ARM Transport - Active Reliable Multicast Transport Protocol

Dmitri Kondratiev

dkondr@bigfoot.com

Table of Contents

Abstract

This document provides design overview of the ARM - Active Reliable Multicast Transport protocol. ARM uses IP Multicast to reliably deliver bulk data to a group of recivers engaged in one-to-many or many-to-many session. ARM design to some extent follows the one of NACK-Oriented Reliable Multicast Protocol (NORM) and also borrows some ideas from StarBurst Multicast File Transfer Protocol (MFTP). In essence ARM Transport is a simplified version of NORM protocol enchansed with similar to MFTP mechanism used for session anouncement and data transfer in separate multicast groups.

Protocol Description

ARM Transport Service Model

The protocol design is principally driven with the assumption of a single sender transmitting bulk data content to a group of receivers. However, the protocol does provide for multiple senders to coexist within the context of a ARM Session. In initial implementations of this protocol, it is anticipated that multiple senders will transmit independently of one another and receivers will maintain state as necessary for each independent sender.

As well as [NORM] ARM provides for three types of bulk data content objects to be reliably transported. These types include static computer memory data content (ARM_OBJECT_DATA), computer storage files (ARM_OBJECT_FILE), and non-finite streams of continuous data content (ARM_OBJECT_STREAM). The distinction between ARM_OBJECT_DATA and ARM_OBJECT_FILE is simply to provide a "hint" to receivers in ARM Sessions serving multiple types of content as to what type of storage should be allocated for received content (i.e. memory or file storage). Other than that distinction, the two are identical, providing for reliable transport of finite units of content.

The static data and file services are anticpated to be useful for multicast-based cache applications with the ability to reliably provide transmission / repair of a large set of static data. Other types of static data / file "casting" services might make use of these transport object types, too.

ARM_OBJECT_STREAM type may be used for reliable messaging or other unbounded, perhaps dynamically produced content. The ARM_OBJECT_STREAM provides for reliable transport analogous to that of the Transmission Control Protocol (TCP) although ARM 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 ARM protocol allows for a small amount of "out-of-band" data (ARM_INFO). This readilyavailable "out-of-band" data allows multicast receivers to quickly and efficiently determine the nature of the bulk content (data, file, or stream) being transmitted to allow application-level control of the receiver node’s participation in the current transport activity. This allows the protocol to be flexible with minimal precoordination among senders and receivers.

Reliable file data transfer is based on "Shared Recovery with Multicast NAKs and Retransmissions" mechanism. To avoid the NAK implosion receivers share recovery responsibilities. When a receiver detects missing data and sends a NAK, another receiver that has received the data can potentially respond to the NAK by retransmitting the missing data.

ARM uses NAKs and retransmission requests sent by receivers to a multicast group address. As a result all ARM receivers equaly participate in repairs. Here ARM dufferes from NORM which uses unicast links to send these requests from receivers to a single sender which is the only one to provide all repairs. ARM algorithm used

to send NAKs and retransmission requests resembles Host Membership Queries in IGMP.

ARM Multicast NAKs and Retransmissions Algorithm in BSG cloud.

Algorithm

In BSG cloud, BSGs of all levels (level 1 and 2) play the same role of receivers sending both NACKs and answering retransmission requests as well. Thus, no matter what level, all BSGs are equal in that they perform the same roles in ARM File protocol. This assumes some file storage and / or file cache on every BSG to retransmit data.

Below the description of ARM basic retransmission algorithm follows with 'receiver' used in place of 'BSG' and vice versa.

- 1. When a receiver detects missing data, it starts a (random) timer.
- 2. When the timer expires and receiver hasn't yet received a NAK for the missing data from the group (multicast) address, it sends a NAK to the group (with a limited scope).
- 3. All receivers within the scope see the NAK, start a (random) timer, and watch for the requested data to be retransmitted.
- 4. If retransmitted data is seen, receivers stop the timer. Else, if their timer expires they retransmit the data to the multicast group address (with a limited scope) if they have it, or send another NAK if not.

Advantages

- Off-load the recovery burden from the sender, and provide faster recovery (since NAK and retransmitted data have a shorter distance to travel).
- Other receivers that also missed the same data can see the NAKs, and don't need to send them to benefit from the retransmitted data.
- Reduced recovery latency.
- Reduced network traffic in the path between sender and receiver.

Caveats.

- May increase overall network traffic.
- Receivers must be capable of gracefully handling redundant data.
- Retransmission must be done in the limited scope.
- Data delivery is delayed because of the time-outs to send NACKs.
- Will not work on network topologies that don't support many-to-many multicast.

Message Types and Header Definitions

There are two primary classes of ARM messages: messages generated by the sender of reliable multicast traffic and messages generated by receivers. These are described in corresponding sections below after the portion of ARM Message header common to all ARM messages is described.

The multi-octect header fileds contain data in network (big-endian) byte order.

ARM Common Message Header

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

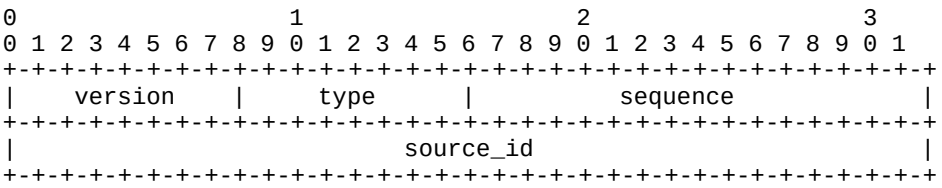


Figure 1. Common Message Header

The "version" field is a 8-bit value indicating the protocol version number. Currently, ARM implementations SHOULD ignore received messages with a different protocol version number. This number is intended to indicate and distinguish upgrades of the protocol which may be non-interoperable. The message "type" field is a 8-bit value indicating the ARM protocol message type. These types are defined as follows:

Message	Value
ARM_INFO	1
ARM_DATA	2
ARM_CMD	3
ARM_NACK	4
ARM_ACK	5
ARM_REPORT	6

Figure 2. Message Types

The "sequence" field is a 16-bit value which is set by the message originator as a monotonically increasing number incremented with each ARM message transmitted. This value can be monitored by receiving nodes to detect packet losses in the transmission. Note that this value is NOT used to detect missing reliable data content, but is intended for use in an algorithm estimating raw packet loss for congestion control purposes. The size of this field is intended to be sufficient to allow detection of a reasonable range of packet loss within the expected delay-bandwidth product of expected network connections. The "source_id" field is a 32-bit value identifying the node which sent the message. A participant’s ARM node identifier (ArmNodeId) can be set according to the application needs but unique identifiers must be assigned within a single ArmSession. In some cases, use of the host IP address or a hash of it can sufice, but alternative methodologies for assignment and potential collision resolution of node identifiers within a multicast session need to be considered. For example, the "source identifier" mechanism defined in the RTPv2 specification [REF RTP] may be applicable to use for ARM node identifiers. At this point in time, the protocol makes no assumptions about how these unique identifiers are actually assigned.

Sender Messages

ARM_DATA

This is expected to be the predominant message type transmitted by ARM senders. These messages will contain data content for objects of type ARM_OBJECT_DATA, ARM_OBJECT_FILE, and ARM_OBJECT_STREAM.

A goal of the protocol design is to provide for parallel transmis- sion of different streams and data/file sets. ARM_DATA messages will generally consist of original data content of the application data being transmitted.

The payload size of these messages will depend on the content being transfered. In case of files, most probably, all messages (except maybe the last one) will have the fixed payload size that receivers may calculate using the formula:

```
arm_payload_size = arm_packet_size - arm_headers_size
```

This allows receivers to allocate appropriate buffering resources and to determine other information in order to properly process received data messaging.

In case of sending object with different message sizes additional messages may be used (ARM_INFO or ARM_CMD) to advertize message size as needed.

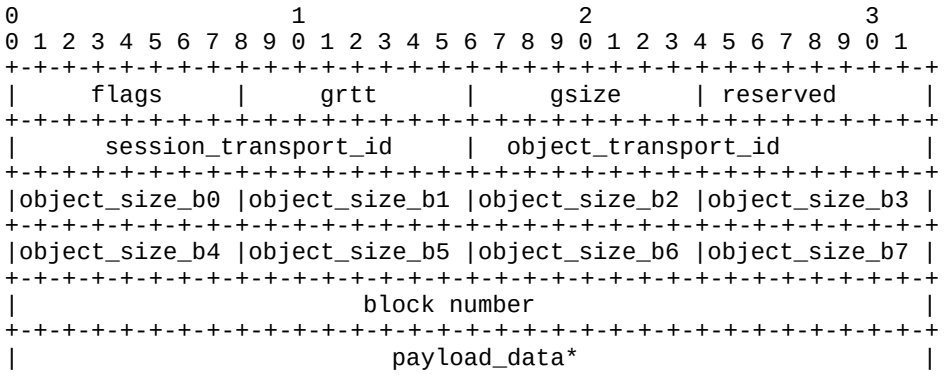


Figure 3. ARM_DATA Header

The "flags" field contains a number of different binary flags pro- viding information and hints regarding how the receiver should han- dle the identified object. Defined flags in this field include:

Table 1. ARM_DATA Flags

Flag	Value	Purpose
ARM_FLAG_REPAIR	0x01	Indicates message is a repair transmission
ARM_FLAG_INFO	0x02	Indicates availability of ARM_INFO for object.
ARM_FLAG_UNRELIABLE	0x04	Indicates that repair transmissions for the specified object will be unavailable. (One-shot, best effort transmission)

ARM_FLAG_FILE	0x8	Indicates object is "file-based" data (hint to use disk storage for reception)
ARM_FLAG_STREAM	0x10	Indicates object is of type ARM_OBJECT_STREAM

The ARM_FLAG_REPAIR flag is set when the associated transmission is a repair transmission. This information can be used by receivers to facilitate a join policy where it is desired that newly joining receivers only begin participating in the NACK process upon receipt of new "fresh" data. The ARM_FLAG_INFO flag is set only when the optional ARM_INFO content is available for the associated object. Thus, receivers will NACK for retransmission of ARM_INFO only when it is available. The ARM_FLAG_UNRELIABLE flag is set when the sender wishes to transmit and object with "best effort" delivery only and will not supply repair transmissions for the object. The ARM_FLAG_FILE flag can be set as a "hint" from the sender that the associated object should be stored in nonvolatile storage. The ARM_FLAG_STREAM flag is set when the identified object is of type ARM_OBJECT_STREAM. Note that the "object_size" field is no longer applicable (Another use for this field for "stream" objects may be determined as this capability is designed).

The "grtt" field contains a non-linear quantized representation of the sender's current estimate of group round-trip time (GRTT). This value is used to control timing of the NACK repair process and other aspects of protocol operation as described in this document.

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

The "session_transport_id" field is a monotonically and incrementally increasing value assigned by a sender to the session being transmitted. Transmissions and repair requests related to that session use the same "session_transport_id" value. For sessions of very long or indefinite duration, the "session_transport_id" field may be repeated, but it is presumed that the 16-bit field size provides an adequate enough sequence space to prevent temporary object confusion amongst receivers and sources (i.e. receivers SHOULD resynchronize with a server when receiving session sequence identifiers sufficiently out-of-range with the current state kept for a given source).

The "object_transport_id" field is a monotonically and incrementally increasing value assigned by a sender to the object being transmitted in context of one session. Transmissions and repair requests related to that session and object use the same "session_transport_id","object_transport_id" value. During the course of its transmission within a ARM session, an object is uniquely identified by the concatenation of the sender "node_id" and the given "session_transport_id","object_transport_id". Note that ARM_INFO messages associated with the identified object carry the same "object_transport_id" value.

The 64-bit "object_size" field indicate the total size of the object (in bytes). Object size in this field is encoded in network byte order (big-endian) with the most significant byte in the lowest address and other octets following in reverse order of significance (object_size_b0, object_size_b1 ... object_size_b7).

This information is used by receivers to determine storage requirements and / or allocate storage for the received object. Receivers with insufficient storage capability may wish to forego reception (i.e. not NACK for) of the indicated object. The "object_size" fields are not applicable for objects of type ARM_OBJECT_STREAM. (Note: The "object_size" fields _may_ be defined to serve an alternative use in this case).

The 32-bit "block number" field contains the number of data block sent in this message. Block number is relative to "object_transport_id", which means that this number starts from 0 for every new ARM_OBJECT sent.

The "payload_data" field contains original data, where:

```
arm_payload_size = arm_packet_size - arm_headers_size
```

ARM_INFO

The ARM_INFO message is used to convey _optional_ "out-of-band" context information for objects transmitted. An example may be MIME type information for the associated file, data, or stream object. Receivers might use this information to make a decision as whether to participate in reliable reception of the associated object. Each ArmObject may have an independent unit of ARM_INFO associated with it. ARM_DATA messages contain a flag to indicate the availability of ARM_INFO for a given NormObject. ARM receivers may NACK for retransmission of ARM_INFO when they have not received it for a given NormObject. The size of the ARM_INFO content is limited to that of a single ARM message for the given sender. This atomic nature allows the ARM_INFO to be rapidly and efficiently repaired within the ARM reliabel transmission process.

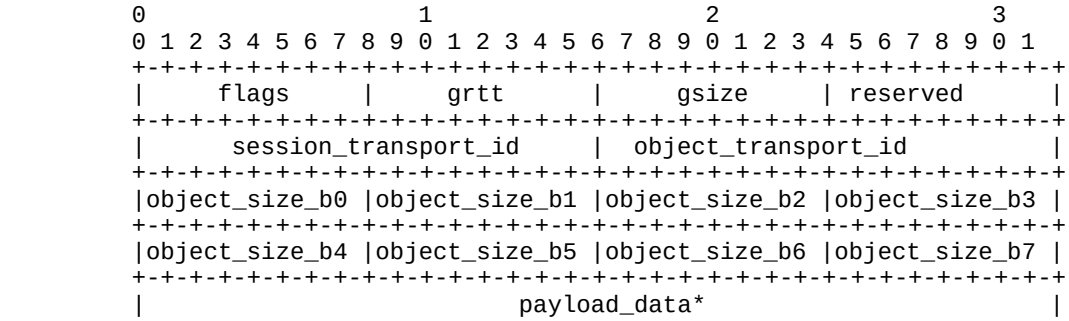


Figure 4. ARM_INFO Header

The "flags", "grtt", "gsize", "session_transport_id", "object_transport_id" and "object_size" fields carry the same information and serve the same purpose as with ARM_DATA messages. These values allow the receiver to prepare buffering, etc for further transmissions from the sender if this is the first message received. The "payload_data" field contains application-defined content which can be used by the receiver application for various purposes.

ARM_CMD

ARM_CMD messages are transmitted by senders to perform a number of different protocol functions. This includes round-trip timing collection, potential congestion control functions, synchronization of receiver NACKing "windows", notification of sender status and other core protocol functions. A core set of ARM_CMD messages will be enumerated. A range of command types will remain undefined for potential application-specific usage. Some ARM_CMD types (possibly including application-defined commands) may have some dynamic content attached. This content will

be limited to a single ARM message to retain the atomic nature of commands. The ARM_CMD message begins with a common header, following the usual ARM message common header. The header format is defined as:

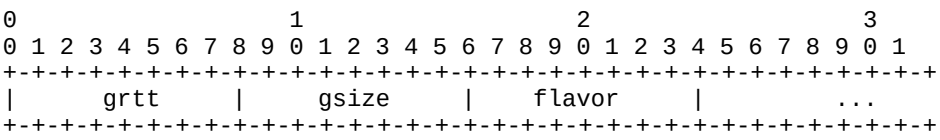


Figure 5. ARM_CMD Common Header

The "grtt" and "gsize" fields provide the same information and serve the same purpose as with ARM_DATA and ARM_INFO messages. The "flavor" field indicates the type of command to follow. The command flavors (types) include:

Table 2. ARM_CMD Flavors

Command	Flavor Value	Purpose
ARM_CMD(FLUSH)	1	Indicates sender temporary or permanent end-of-transmission. (Assists in robustly initiating outstanding repair requests from receivers).
* ARM_CMD(SQUELCH)	2	Advertisement of current repair window in response to out-of-range NACKs.
ARM_CMD(ACK_REQ)	3	Requests positive acknowledgement from a list of receivers.
* ARM_CMD(REPAIR_ADV)	4	Advertise sender's aggregated NACKs for suppression of unicast feedback.
* ARM_CMD(CC)	5	Probe possibly used for explicitly collecting congestion control feedback.
ARM_CMD(APPLICATION)	6	Robustly repeated application-defined command which can temporarily preempt ARM data transmission.

* Currently unsupported ARM_CMD flavors

ARM_CMD(FLUSH)

The ARM_CMD(FLUSH) command is sent when the sender reaches the end of any data content and pending repairs it has queued for transmission. This command is repeated once per 2*GRTT to excite the receiver set for any outstanding repair requests for data up to and including the point indicated by the FLUSH message. The number of repeats is equal to ROBUST_FACTOR. The greater this number, the higher the probability that all applicable receivers will be excited for repair requests (NACKs)

and the corresponding NACKs are delivered to the sender. If a NACK message interrupts the flush process, the sender will re-initiate the flush process when repair transmissions are completed. Note that receivers also employ a timeout mechanism to self-initiate NACKing when a sender is determined to have gone "idle". This inactivity timeout is related to 2*GRTT*ROBUST_FACTOR and will be discussed more later. With a sufficient ROBUST_FACTOR value, data content is delivered with a high assurance of reliability. The penalty of a large ROBUST_FACTOR value is potentially excess sender ARM_CMD(FLUSH) transmissions and a longer timeout for receivers to self-initiate a terminal NACK process. The format of the ARM_CMD(FLUSH) message (in addition to the ARM message common header) is:

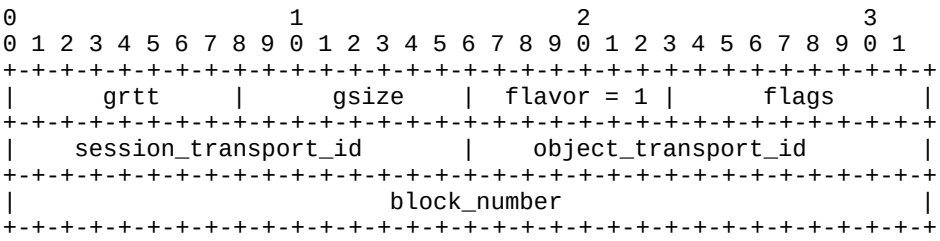


Figure 6. ARM_CMD(FLUSH) Message

In addition to the common ARM_CMD "grtt", "gsize", and "flavor" fields, the ARM_CMD(FLUSH) message contains fields to identify the current logical transmit position of the sender. These fields consist of "session_transport_id", "object_transport_id" and "block_num ber". These fields are interpreted in the same manner as the fields of the same names in the ARM_DATA message type. Receivers are expected to check their completion state and initiate the NACK repair process if they have outstanding repair needs up through this transmission point. If the receivers have no outstanding repair needs, no response is generated.

A single "flag" value is currently defined:

```
ARM_FLUSH_FLAG_EOT = 0x01
```

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

ARM_CMD(ACK_REQ)

The ARM_CMD(ACK_REQ) message is used by the sender to request acknowledgement from a specified list of receivers. This message serves in a lightweight positive acknowledgement mechanism which can be optionally employed by the reliable multicast application to deterministically determine that watermark points in the reliable transmission have been achieved by specific receivers. The format of the ARM_CMD(ACK_REQ) message (in addition to the ARM message common header) is shown below.

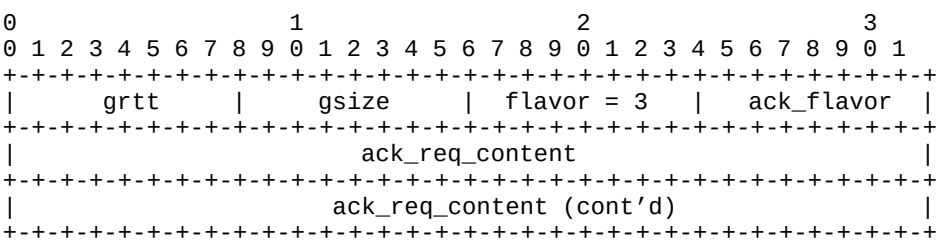


Figure 7. ARM_CMD(ACK_REQ) Message

The ARM_CMD(ACK_REQ) Message consists of "grtt", "gsize", and "flavor" fields as with other ARM_CMD messages. Then an "ack_flavor" field is specified followed by 64-bits of "ack_flavor" specific content and a list of ArmNodeIds which are expected to explicitly respond to the acknowledgement request. The "wildcard" ArmNodeId may be listed when a response is expected from all receivers.

The "ack_flavor" field is used to indicate the interpretation of the 64-bit ack_req_content" field space following the "ack_flavor" field. The following "ack_flavor" values are defined:

ACK Type	Flavor Value	Purpose
ARM_ACK(WATERMARK)	1	Request for acknowledgement of reliable reception of watermark transmission point.
ARM_ACK(SESSION)	2	Request for acknowledgement of reliable reception of complete session
ARM_ACK(RTT)	3	Sender timestamp information and optional request for explicit RTT collection response

ARM_ACK(WATERMARK) and ARM_ACK(SESSION)

The ARM_ACK(WATERMARK) identifies a watermark point in the sender's reliable transmission and explicitly requests positive acknowledgement from a portion of the receiver set.

The ARM_ACK(SESSION) also identifies a watermark point - end of session - in the sender's reliable transmission and explicitly requests positive acknowledgement from receiver. This command indicates to reciever that no more data will be sent in the current session and session ends.

The only difference in format of the ARM_CMD(ACK_REQ(WATERMARK)) and ARM_CMD(ACK_REQ(SESSION)) message is the value of "ack_flavor" field. The format (in addition to the ARM common message header) is:

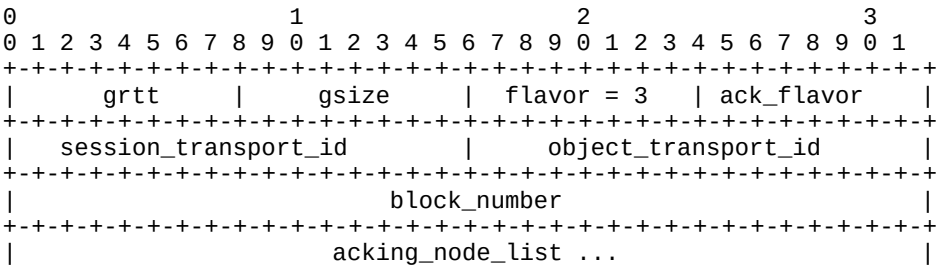


Figure 8. ARM_CMD(ACK_REQ(WATERMARK)) and and ARM_CMD(ACK_REQ(SESSION)) Messages

The "session_transport_id", "object_transport_id" and "block_number" are used to identify the watermark point for which the positive acknowledgement request applies. This watermark point is similar to that used in ARM_CMD(FLUSH) message.

It should be noted that all receivers are expected to treat the ACK_REQ command equivalently to a FLUSH command and appropriately initiate NACK repair cycles in response to any detected missing data up to the watermark point.

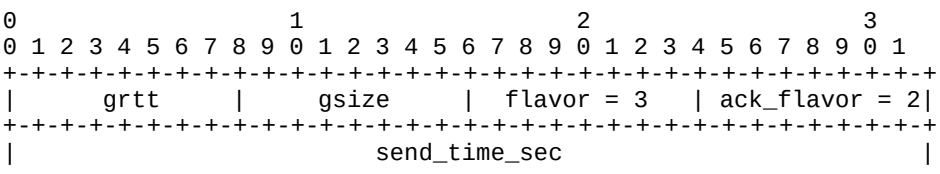
The "acking_node_list" field contains the current list of receiver ArmNodeIds which should reply with postive acknowledgement to this request. The packet payload length implies the length of the "acking_node_list" and its length is limited to the ArmSegmentSize. The ArmNodeIds are listed in network (Big Endian) order. The indicated receivers SHALL send a ARM_ACK message in response to this request IF they have no outstanding repair needs up to and including the watermark point. Note this does _not_ necessarily mean the receivers actually received all of the data, but simply that, for whatever reason (including the fact they may have already received the data or if the receiving application simply chose _not_ to receive the indicated data), they have no outstanding repair needs prior to the watermark point. Verification of actual received data content may also be accomplished by another means outside of this transport layer protocol. Receivers SHALL randomly spread their response to this request using a uniform distribution over 1 GRTT of time. Again, note the size of the included list is limited to the sender's ArmSegmentSize setting. Thus, multiple ARM_CMD(ACK_REQ) cycles may be required to achieve responses from all receivers specified. Also, the number of attempts to excite a response from a given receiver SHALL be limited to ROBUST_FACTOR. The ARM_CMD(ACK_REQ) is repeated at a rate of once per 2*GRTT. Note that the content of the attached ArmNodeId list will be dynamically updated as this process progresses and ACKs are received from the specified receiver set. The process SHALL terminate when all desired receivers have responded or the maximum number of attempts has been achieved. Note that repair requests can interrupt the positive acknowledgement process and the positive acknowledgment process will resume only when there are no pending repair transmissions up to the specified watermark point.

ARM_CMD(ACK_REQ(RTT))

The ARM_CMD(ACK_REQ(RTT)) is periodically transmitted by the sender to provide a reference point (a timestamp) so that receivers can calculate appropriate response content in ARM_NACK and ARM_ACK messages from which the sender can monitor and estimate the current GRTT. Currently, this reference is sent separately from other sender message and not included in every message because of the excessive overhead it may impose on data transmission. Generally, the GRTT is not expected to be so dynamic as to require rapid update. However, a technique is being investigated by the NORM author to provide a low overhead reference which could be attached to every sender transmission and used for the receiver response generation [REF].

This command may also potentially be leveraged to serve as part of ARM congestion control to periodically provide updated congestion control information and/or probing to the group. If this is the case, there will likely be sufficient content in this message that it merits a separate message rather than be periodically included in the overhead of other sender transmissions.

This command may also be extended to assume some responsibility in initializing and updating a group size estimator used to appropriately scale NACK suppression back-off timing, etc. For now, a minimal format is defined as a placeholder for this message. The format of the ARM_CMD(GRTT_REQ) message (in addition to the ARM message common header and the ARM_CMD common header) is:



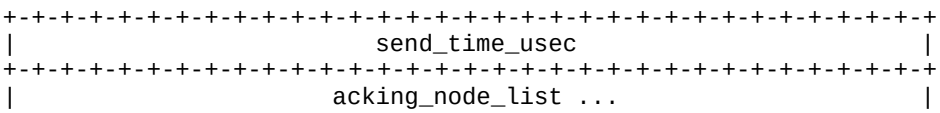


Figure 9. ARM_CMD(ACK_REQ(RTT)) Message

The "send_time" field is a precision timestamp indicating the time that the ARM_CMD(GRTT_REQ) message was transmitted. This consists of a 64-bit field containing 32-bits with the time in seconds ("sent_time_sec") and 32-bits with the time in microseconds ("send_time_usec") since some reference time the source maintains (usually 00:00:00, 1 January 1970). The ordering of the fields in Big Endian network order.

The "acking_node_list" again is a list of ArmNodeIds for receivers which should explicitly respond to the request. The "wildcard" NormNodeId may be listed when a response is expected from all receivers. When this "wildcard" response is elicited, the receiver set is expected to randomly spread their response over time, scaled as a function of the "grtt" and "gsize" values to avoid response implosion. The receivers corresponding to other listed specific ArmNodeIds are expected to respond immediately with an appropriate ARM_ACK message (unless a ARM_NACK message is already queued for response). Both ARM_NACK and ARM_ACK messages are designed to contain information to allow the sender to determine round trip times for responding receivers.

ARM_CMD(APPLICATION)

This command allows the ARM application to robustly transmit application defined content. The message is repeated ROBUST_FACTOR times at a rate of once per 2*GRTT. This rate of repeat allows the application to collect a response (if that is the application's purpose for the command) before it is repeated.

Some commands of this type may be sent in parallel with ongoing data while others preempts any existing transmission. In later case regular data object transmission is resumed when the repeated transmission of this message has completed.

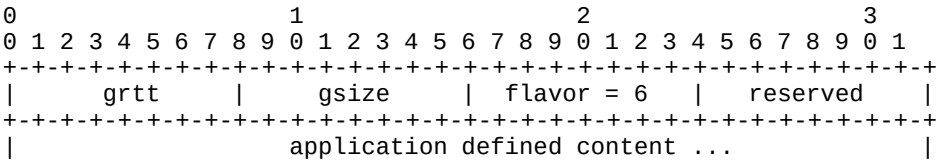


Figure 10. ARM_CMD(APPLICATION) Message

ARM_CMD(APPLICATION(SESSION_INFO))

This message contains current session info. Receivers may use information in this command to get additional properties of the session. At the moment of writing ARM_CMD(APPLICATION(SESSION_INFO)) contains information about file session and has the following format :

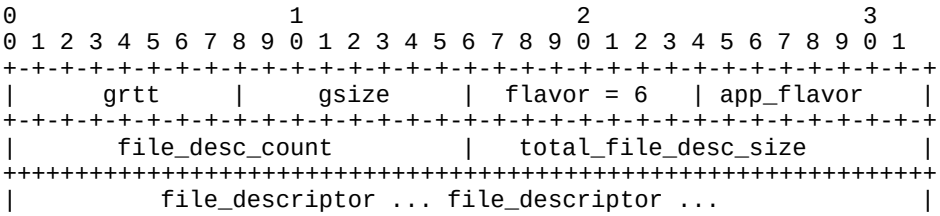


Figure 11. ARM_CMD(APPLICATION(SESSION_INFO)) Message

Where "app_flavor" defines ARM_CMD(APPLICATION) subtype and currently has a single value:

app_flavor	Value
ARM_CMD_APPLICATION_SESSION_INFO	1

The "file_desc_count" field contains a count of "file_descriptor" records in this message.

The "total_file_desc_size" fieled contains the total size (in bytes) of of all "file_descriptor" records in this message.

The "file_descriptor" is a record of the following format:

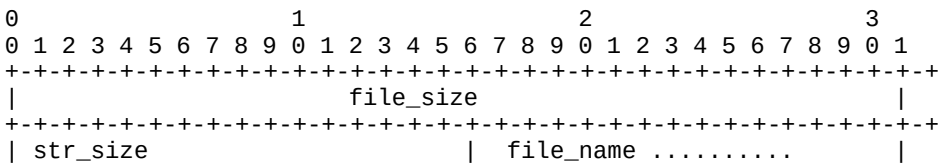


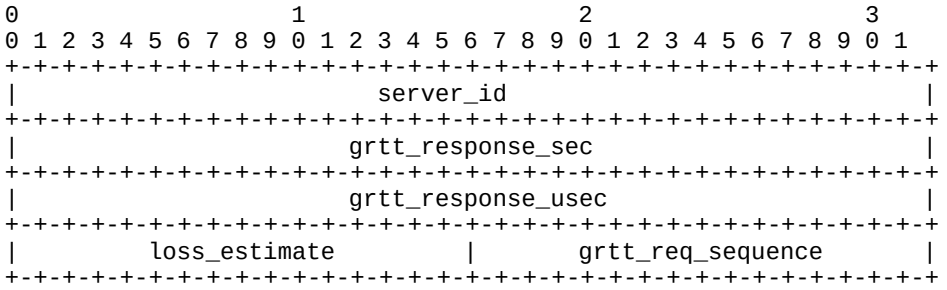
Figure 12. Session Info File Descriptor

Receiver Messages

ARM_NACK

The principal purpose of ARM_NACK messages will be for receivers to request repair of content via negative acknowledgement upon detection of incomplete data. ARM_NACKs will be transmitted according to the rules of NACK generation and suppression of the ARM NACK process. A goal for the content of these messages is to use a format which can be potentially used by compatible intermedi- ate systems [REF Generic Router Assist Building Block] to provide assistance in promoting protocol scalability and efficiency when available. ARM_NACK messages generated will also contain addi- tional content to provide feedback to sender(s) for purposes of round-trip timing collection, congestion control, etc.

ARM_NACK messages are transmitted by ARM receivers in response to the detection of missing data in the sequence of transmissions received from a particular source. The specific times and condi- tions under which receivers will generate and transmit these ARM_NACK messages are governed by the processes described in detail later in this document. The payload of ARM_NACK messages contains one or more "ObjectNACKs" for different objects and por- tions of those objects. In addition to the common message header the ARM_NACK messages contain the following fields:



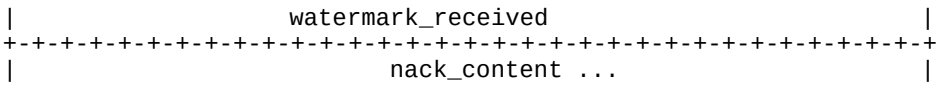


Figure 13. ARM_NACK Header

The "server_id" field identifies the source to which the ARM_NACK message is destined. Other sources should ignore this message. (Note that this another reason why multiple potential sources within an ARM session MUST have unique NormNodeIds).

The "grtt_response" fields contain a timestamp indicating the time at which the ARM_NACK was transmitted. The format of this timestamp is the same as the "send_time" field of the ARM_CMD(ACK_REQ(RTT)). However, note that the "grtt_response" timestamp is _relative_ to the "send_time" the source provided with the corresponding ARM_CMD(ACK_REQ(RTT)) command. The receiver adjusts the source's ARM_CMD(ACK_REQ(RTT)) "send_time" timestamp by the time differential from when the receiver received the ARM_CMD(ACK_REQ(RTT)) to when the ARM_NACK was transmitted to calculate the value in the "grtt_response" field. This is the "receive_to_response_differential" value in he following formula:

"grtt_response" = request "send_time" + receive_to_response_differential

The receiver may set the "grtt_response" to a ZERO value, to indicate that the it has not yet received a ARM_CMD(ACK_REQ(RTT)) command from the source and the source should ignore the grtt_response in this message.

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

"loss_estimate" = decimal_loss_fraction * 65535.0

The "grtt_req_sequence" field contains the loss sequence number identifier of the received ARM_CMD(ACK_REQ(RTT)) to which the response information in this ARM_NACK applies. This sequence number is the one from the ARM common message header for the applicable ARM_CMD(ACK_REQ(RTT)) command. This information can possibly assist the sender in congestion control operation.

The "watermark_received" field indicates the maximum "object_transport_id, block_number" received at the moment of sending this NACK. It has the following format:

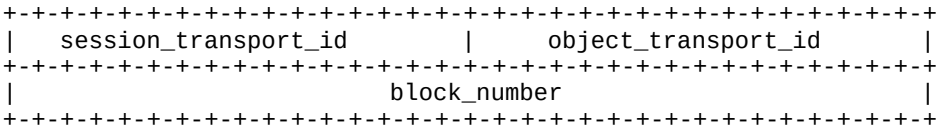
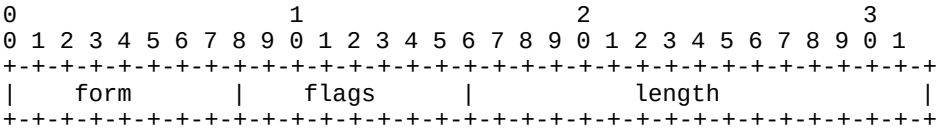


Figure 14. Watermark Received

The "nack_content" of the ARM_NACK message specifies the repair needs of this client pertaining to the indicated "server_id". The repair needs are specified as one or more lists of individual "items", "ranges" of identified ARM_DATA and/or ARM_INFO messages required for the receiver to complete reliable reception of objects being transmitted by the sender.

Each list of "items", "ranges" is specified with the following packet format:



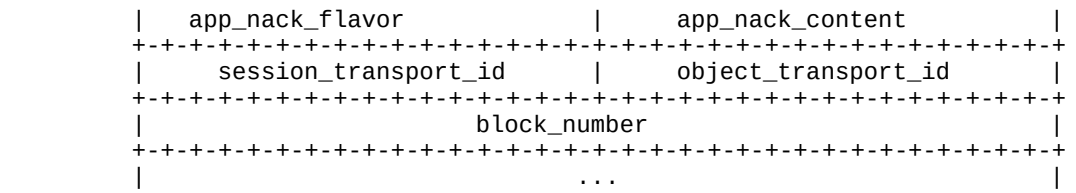


Figure 15. NACK Content

The "form" field indicates currently whether the NACK content that follows is a list of ARM_NACK_ITEMS or ARM_NACK_RANGES. Possible values for the "form" field include:

Form	Value
ARM_NACK_ITEMS	1
ARM_NACK_RANGES	2

When the list consists of individual ARM_NACK_ITEMS, each concatenation of "session_transport_id, object_transport_id, block_number" identifies an individual repair need of the NACKing receiver. When the list consists of ARM_NACK_RANGES, pairs of "session_transport_id, object_transport_id, block_number" sets are given to indicate the inclusive range of sender information needed by the receiver.

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

Flag	Value	Purpose
ARM_NACK_BLOCK	0x01	Indicates the entire listed block(s) are required as repair.
ARM_NACK_INFO	0x02	Indicates the object’s ARM_INFO is required as repair.
ARM_NACK_OBJECT	0x04	Indicates the entire listed object(s) are required as repair.
ARM_NACK_APP	0x08	Indicates NACK content defined by application

When the ARM_NACK_BLOCK flag is set, this indicates the receiver is missing the indicated block(s) for the indicated "object_transport_id".

When the ARM_NACK_INFO flag is set, this indicates the receiver is missing the ARM_INFO message for the indicated "object_transport_id". Note the ARM_NACK_INFO may be set in combination with the ARM_NACK_BLOCK or may be set alone.

When the ARM_NACK_OBJECT flag is set, this indicates the receiver is missing the entire ArmTransportObject referenced by the "object_transport_id". This implicitly includes any available ARM_INFO if applicable. The "block_number" field is ignored when this flag is set.

The "length" field is given (in bytes) to indicate the length of the list of ARM_NACK_ITEMS or ARM_NACK_RANGES. Multiple lists of NACK items and/or ranges may be concatenated together within a sin- gle ARM_NACK message.

When the ARM_NACK_APP flag is set, this indicates the receiver is missing _application defined_ NACK content which apllication may set in "app_nack_content" field according to the value of "app_nack_flavor". Also apllication may chose to use

the rest of the packet starting from "app_nack_content" when it needs space for additional NACK content.

The "app_nack_flavor" field is set to application defined subtype or flavor of NACK content. At the moment of this writing the only value for this field defined is:

nack_flavor	Value
ARM_NACK_APP_SESSION_INFO	1

The "app_nack_content" field may contain _application defined_ NACK content as required by "app_nack_flavor" type.

ARM_NACK_APP_SESSION_INFO message is sent by recivers to request retransmit of ARM_CMD(APPLICATION(SESSION_INFO)) message.

ARM_ACK

The basic operation of ARM transport will _not_ rely on the use ARM_ACK (positive acknowledgement) messages. However, some applications may benefit from some limited form of positive acknowledgement for certain functions. A simple, scalable positive acknowledgement scheme is defined which can be leveraged by protocol implementations when appropriate.

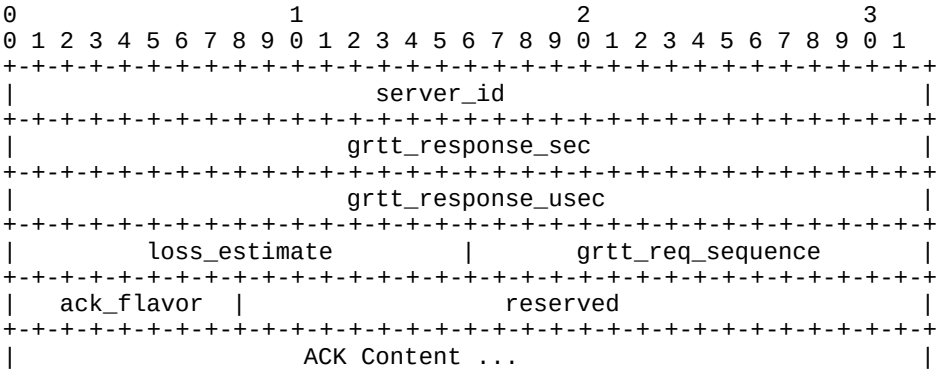


Figure 16. ARM_ACK Message

The "server_id", "grtt_response_sec", "grtt_response_usec", "loss_estimate", and "grtt_req_sequence" fields serve the same purpose as the corresponding fields in ARM_NACK messages.

The "ack_flavor" field indicates the nature of the following ACK content. This directly corresponds to the "ack_flavor" field of the ARM_CMD(ACK_REQ) message.

Currently these include: ARM_ACK(WATERMARK), ARM_ACK(SESSION) and ARM_ACK(RTT)

For example, if the sender has requested explicit positive acknowledgement of reception of a watermark "object_transport_id", the receiver will respond with "ack_flavor=ARM_ACK(WATERMARK) and appopriately valued "object_transport_id" field. If the ARM_ACK is simply a response to an explicit ARM_CMD(GRTT_REQ), the "ack_flavor" is set to ARM_ACK(RTT) and there is _no_ ACK content. (The length of the content may be inferred from the ARM_ACK packet payload size).

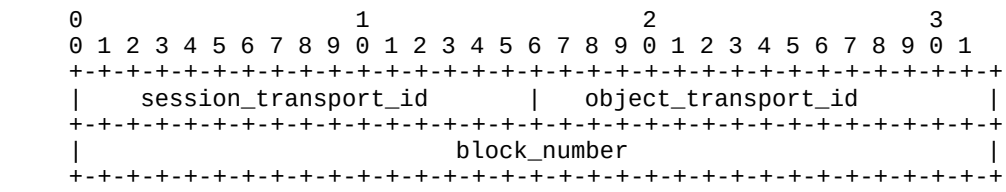


Figure 17. ARM_ACK(WATERMARK) and ARM_ACK(SESSION) Content

The "session_transport_id", "object_transport_id" and "block_number" are used by the receiver to acknowledge a ARM_CMD(ACK_REQ(WATERMARK)) transmitted by the sender identified by the "server_id" field.

The ARM_ACK(RTT) message has no attached content. Only the ARM_ACK header applies.

Since it may be useful for applications to leverage the ARM positive acknowledgement mechanism for other purposes, additional ARM_ACK "ack_flavors" may be used by the application for other purposes. The content of the ARM_ACK message (past the ARM_ACK header "reserved" field) for application-defined "ack_flavor" values, is specific to the application but is limited by ARM message size of the sender referenced by the "server_id".

Detailed Protocol Operation

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

- 1) The sender transmits an ordinal set of ArmObjects in the form of ARM_DATA (and optional ARM_INFO) messages labelled with ArmSessionTransportIds, ArmObjectTransportIds and block numbers.
- 2) The sender periodically transmits ARM_CMD(ACK_REQ(RTT)) messages and/or ARM_CMD(CC) (TBD) messages as needed to initialize and collect roundtrip timing and congestion control feedback from the receiver set. These messages are transmited without interuption of ARM_DATA, ARM_NACK and other messages.
- 3) As receivers detect missing content from the receiver, they initiate repair requests with ARM_NACK messages. Note the receivers track the sender's most recent "session_transport_id, object_transport_id and block_number" transmit position and NACK _only_ for content ordinally prior to that transmit position. The receivers use random backoff timeouts before generating ARM_NACK messages and wait an appropriate amount of time before repeating the ARM_NACK if their repair request is not satisified.
- 4) The sender aggregates repair requests from the receiver set and "rewinds" to send appropriate repair messages. The sender sends repairs for the earliest ordinal transmit position first and main- tains this ordinal repair transmission sequence.
- 5) The sender transmits ARM_CMD(FLUSH) messages when it reaches the end of newly available transmit content. Receivers respond to the ARM_CMD(FLUSH) messages with ARM_NACK transmissions (following the same suppression backoff timeout strategy as for data).
- 6) The sender transmits ARM_CMD(ACK_REQ(WATERMARK)) and and ARM_CMD(ACK_REQ(SESSION)) messages when it reaches the end of newly available transmit content or completes current session. Receivers first respond

with ARM_NACK requests and when all missing data is collected, then with ARM_ACK

- 7) The sender transmission rate is subject to rate control limits determined by congestion control. Each sender in a ArmSession maintains its own independent congestion control state.

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