DRAFT

NACK-Oriented Reliable Multicast (*norm*) User's Guide

Quick Links

Conventions

Through out this document, the following conventions are used:

- **NORM** (upper case bold) - refers to the NACK-Oriented Reliable Multicast specification developed under the Internet Engineering Task Force.

- *norm* (lower case italicized) - refers to a specific software implementation of NORM developed by the Naval Research Laboratory (NRL).

- Sender. "Sender" refers to the initiator of data in a *norm* multicast or unicast data flow. In a multicast group, a node can be a sender of flows of information, but also a receiver of flows of information. The *norm* commands

- Receiver. "Receiver" refers to the receiver of a specific *norm* multicast or unicast flow. Although a node may be a receiver, it will also transmit certain feedback data to the sender.

Introduction

NORM (NACK-Oriented Reliable Multicast) is an Internet protocol that is out growth of work in reliable multicast. It is one of three approaches endorsed by the IETF for addressing the problem of reliable multicast. The other two approaches are Asynchronous Layered Coding (ASL) and Tree-based Reliable Acknowledgement (TRACK).

Standard Internet transport layer mechanisms for ensuring reliability, i.e., TCP ACKs, were shown not to scale in multicast environment due to "ACK implosions" when too many receivers ACK-ed at once. Consequently, multicast applications used unreliable transport (i.e., UDP) in order to _____.

NORM addresses the reliability issue by incorporating forward error correction (FEC) and NACKs (negative acknowledgement). Rather than sending an ACK for data that <u>are</u> received, receivers send a NACK for data that <u>are not</u> received. The NACK induces the sender to provide FEC for erasure repair. "NACK implosion" is circumvented through the use of exponential backoff timers among the receivers to create a NACK suppression mechanism. This allows **NORM**-enabled applications to scale. Finally, a rate-based congestion control mechanism ensures fair sharing of the network with other flows, thus enabling **NORM** to be used in general Internet environments.

Detailed descriptions of NORM operating mechanisms can be found in the following IETF documents:

> NACK-Oriented Reliable Multicast (NORM) Protocol - Internet Draft - November, 2003
>
> NACK-Oriented Reliable Multicast (NORM) Building Blocks - Internet Draft - November, 2003

This guide pertains to a specific **NORM** implementation developed by the Naval Research Laboratory Networks and Communication Systems Branch (Code 5520).

NORM considerations

Use of any implementation of NORM will entail overhead. Specifically, NORM default messages contain 48-byte headers in addition to any TCP/IP headers. For single purpose applications, however, such as message streaming, the 16 bytes of this that comprise the

NORM performs round-trip timing probes that may be reduced for static, dedicated networks.

User-specified *norm* features

In addition to default features inherent to the **NORM** standard, *norm* has several user-specified features.

Reliable File Multicasting (or unicasting)

The sender can send lists of files or contents of directory trees. This also includes the ability to monitor the transmission list for changes (e.g. a directory may act as a "hot" outbox)

The receiver can store received files/directories to a specified location and optionally "post-process" received files with user-specified program or shell script.

Reliable Streaming (2 modes)

The user can specify two reliable streaming modes:

1. Classical TCP-like byte stream mode, which does not delineate boundaries of in the data stream.

2. Message streaming with automated message boundary recovery at receivers.

Fixed Rate or Congestion Controlled Operation

The user can specify a fixed transmission rate. Or, the user can specify a congestion control mode that creates a "TCP-friendly" flow that fair shares the bandwidth. The latter is required for Internet operations.

"Silent Receiver" option

– Individual receivers can configured for silent (EMCON) mode
– Group may have mix of silent and nacking receivers.

Installation and set up

The *norm* application can be easily built on most Unix-based platforms and WIN32 platforms. The *norm* source code is also available for checkout from a CVS repository at http://pf.itd.nrl.navy.mil/projects/norm. See the instructions there for further details. Note that the NRL Protolib source code distribution is required and is also available via CVS from the NRL ProteanForge site (http://pf.itd.nrl.navy.mil/projects/protolib/).

UNIX

The *norm* program can be built with *gcc* on most Unix platforms.

1) Download and unpack the *norm* source code distribution.
2) cd norm/unix directory
3) Select the appropriate Makefile.<os> for your operating system and type "make –f Makefile.<os> norm" to build.

WIN32

A Visual C++ workspace and project is provided to build the *norm* tool. Note that the Microsoft Platform SDK may be required to build *norm* with some of its more advance networking features such as IPv6 support and eventual RSVP support.

1) Download and unpack the *norm* source code distribution
2) cd norm/win32 directory
3) Open the "norm.dsw" Visual C++ workspace file
4) Type F7 to build the norm.exe program. (The executable will be found in the directory "norm/win32/Release" or "norm/win32/Debug" depending upon which configuration is selected).

norm usage

norm must currently be launched from a command line. To launch *norm*, use the following command line syntax:

General:	`address <address>/<port> ttl <value> rate <value (bits/sec)> backoff <value>`
Sender:	`cc {on|off} send <path> input {<device>|STDIN} minput {<device>|STDIN}  interval <value (seconds)> repeat <count> rinterval <value (seconds)> txbuffer <value (bytes)>  segment <value (bytes)> block <count> parity <count> auto <count> extra <count>`
Receiver:	`rxcachedir <path> output {<device> | STDOUT} moutput {device | STDOUT} processor <command> unicastNacks silent rxbuffer <value (bytes)>`
Debugging:	`debug <level (1-12)> trace log <path> txloss <percent> rxloss <percent>`

General:

All nodes, whether they are senders or receivers during a specific session, may be transmitting information. Senders will be transmitting session data, receivers may be transmitting feedback data (if the `silent` option has not been set). The following general commands pertain to both senders and receivers.

address <address>/<port>	Designates multicast session address and port number. In unicast mode, the sender must designate the specific receiver, and vice versa.	Required
ttl <value>	Designates session multicast time-to-live (hop count). The default value is ___.	Optional

rate <value (bits/sec)>	Designates peak transmit rate in bits per second.	Required
backoff <value>	Designates the NACK backoff factor. For general Internet operations, a value of '4' is recommended for multicast, and a value of '6' is recommended for unicast.	Optional

Sender:

"Senders" are nodes that initiate a specific *norm* session and transmit the *norm* data.

cc {on|off}	Turns congestion control on or off. When turned on, a rate-based congestion control scheme allows fair sharing of the network with other network flows. When turned off, *norm* will transmit at the peak transmit rate set by the rate command. NOTE: Congestion control <u>should be turned on</u> when using the Internet in accordance with IETF guidelines. Default = on.	Optional
send <path>	Specifies the path/name of the *file/directory* to be transmitted.	Required
input {<device>|STDIN}	Specifies the source of the *byte stream* to be transmitted. Appropriate when supporting applications that do not have native delineation of boundaries in the data stream, such as files.	Conditionally required; either `input` or `minput` must be specified
minput {<device>|STDIN}	Specifies the source of the *message stream* to be transmitted. Appropriate when supporting applications that have native delineation of boundaries in the data stream, such as video.	Conditionally required; either `input` or `minput` must be specified
interval <value (seconds)>	Specifies the object transmission interval. See "*norm* Intervals", below, for more discussion on this.	Optional

	Default =	
updatesOnly	Upon repeat transmission of the transmit file/directory list, NORM will only transmit files which have been added or updated since the previous transmission. This, along with the "repeat" and "rinterval" options can be used to create a sort of "hot outbox" capability.	Optional
repeat <count>	Transmit list repeat count. See "*norm* Intervals", below, for more discussion on this. Default =	Optional
rinterval <value (seconds)>	Transmit list repeat interval. See "*norm* Intervals", below, for more discussion on this. Default =	Optional
txbuffer <value (bytes)>	Sets the transmit buffer size. See "Transmit Buffer Size Considerations", below, for more discussion on this. Default =	Optional
segment <value (bytes)>	Sets *norm* segment size. See "FEC Considerations", below, for more discussion on this. Default =	Required
block <count>	Sets data segments per FEC block. See "FEC Considerations", below, for more discussion on this. Default =	Optional
parity <count>	Sets the parity segments calculated per block. This is an integer value. See "FEC Considerations", below for more discussion on this. Default = Range =	Optional
auto <count>	Sets the auto parity sent per block. This is an integer value. See "FEC Considerations", below for more	Optional

	discussion on this. Default = Range =	
extra <count>	Sets the extra parity sent per request. See "FEC Considerations", below for more discussion on this. Default =	Optional

Receiver:

"Receivers" are the recipients of *norm* data during specific *norm* sessions. (Note: a receiver during one *norm* session may be a sender during another *norm* session.)

rxcachedir <path>	Receive cache directory specifies directory path to temporary cache or permanently store received files/streams.	Required for receive nodes
output {<device> | STDOUT}	Received byte stream destination	Conditionally required; either `output` or `moutput` must be specified
moutput {device | STDOUT}	Received message stream destination	Conditionally required; either `output` or `moutput` must be specified
processor <command>	Specifies command (and command-line options) for post-processing of received files. Null post-processing can be specified with '`-X none`' and files will just be permanently saved to disk if the "`-a`" option (archive mode) is also set. (default = "xv -rmode 6 -root -quit" (for Unix), Win32 uses your default web browser by default)	Optional
unicastNacks	Enables unicast NACKing.	Optional
silent	Sets the silent receiver mode.	Optional

rxbuffer <value (bytes)>	Specifies the size of the receive buffer. See "Receive Buffer Size Considerations", below, for discussion on this. Default =	Optional

Debugging:

Debugging options may be applied to both senders and/or receivers.

debug <level (1-12)>	Debug message level. Sets the level of debugging detail displayed or logged. Current debug levels are as follows: • 0 : Misc errors only. • 2 : Major events (e.g. file tx/rx start/stop) plus client statistic reports. • 4 : Detailed file reception progress • 6 : General protocol operation. • 8 : More detailed protocol operation. • 10 : High level NACK construction/handling • 12 : Detailed NACK construction/handling (default = debug level 0)	Optional
trace	Message trace output. Enables time stamped debug message tracing.	Optional
log <path>	Debug log file (stderr default)	Conditional.

txloss <percent>	Sets simulated transmit packet loss at the sender. This will produce correlated losses across a multicast group.
	
	Default = 0%
	Range = 0 - 100%.
		Optional

rxloss <percent>	Sets simulated receive packet loss at the receiver. This will produce decorrelated losses at individual receivers.	Optional

	Default = 0%
	Ranges = 0 - 100%.	

norm Intervals Considerations

norm can send data objects (e.g., files) in several modes: singularly or as groups; only once or repeatedly. The following *norm* commands can control these modes:

`interval <value (seconds)>` controls the interval between transmission of data objects.

`repeat <count>` controls the number of times a data object (or group of objects) will be transmitted. A <count> value of –1 indicated infinite repeat transmissions of the transmit file/directory list items.

`rinterval <value (seconds)>` controls the interval between repeats.

`updatesOnly` : transmit added/updated files only upon repeat.

Transmit Buffer Size Considerations

The transmit buffer is the amount of memory allocated for buffering calculated FEC parity segments for possible retransmission when transmitting files/data objects. *norm* senders keep state on up to 256 objects in the current implementation, and the *norm* sender can repair file/data objects for which the FEC has been kept in this transmit cache.

The transmit buffer is also used hold parity segments for the *norm* sender's stream content. This can be set to limit the "age" of data the sender is "willing" to repair.

FEC Considerations

The choices of FEC parameters are closely tied to both the application and the network. In general, the tradeoff that needs to be considered is "latency vs. overhead".

Each *norm* data block has an FEC parity block associated with it. The size of this parity block is set with the `parity <count>` parameter. The larger the parity block, the more robust the erasure correction operation will be. However, the parity block imposes demands on the CPU, and the larger this block, the greater the demands placed on the CPU.

norm data blocks can be sent in several modes: without their associated parity blocks, with partial parity, or with full parity. This is determined by the `auto <count>` parameter.

Sending *norm* blocks without auto parity will produce a certain amount of latency associated with the time required for NACKing and transmitting parity data. However, this will result in minimum overhead because parity data will only be sent when requested by a receiver.

Sending *norm* blocks with full parity will minimize latency, however, there will be an associated overhead cost due to the routine transmission of the parity blocks.

norm blocks can also be sent with partial parity, which may achieve an optimum or near-optimim balance between latency and overhead costs. In this case, receivers will attempt to repair any erasures using the partial parity that was sent with the data block, and NACK for additional parity only when necessary. Assuming uniformly distributed losses, the bigger the block size, the more effectively "auto parity" can be applied to repair blocks without NACKing.

In some situations, a network may experience higher losses than originally anticipated. In this situation, the `extra <count>` parameter may be able to provide some level of mitigation. The `extra <count>` command sets the amount of additional parity sent when a receiver NACKs.

As stated, the choices of FEC parameters will be dependent on both the application and the network. The tradeoffs involved can be studied and appropriate solutions determined using a simulation environment, such as the ns-2 simulator. A discussion of *norm* and the ns-2 simulator is found at the end of this document.

Receive Buffer Size Considerations

– Amount of memory allocated for buffering partially received FEC coding blocks *per* sender.
– NORM receivers keep state on up to 256 objects for each sender.
– Silent receivers use the receive buffer size for receive streams instead of the default behavior of using the size "hint" provided by the sender.

Sample Scripts

The sample scripts listed below assume that NRL's MGEN packet generator is being used as a data source. For more information about MGEN, to include download and installation instructions, see http://pf.itd.nrl.navy.mil/projects/mgen/.

- Stream transmission (with MGEN sender):

```
mgen event "on 1 sink dst 0.0.0.0/1 periodic [200 1252]"
output /dev/null sink STDOUT | norm addr 224.1.1.1/5001 rate
3000000 segment 1252 block 40 parity 16 auto 6 backoff 0
minput STDIN
```

- Stream reception (with MGEN receiver):

```
norm addr 224.1.1.1/5001 backoff 0 moutput STDOUT | mgen
source STDIN output mgenLog.drc
```

- File transmission:

```
norm addr 224.1.1.2/5002 rate 5000000 send <fileName>
```

- File reception:

```
norm addr 224.1.1.2/5002 rxcachedir /tmp
```