (or other multicast protocols) where there is the potential for multiple sources of data, it is necessary to provide some mechanism to uniquely identify the sources (and possibly some or all receivers in some cases) within the group. Identity based on arriving packet source addresses is insufficient for several reasons. These reasons include routing changes for hosts with multiple interfaces which result different packet source addresses for a given host over time, network address translation (NAT) or firewall devices, or other transport/network bridging approaches. As a result, some type of unique source identifier (sourceId) field should be present in packets transmitted by reliable multicast session members.

### 3.5 Data Content Identification

The data and repair content transmitted by a NORM sender requires some form of identification in the protocol header fields. This identification is required to facilitate the reliable NACK-oriented repair process. These identifiers will be used in NACK messages generated.

There are two very general types of data which may comprise bulk transfer session content. These data types are static objects of finite size and continuous non-finite streams. A given application may wish to reliably multicast data content using either one or both of these data models. While it may be possible for some applications to further generalize this model and provide mechanisms to encapsulate static objects as content embedded within a stream, there are advantages to many applications to provide distinct support for static bulk objects and messages with the context of a reliable multicast session. These applications may include content caching servers, file transfer or collaborative tools with bulk content. Applications with requirements for these static object types can then take advantage of transport layer mechanisms.
(i.e. segmentation/reassembly, caching, integrated forward error correction coding, etc) rather than being required to provide their own mechanisms for these functions at the application layer.

As noted, some applications may alternatively desire to transmit bulk content in the form of one or more streams of non-finite size. Example streams include continuous quasi-realtime message broadcasts (e.g. stock ticker) or some content types which are part of collaborative tools or other more complex applications. And, as indicated above, some applications may wish to encapsulate other bulk content (e.g. files) into one more streams within a multicast session. Additionally, multiple streams can allow for parallelized transmission within the context of a single multicast session.

The components described within this building block draft document are envisioned to be applicable to both of these models with the potential for a mix of both types within a single multicast session. To support this requirement, the normal data content identification should include a field to uniquely identify the object or stream <objectId> within some reasonable temporal or ordinal interval. Note that it is _not_ expected that this data content identification will be globally unique. It is expected that the object/stream identifier will be unique with respect to a given sender within the reliable multicast session and during the time that sender is supporting a specific transport instance of that object or stream.

Since the "bulk" object/stream content will generally require segmentation, some form of segment identification <segmentId> must also be provided. This segment identifier will be relative to any object or stream identifier which has been provided. Thus, in some cases, NORM protocol instantiations may be able to receive transmissions and request repair for multiple streams and one or more sets of static objects in parallel. For protocol instantiations employing FEC the <segmentId> portion of the data content identifier may consist of a logical concatenation of a coding block identifier <blockId> and identifier for the specific data or parity segment of the code block. The RMT FEC Building Block (currently "draft-ietf-rmt-bb-fec-04.txt") provides a standard message format for identifying FEC transmission content.

Additionally, flags to determine the usage of the content identifier fields (e.g. stream vs. object) may be applicable. Flags may also serve other purposes in data content identification. It is expected that any flags defined will be dependent upon individual protocol instantiations.

In summary, the following data content identification fields may be
required for NORM protocol data content messages:

1) Source node identifier (<senderId>)
2) Object/Stream identifier (<objectId>), if applicable.
3) FEC Block identifier (<blockId>), if applicable.
4) Segment identifier (<segmentId>)
5) Flags to differentiate interpretation of identifier fields or identifier structure which implicitly indicates usage.
6) Additional FEC transmission content fields per FEC Building Block

These fields have been identified since any generated NACK messages will use these identifiers in requesting repair or retransmission of data. NORM protocols are expected to greatly benefit from interaction with other reliable multicast building blocks (Generic Router Assist (GRA), in particular) and those other building blocks will need to appropriately consider these anticipated requirements.

3.6 Forward Error Correction

Multiple forward error correction (FEC) approaches have been identified which can provide great performance enhancements to the repair process of NACK-oriented and other reliable multicast protocols [Metzner84, Macker97]. NORM protocols can reap additional benefits since FEC-based repair does not generally require explicit knowledge of repair content within the bounds of its coding block size (in packets).

Generally, repair packets generated using FEC algorithms with good erasure filling properties (e.g. Reed-Solomon) will be transmitted only in response to NACK repair requests from receiving nodes. However, there are benefits in some network environments for transmitting some predetermined quantity of FEC repair packets multiplexed with the regular data segment transmissions [Gossink98]. This can reduce the amount of NACK traffic generated with relatively little overhead cost when group sizes are very large or the network connectivity has a large delay*bandwidth product with some nominal level of expected packet loss. While the application of FEC is not unique to NORM, these sorts of requirements may dictate the types of algorithms and protocol approaches which are applicable.

A specific issue related to the use of FEC with NORM is the mechanism used to identify which portion(s) of transmitted data content to which specific FEC packets are applicable. It is expected that FEC algorithms will be based on generating a set of parity repair packets for a corresponding block of transmitted data packets. Since data content packets are uniquely identified by the
concatenation of <sourceId/objectId/blockId/segmentId> during transport, it is expected that FEC packets will be identified in a similar manner. The RMT FEC Building Block specification (currently "draft-ietf-rmt-bb-fec-04.txt") provides detailed recommendations concerning application of FEC and standard formats for related reliable multicast protocol messages.

3.7 Round-trip Timing Collection

The measurement of packet propagation round-trip time (RTT) among members of the group is required to support NACK suppression algorithms, timing of sender commands or certain repair functions, and congestion control operation. The nature of the round-trip information collected is dependent upon the type of interaction among the members of the group. In the case where only "one-to-many" transmission is required, it may be necessary that only the sender require RTT knowledge of the greatest RTT (GRTT) among the receiver set and/or RTT knowledge of only a portion of the group. Here, the GRTT information might be collected in a reasonably scalable manner. For congestion control operation, it is possible that RTT information may be required by each receiver in the group. In this case, an alternative RTT collection scheme may be utilized where receivers collect individual RTT measurements with respect to the sender and advertise them (or a competed subset) to the group or sender. Where it is likely that exchange of reliable multicast data will occur among the group on a "many-to-many" basis, there are alternative measurement techniques which might be employed for increased efficiency [Ozdemir99]. And in some cases, there might be absolute time synchronization among hosts which may simplify RTT measurement. There are trade-offs in multicast congestion control design which need further consideration before a universal recommendation on RTT (or GRTT) measurement can be specified. Regardless of how the RTT information is collected (and more specifically GRTT) with respect to congestion control or other requirements, the sender will need to advertise its current GRTT estimate to the group for timing of the

3.7.1 One-to-Many Sender GRTT Measurement

The goal of this form of RTT measurement is for the sender to learn the GRTT among the receivers who are actively participating in NORM operation. The set of receivers participating in this process may be the entire group or some subset of the group determined from another mechanism within the protocol instantiation. An approach to collect this GRTT information follows.

The sender periodically polls the group with a message (independent or "piggy-backed" with other transmissions) containing a <sendTime>
timestamp relative to an internal clock at the sender. Upon reception of this message, the receivers will record this <sendTime> timestamp and the time (referenced to their own clocks) at which it was received <recvTime>. When the receiver provides feedback to the sender (either explicitly or as part of other feedback messages depending upon protocol instantiation specification), it will construct a "response" using the formula:

\[ <\text{grttResponse}> = <\text{sendTime}> + (<\text{currentTime}> - <\text{recvTime}>) \]

where the <sendTime> is the timestamp from the last probe message received from the source and the (<currentTime> - <recvTime>) is the amount of time differential since that request was received until the receiver generated the response.

The sender processes each receiver response by calculating a current RTT measurement for the receiver from whom the response was received using the following formula:

\[ <\text{receiverRtt}> = <\text{currentTime}> - <\text{grttResponse}> \]

During the each periodic GRTT probing interval, the source keeps the peak round trip estimate from the set of responses it has received. The GRTT estimate should be filtered to be conservative towards maintaining an estimate biased towards the greatest receiver RTT measurements received. A conservative estimate of GRTT maximizes the efficiency redundant NACK suppression and repair aggregation. The update to the source’s ongoing estimate of GRTT is done observing the following rules:

1) If a receiver’s response round trip calculation is greater than the current GRTT estimate AND current peak, the response value is immediately fed into the GRTT update filter given below. In any case, the source records the "peak" receiver RTT measurement for the current probe interval.

2) At the end of the response collection period (i.e. the GRTT probe interval), if the recorded "peak" response is less than the current GRTT estimate AND this is the third consecutive collection period with a peak less than the current GRTT estimate the recorded peak is fed into the GRTT update. (Implicitly, Rule #1 was applied otherwise so no new update is required).

3) At the end of the response collection period, the peak tracking value is set to either ZERO if the "peak" is greater than or equal to the current GRTT estimate (i.e. Already incorporated into the filter under Rule #1) or kept
the same if its value is less than the current GRTT estimate AND was not yet incorporated into the GRTT update filter according to Rule #2. Thus for decreases in the source’s estimate of GRTT, the "peak" is tracked across three consecutive probe intervals. The current MDP implementation uses the following GRTT update filter to incorporate new peak responses into the GRTT estimate:

```plaintext
if (peak > current_estimate)
    current_estimate = 0.25 * current_estimate + 0.75 * peak;
else
    current_estimate = 0.75 * current_estimate + 0.25 * peak;
```

This update method is biased towards maintaining an estimate of the worst-case round trip delay. The reason the GRTT estimate is reduced only after 3 consecutive collection periods with smaller response peaks is to be conservative where packet loss may have resulted in lost response messages. And then the reduction is additionally conservatively weighted using the averaging filter from above.

The GRTT collection period (i.e. period of probe transmission) could be fixed at a value on the order of that expected for group membership and/or network topology dynamics. For robustness, more rapid probing could be used at protocol startup before settling to a less frequent, steady-state interval. Optionally, an algorithm may be developed to adjust the GRTT collection period dynamically in response to the current GRTT estimate (or variations in it) and to an estimation of packet loss. The overhead of probing messages could then be reduced when the GRTT estimate is stable and unchanging, but be adjusted to track more dynamically during periods of variation with correspondingly shorter GRTT collection periods.

In summary, although NORM repair cycle timeouts are based on GRTT, it should be noted that convergent operation of the protocol does not _strictly_ depend on accurate GRTT estimation. The current mechanism has proved sufficient in simulations and in the environments in which NORM-like protocols have been deployed to date. The estimate provided by the algorithm tracks the peak envelope of actual GRTT (including operating system effect as well as network delays) even in relatively high loss connectivity. The steady-state probing/update interval may potentially be varied to accommodate different levels of expected network dynamics in different environments.

3.7.2 One-to-Many Receiver RTT Measurement

(TBD – Receivers "ping" sender for RTT measurement, and then
receivers competitively (worst case RTT metric) advertise their measurements to the sender and optionally group so the sender can determine GRTT ... Sender should still robustly advertise its current GRTT knowledge to the group so group can use appropriate timing)

3.7.3 Many-to-Many RTT Measurement

(TBD - Describe approach based on Ozdemir99, if appropriate/applicable?)

3.7.4 Sender GRTT Advertisement

To facilitate deterministic NORM protocol operation, the sender should robustly advertise its current estimation of GRTT to the receiver set. Common, robust knowledge of the sender’s current operating GRTT estimate among the group will allow the protocol to progress in its most efficient manner. The sender’s GRTT estimate can be robustly advertised to the group by simply embedding the estimate into all pertinent messages transmitted by the sender. The overhead of this can be made quite small by quantizing (compressing) the GRTT estimate to a single byte of information. The following C-language function algorithm allows this to be done over a wide range of GRTT values while maintaining a greater range of precision for small GRTT values and less precision for large values:

```c
unsigned char QuantizeGrtt(double grtt)
{
    if (grtt > 1.0e3)
        grtt = 1.0e3;
    else if (grtt < 1.0e-06)
        grtt = 1.0e-06;
    if (grtt < 3.3e-05)
        return ((unsigned char)(grtt * 1.0e06) - 1);
    else
        return ((unsigned char)(ceil(255.0 -
                                  (13.0 * log(1.0e03/grtt)))));
}
```

Note that this function is useful for quantizing GRTT times in the range of 1 microsecond to 1000 seconds. Of course, NORM protocol implementations may wish to further constrain advertised GRTT estimates (e.g. limit the maximum value) for practical reasons.

3.8 Group Size Determination/Estimation

(TBD) Group size may be approximated from the density of feedback
messages which follow the exponentially weighted random backoff. NORM_NACK messages might be used during normal protocol operation or a bootstrap procedure can be created to obtain an initial size estimation and track group size with receiver join/leave dynamics. This might also be combined with congestion control feedback collection. The details of this are TBD.

3.9 Congestion Control Operation

(TBD - A NACK-oriented protocol may place limitations/requirements on collection of information to facilitate congestion control of senders. There are a number of specific issues of TCP-Friendly Multicast Congestion Control (TFMCC) which must be addressed.)

3.10 Router/Intermediate System assistance

(TBD - NACK-oriented protocols can potentially benefit greatly from router assistance. In particular, additional NACK suppression would be beneficial (This may impact how synchronized receiver NACK cycles are, sender advertisement of NACK-cycle parameters (i.e. GRTT, group size, etc), NACK content, others)

3.11 Additional protocol mechanisms

(TBD - e.g. positive acknowledgement collection, performance statistics collection, session management, etc)

4.0 Security Considerations (TBD)

5.0 References


6.0 Authors’ Addresses

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

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

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

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