norm User's Guide

NORM Version 1.4b4

1. Background

This document describes the usage of a demonstration application that uses the NACK-Oriented Reliable Multicast (NORM) transport protocol for reliable transmission of files, byte or message stream content. The name of the executable binary is "*norm*". It should be noted that this "demonstration application" applies a subset of the capability of the NORM protocol. Additionally, the current version of this application does not use the NORM Application Programming Interface (API) that is described in the "NORM Developer's Guide". The current *norm* demonstration application source code preceded the development of the NORM API. A future version of this demonstration application will be created that uses the NORM API and will also serve as a reference to NORM developers.

The *norm* application supports the following uses:

1. One time or repeated tranmission/reception of a set of files or directories

2. Transmission/reception of a byte stream piped into the STDIN of the sender application instance (Unix systems only)

3. Transmission/reception of a "message" stream piped into the STDIN of the sender application instance (Unix systems only)

The *norm* command-line (and run-time remote control interface) allow configuration of a large number of NORM protocol parameters. Again, note that while a considerable range of NORM protocol functionality is available in the *norm* application, it does not demonstrate the full set of NORM protocol capabilities. The distribution also includes the raft and npc (NORM Pre-coder) applications that can be used as "helpers" to the *norm* demonstration application for various purposes.

The NORM protocol is described in Internet Engineering Task Force (IETF) Request For Comments (RFC) RFC 3940 and RFC 3941. These are experimental RFC standards. These documents have been revised in recent Internet-

Drafts and it should be noted that the Naval Research Laboratory (NRL) implementation of NORM that is represented here has been updated to reflect the revised protocol.

In addition to this demonstration application, NRL provides a NORM protocol library with a well-defined API that it is suitable for application development. Additionally, the NRL source code distribution supports building the NORM protocol as a component into ns-2 and OPNET network simulation environments. Refer to the NRL NORM website <http://cs.itd.nrl.navy.mil/work/norm> for these other components as well as up-to-date versions of this demonstration application and documentation.

2. Building *norm*

The *norm* application can be built from the NRL NORM source code distribution. For several Unix-based operating systems, "Makefiles" are provided to build the NORM protocol library and example applications including this one. For Win32 and WinCE systems, workspace and project configuration files are provided for the Microsoft Visual C++ development environment.

2.1. Unix

To build the *norm* demonstration application:

1. Download and unpack the NORM source code tarball.
2. `cd norm/unix`
3. `make –f Makefile.<operating system> norm`

2.2. Win32

To build the *norm* demonstration application:

1. Download and unpack the NORM source code tarball
2. Make sure your VC++ environment has the Microsoft "Platform SDK" installed and is configured to use its header, library, and executable files.
3. Open the "`norm.sln`" (VC++ .Net), "`norm.dsw`" (VC++ 6.0), or "`norm.vcw`" (Embedded VC++) workspace file.
4. The "*norm*" project can be selected and built. The "`norm.exe`" file will be found in the "`norm/win32/norm/Release`" directory.

3. Concepts of Operation

The *norm* application supports several different uses. The most typical use is reliable multicast of files from a sender to a set of receivers. However, on Unix systems, the option is available to pipe (via STDIN) live byte or "message" streams into the *norm* sender application for transmission to the receiver(s).

The *norm* address command specifies the destination address and port to which NORM protocol messages are transmitted. For multicast operation, senders and receivers must use a common address and port number. Unicast operation is also supported, but some care must be taken with usage. Typically, for unicast operation, receivers should be configured with the unicastNacks option to ensure that feedback messages are properly directed back to the appropriate sender.

NORM messages are sent in User Datagram Protocol (UDP) packets. The user must make sure that any firewall configuration allows transmission and reception of UDP datagrams for *norm* to work properly.

Most of the NORM protocol parameters are set at the sender and the NORM protocol advertises parameters to the receiver(s) in the headers of NORM messages. This allows for somewhat loosely coordinated multicast operation.

Typically, it is expected that receivers will join the applicable multicast group and begin listening ahead of time. Then, the sender(s) will transmit content to the group for reliable transfer. For NORM stream operation, it is important to note that the *norm* demonstration application only supports a single sender per multicast group.

3.1. File Transmission

Receiver *norm* instances must use the rxcachedirectory command to specify a file system directory that is used to store received content. Note that the post processing (see processor command description) option of *norm* allows received content to be processed and/or removed from this cache directory to a permanent storage location if desired.

The sendFile command is used for the *norm* sender(s) to specify the file(s) and/or directories that should be transmitted. By default, the files are sent once. Directories are recursively scanned for files and those files are transmitted once. Note that zero-sized files are not transmitted. The repeatcount and rinterval (repeat interval) commands can be used to repeat transmission of the file/directory set on a scheduled interval after completion of prior transmission. Additionally, the sender updatesOnly option can be specified so that on repeated scan of the file/directory set, only files that have been changed or added are transmitted. This allows the option for a "hot outbox" to be set up that is monitored by the *norm* sender for transmission of files to the group. A simple multicast file sharing capability can be created in this way.

By default, files enqueued for transmission with the sendFile command are transmitted immediately, one after the other, but the interval command is available to pace the transmission of files. This can be used to allow time for post-processing at the receivers of subsequent files before new files are sent.

This is a synopsis of the most typically-used commands for file transmission. A number of other commands are available to customize the *norm* file transmission behavior. The reader is encouraged to read the descriptions of the available commands given later to understand the full range of options available.

3.2. Stream Transmission

Currently, this option is available only for Unix-based operating systems. Instead of transmitting files from the file system, the user may pipe (via STDIN) content directly to the *norm* sender application instance using the *norm* input or minput commands.. At receivers, the received content is directed to a descriptor set using the *norm* receiver output or moutput command. Two forms of stream transmission are available:

1. raw, unformatted "byte" streams, and

2. "message" streams.

The distinction between these two types is the presence of explicit message boundaries. The NORM protocol allows receivers to automatically recover message boundaries that have been marked by the NORM sender. This is useful when receivers may join the NORM session while it is already in progress or if there is intermittent network connectivity.

The input and output commands resepectively set sender and receiver "byte-stream" operation while the minput and moutput commands similarly set "message-stream" operation. It is expected that the "message-stream" operation offers the most utility for most purposes. "Byte-stream" operation may be used if the content is something like human-readable text, etc where distinct message boundaries may not be important. Again, note that *norm* receivers should begin listening before the sender begins transmitting for most effective uses of "byte-stream" operation.

For "message stream" operation, the *norm* application presumes that the first two bytes of messages are the message size (in bytes) in Big Endian (network) byte order. The NRL mgen (see <http://cs.itd.nrl.navy.mil/work/mgen>) and raft applications can be used to provide messages to *norm* in this format. The mgen application can be used to measure NORM message delivery performance for testing and experiment purposes, while raft provides the ability to capture UDP packet flows (e.g. Real-Time Protocol (RTP) video, etc) and reliably "tunnel" the UDP messages through NORM transport. At the receiver(s), raft can be correspondingly used to reconstruct UDP datagrams from the *norm* "message-stream" content. The usage of raft is described in Appendix A (TBD) of this document.

3.3. General Properties Overview

Most *norm* commands are for specifically sender or receiver operation. There are some commands that apply to both. These include the instance command that establishes a "remote control" inter-process communication facility that can be used to pass commands to instances of the *norm* program that are already running. Also, the debug, trace, and log commands are provided to display and/or store debugging output from the *norm* application and NORM protocol code. The txloss and rxloss commands are provided to invoke random dropping of sent or received NORM protocol messages for testing purposes. Other commands, like address, ttl, loopback, txport, and interface control the behavior of NORM UDP packet transmission and reception.

3.4. Sender Properties Overview

The *norm* sender configuration controls most aspects of NORM protocol operation. This includes transmission rate, packet size, Forward Error Correction (FEC) configuration, etc. The rate command determines the *norm* sender transmission rate in units of bits/second. The segment command sets the maximum size of NORM message payloads. The block, parity, and auto commands respectively set the number of user data segment per FEC block, number of calculated parity segments per FEC block, and number of proactively (automatically) transmitted parity segments per FEC block. The backoff command sets the maximuim number of round-trip time intervals over which timer-based feedback suppression is scaled and the grtt command sets the sender's initial estimate of round-trip time for the group. A detailed understanding of these various NORM protocol parameters can be attained by reviewing the NORM protocol specification documents and the "NORM Developer's Guide".

Another significant *norm* sender command is the txbuffer command. This sets the size of the NORM sender cache for calculated parity segments and FEC block repair state. For *norm* stream operation, this command also determines the size of the stream buffer. The stream buffer size limits the "repair window" when *norm* stream operation is used. A relatively large stream buffer size may be needed for high (bandwidth*delay, packet loss) conditions. Some other significant commands applicable to *norm* stream operation include the push and flush commands.

Although NORM is a NACK-based protocol, it does support optional collection of positive acknowledgement (ACK) from a subset of the receiver group. The *norm* ackingNodes and related ackshot commands can be used to exercise this optional protocol behavior.

Finally, in addition to the fixed transmission rate operation set with the rate command, *norm* also supports enabling automated congestion control with the cc command. The bounds of congestion control rate adjustment can be optionally set with the limit command.

3.5. Receiver Properties Overview

As mentioned, most of the NORM protocol behavior is controlled by the sender, but there are some options that the receiver can exercise. The most significant of these is the ability to put the receiver in an emission-controlled (EMCON), or "silent receiver" mode where no NACK or other feedback messages are generated. The silentClient command is available for this purpose. This is useful when using NORM for reliable transport over unidirectional network connectivity (In this case, it is expected the *norm* sender has been configured with some amount of auto (proactive) FEC parity in its transmission to overcome nominal packet loss). For optional use with the silentClient command, the lowDelay command is available to expedite delivery of received content (even if for a partially-received FEC block) to the application when subsequent FEC coding blocks are received. The default behavior would be for the *norm* receiver to buffer partially-received content as long as possible for possible repair in response to some other NACKing (non-silent) receiver. The lowDelay command overrides this default behavior.

The processor command is available to the *norm* receiver to specify a application, command, or script that is invoked upon successful completion of reception of files. The specified command is invoked with the received file name as the last argument. Users may employ this command to move received content to a permanent storage location, display received content, or other purposes (One could even cleverly control *norm* receiver or other system operation in this way if desired). A related command is the saveAborts command that causes even incomplete files (aborted) to be passed to the receiver post processor. An example use of this option would be if the files transmitted were pre-encoded with the npc (Norm Pre-Coder) utility such that original file content can be recovered from a partial npc-encoded file (See the npc User Guide for details).

4. *norm* usage

The *norm* application is launched from a command-line interface (e.g. Unix or DOS shell). Many of the *norm* parameters have default values, but typically the user will wish to set at least some of these differently than their defaults.

A minimal example *norm* sender command-line syntax is:

```
norm addr <addr/port> sendFile <fileName>
```

The corresponding minimal *norm* receiver command-line syntax would be:

```
norm addr <addr/port> rxcachedir /tmp
```

The sender would begin sending the specified `<filename>` at a transmit rate of 64 kbps. The receiver would receive the file and store it in the "`/tmp`" directory.

Typically, it is expected that the user would wish to set the *norm* transmit rate or enable congestion control operation. The *norm* application was designed principally for long-term participation in an IP multicast group with the receiver application running all of the time, post-processing received content as it arrived, and sender(s) transmitting content to the group (e.g., using the "hot outbox" approach mentioned) as it was available.

The *norm* receiver command-line syntax to support this operation would be:

```
norm addr <addr/port> rxcachedir /tmp processor <postprocessor>
```

The norm sender command-line syntax would be:

```
norm addr <addr/port> rate <bps> sendFile <outboxDirectory> repeat -1 updatesOnly
```

The "`repeat -1`" would cause norm to scan the `<outboxDirectory>` indefinitely for new file content and transmit those files to the specified group address and port. Note the transmit rate is specified in units of bits/second.

4.1. Unicast Operation

For unicast operation, the following usage is recommended:

Receiver: `norm addr 127.0.0.1/<port> unicastNacks rxcache <cacheDirectory> …`

Sender: `norm addr <rcvrAddr/port> sendFile <filename> …`

The *norm* receiver address is really a "don't care" value since feedback is transmitted to the sender's unicast source address and port detected during packet reception.

5. Command Reference

The following tables list the available *norm* command-line options. Note that if the instance command is used, many of these commands may be issued to instances of the *norm* application that are already running. The tables are grouped by the categories of "General", "Sender", and "Receiver" commands.

Note that *norm* commands may be abbreviated on the command-line if desired.

5.1. General Commands

`address <addr>/<port>`	Designates session address and port number. For multicast operation, sender(s) and receiver(s) should use common address and port parameters. For unicast operation, the sender must designate the intended receiver address and port and the receiver must specify the same session

	port number. The receiver unicastNacks command should be used for unicast operation. IPv4 and IPv6 addresses are supported.
`txport <port>`	Specify a specific source port number for NORM transmission. This can also be set to equal the session port (ie. rxport). Default is system-assigned transmit port number.
`ttl <value>`	Designates session multicast time-to-live (hop count). The default value is 255.
`interface <ifaceName>`	Sets the network interface for multicast packet transmission/reception. <ifaceName> is name or IP address of a valid network interface. Default is system default multicast interface.
`loopback {on | off}`	Optionally enables reception of norm's own messages. Useful for loopback testing of sender/receiver configuration. Default = "off".
`txrobustfactor <value>`	Set the “robust factor” that determines the number of times NORM protocol NORM_CMD(FLUSH) messages are sent at end-of-transmission, how robustly positive acknowledgement collection is conducted, and how other “robustly-transmitted” (repeat-transmitted) sender control messages are managed. Higher “robust factor” values makes increases the assurance of protocol success in the face of significant packet loss. Lower values can be used to make *norm* less “chatty” but at the cost of reduced certainty that protocol operation will succeed under all circumstances. Note that this parameter needs to be consistently set at both the *norm* senders and receivers. Unlike many of the other parameters, this value is not advertised by the sender to the receivers in the NORM protocol message headers. The default value is 20. This provides about 95% likelihood of protocol success even with 50% packet loss. This is based on the probability that the receiver gets the sender NORM_CMD(FLUSH) messages and the sender gets NACKs from the receivers needed to complete reliable transfer. The special value of -1 will make *norm* indefinitely perform the related protocol actions (sender flush transmission, positive acknowledgement collection until success, etc). This is typically not recommended.Default = 20.
`instance <instanceName>`	Specifies "name" of the first running instance. If a *norm* instance is already running with the specified <instanceName> the commands given will be issued to that already-running instance of norm.
`debug <debugLevel>`	Sets verbosity of debug output. Higher values are more verbose. The range is from 0-12. The default debug level is zero.
`trace {on | off}`	Enable/disable NORM protocol message trace in debug output. Message trace is timestamped logging of information for every packet sent or received. Default = "off".
`log <filename>`	Directs debug output to specified file.
`txloss <percent>`	Sets percentage of messages to be transmitted that are randomly dropped (for testing purposes). Default = 0.0 percent.

`rxloss <percent>`	Sets percentage of received messages that are randomly dropped (for testing purposes). Default = 0.0 percent.
`help`	Displays command set with short descriptions.

5.2. Sender Commands

`rate <rate in bits/sec>`	Sets the sender maximum transmission rate. All sender transmissions (user data, repair, protocol messages) are subject to this rate limit. Default = 64 kbps.
`cc {on|off}`	Enables/disables NORM TCP-friendly congestion control operation. 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 transmit rate set by the rate command. Default = "off".
`limit <rateMin:rateMax>`	Sets lower/upper bounds on transmit rate adjustment when congestion control operation is enabled. A value of -1.0 indicates no limit. Default rateMin/rateMax = -1.
`segment <bytes>`	Sets *norm* message payload size (segment size) in bytes. Default = 1024 bytes.
`block <segments>`	Sets number of user (source) data segments per FEC encoding block. Default = 64.
`parity <count>`	Sets number of parity segments calculated per FEC encoding block. Default = 32.
`auto <count>`	Sets the number of parity segments that are proactively (automatically) transmitted with each block of user (source) data segments. A non-zero count can provide for robustness/reliability with no NACKing from receivers required. This value must be less than or equal to the value set with the parity command.Default = 0.
`extra <count>`	Instructs the sender to respond to repair requests (NACKs) by sending <count> extra repair segments beyond what the receiver(s) requested. For experimental purposes. Default = 0.
`silentClient`	This informs the *norm* sender that silent receivers will be used and it should redundantly transmit NORM_INFO content at the end of each FEC coding block.
`grtt <seconds>`	Sets the *norm* sender's initial estimate of group round-trip timing. This value affects the latency of the NORM repair process (and thus impacts buffer size requirements). Default = 0.5 seconds.
`backoff <factor>`	Set the factor used to scale feedback suppression backoff timeouts. Small groups not concerned about feedback implosion may use small or zero values to minimize delay of NORM repair process. Default = 4.
`txbuffer <bytes>`	Sets the *norm* sender transmit buffer and stream buffer size, if applicable. The transmit buffer is used to cache calculated FEC parity segments and FEC code block repair state. The stream buffer size limits the "repair window" for stream transmission (and hence maximum possible latency).Default = 1Mbyte.

`txcachebounds <countMin:countMax:sizeMax>`	This sets the “transmit cache bounds” that are used to determine how many prior transmit objects for which the *norm* sender maintains state. This essentially limits the “repair window size” that the NORM sender observes has for responding to repair requests (NACKs) from receivers. The “transmit cache bounds” also SHOULD be set to be compatible with any use of the requeue option described below (i.e. the safest thing to do is set <countMin> here to a value greater or equal to the number of files in the transmit file/directory list, including the count of files in any directories). The <countMin> value sets a minimum number of transmit objects (files) for which the NORM sender will keep repair state, regardless of the file sizes while the <countMax> value sets a maximum object count that the NORM sender will keep in its “repair window”. The <sizeMax> value in units of bytes) limits the repair window according to sum total of the file sizes in the cache, providing that state is kept for at least <countMin> objects. The <countMax> limit is most useful when the file sizes are somewhat small (i.e. <sizeMax> is not reached) and the user wishes to limit repairs of “older” files sent. Note that <sizeMax> does not directly relate to memory allocation since NORM recovers file data directly from the file storage system as needed. An IMPORTANT caveat here is that the current NRL NORM implementation has a hard-coded limit that NORM receiver will keep state for a maximum of 256 objects per sender. Thus, the value of setting the txcachebounds count values greater than 256 is limited. This limitation will be fixed in an updated to the NORM code and will be reflected here. Default: countMin = 8, countMax = 256, sizeMax = 20 Mbyte
`gsize <count>`	Sets the estimate of receiver group size used by NORM for scaling time-based feedback suppression.
`sendFile <path>`	Adds a file or directory to the *norm* transmit file/directory list. Directories are recursively scanned for files. Zero-sized files are not transmitted.
`repeatcount <count>`	Repeat scan or transmission of transmit file/directory list set with sendFile command. A <count> of -1 means infinite repeats. With each “repeat” pass through the transmit file/directory, the files are sent with new NormTransportId values and considered separate transmit objects by the NORM protocol. This is different than the requeue option which causes the file(s) to be repeat transmitted with the same NormTransportId. Note these two different options can be used together and the result is a “multiplicative” effect with regard to the amount transmission that occurs.Default = 0.
`rinterval <seconds>`	Specifies a time delay between repeated scan or transmission of transmit file/directory list. Default = 2 seconds.

`requeue <count>`	Specifies the number of additional repeat transmissions of the each file using the same NormTransportId such that the multiple transmissions can be “stitched” together by the receiver into a successful reception even if a single transmission is unsuccessful (useful for “silent receiver” mode along with the “auto” parity option). This is distinct from the repeatcount option in that the repeatcount option specifies how many repeated passes through the transmit file/directory list with files getting new NormTransportIds and thus considered separate NORM transmit objects. Note these two different options can be used together and the result is a “multiplicative” effect with regard to the amount transmission that occurs. A requeue <count> value of 0 means that each file in the transmit file/directory list is sent once as a distinct NORM transport object (i.e. no requeue occurs). A requeue <count> value of -1 indicates the files are requeued indefinitely (and thus any configured “repeatcount” or “updatesOnly” options become irrelevant). Note that if the number of files in the transmit file/directory list exceeds the txcachebounds limits, then the “requeue” option will not work. Thus, it is important to set the txcachebounds accordingly to use the requeue option. Default = 0 (disabled)
`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.
`oneshot`	Causes the *norm* sender application to exit after the NORM TX_FLUSH_COMPLETED event at the end of file list transmission. By default, the *norm* sender application will run indefinitely.
`ackingNodes <node1,node2,…>`	The comma-delimited list of NORM node identifiers is used with NORM positive acknowledgement operation. Acknowledgment from the specified list of nodes is collected for each transmitted file before sending subsequent files. The *norm* application uses its host's default IP address for a "node id". Default is no acking nodes.
`ackshot`	The sender application exits after completing positive acknowledgement collection.
`input {<device> | STDIN}`	Sets *norm* sender "byte-stream" operating mode using input from specified <device> path or STDIN. With STDIN, the STDOUT of another process may be piped into the *norm* sender. (Unix-only).
`minput {<device> | STDIN}`	Sets *norm* sender "message-stream" operating mode using input from specified <device> path or STDIN. With STDIN, the STDOUT of another process may be piped into the *norm* sender. "Messages" are expected to have a 2-byte, Big Endian message size prefix. (Unix-only).
`flush <flushMode>`	Sets *norm* sender flush behavior for "message-stream" operation. Valid options include "none", "passive", and "active". With "none", no flushing is invoked; stream transmission simply pauses when no input data is available and *norm* always sends full NORM_DATA mes-

	sages according to the set <segmentSize>. With "passive" or "active" flushing enabled, the NORM stream is flushed with each completed message and variable-sized NORM_DATA messages may result. With "active" flushing, the NORM_CMD(FLUSH) message is actively transmitted when there is no data available to transmit. This makes NORM more "chatty" but provides more robust, lower-latency reliability for stream transmission.Default <flushMode> = "none".
`push`	When set, new input data is always written to the NORM stream regardless of pending repair transmissions. This favors new application data transmission over repair of older stream data. Suitable for applications that can tolerate "quasi-reliability" and desire low latency.

5.3. Receiver Commands

To invoke *norm* receiver operation, one of the `rxcachedir`, `output`, or `moutput` commands MUST be given on the command-line.

`rxcachedir <path>`	Sets the directory where received file content is stored by the *norm* receiver. This is a required command for *norm* file reception.
`processor <command>`	Specifies command (and any options for that command) for post-processing of received files. The received <filename> is appended as the last argument to the specified command when invoked for each received file.
`saveAborts`	Causes incomplete (aborted) files that are partially received to be saved and/or passed to the post processor. The default behavior is that incomplete, partially received files are deleted.
`output {<device> | STDOUT}`	Received "byte-stream" output is directed to the specified <device> path or STDOUT.
`moutput {<device> | STDOUT}`	Received "message-stream" output is directed to the specified <device> path or STDOUT. Output messages will have a 2-byte, Big-Endian prefix indicating the message size.
`unicastNacks`	Feedback messages are unicast back to detected sender source address(es) instead of being sent to the *norm* session address. Default behavior is feedback is sent to the session (usually multicast) address. This receiver option is RECOMMENDED for unicast operation.
`silentClient`	In this mode, the receiver sends no feedback messages and relies solely upon sender proactive (auto parity) FEC content for reliable reception.
`lowDelay`	For use with silentClient operation. Source data for partially-received (incomplete) FEC coding blocks is provided to the application immediately when subsequent FEC blocks are received. This minimizes delay of delivery of user data to the application. The default behavior is to buffer partially-received FEC blocks for as long as possible in case repair transmissions (due to other non-silent receivers) are provided.

`rxrobustfactor <value>`	This sets a “robust factor” value at receivers that determines how persistently the receiver keeps state for remote senders that are not currently, actively transmitting data. This also corresponds to the maximum number of times the *norm* receiver will “self-initiate” NACKing to such an inactive sender before giving up. Unless rxpersist is specified (see below), the receiver will also free memory resources allocated for an inactive sender at this time. A <value> of -1 causes the receiver to be “infinitely” persistent. The default value is 20.
`rxpersist`	If this option is given, the receiver keeps full state on remote senders indefinitely, even when they go “inactive” (see above). The default behavior when this is not specified is for *norm* to free buffer memory resources allocated for senders after a timeout based on the txrobustfactor, rxrobustfactor, and measured GRTT.
`rxbuffer <bytes>`	Specifies the size of the *norm* receiver buffer that is allocated on a per-sender basis. This buffer is used to cache partially-received FEC coding blocks and associated object repair state. An operating mode or network connectivity with significant (bandwidth*delay, packet loss) may necessitate larger rxbuffer settings to preserve protocol efficiency.Default = 1 Mbyte.
`rxsockbuffer <bytes>`	Sets the size of the UDP receive socket buffer used for *norm* sockets. An extremely high transmission rate may require socket buffer settings above normal system defaults.

6. Parameter Considerations

(TBD) Discuss the considerations and trade-offs of NORM parameter selection. (e.g. FEC parameters, buffer sizes, etc). Note some of these issues are described in the NORM Developer's Guide and/or NORM Protocol specifications.

7. Example Usage

The example command-line usages 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://cs.itd.nrl.navy.mil/work/mgen>.

1. Message stream" transmission example(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
   ```

2. "Message stream" reception example (with MGEN receiver):

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

3. File transmission:

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

4. File reception:

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

A. "raft" Usage

The NRL NORM source code distribution supports building the "Reliable Application For Tunneling" (*raft*) utility that can be used as a helper application with *norm* to tunnel a UDP datagram flow over a reliable NORM "message stream" tunnel. The *raft* application can be configured to listen to a UDP port, optionally joining an IP Multicast group, and output received UDP payloads as "messages" to its STDOUT. This, in turn, can be piped into the STDIN of the *norm* application for message-stream tranmission. Additionally, on the *norm* receiver side, *raft* can be configured to accept messages from STDIN and re-encapsulate these as UDP datagrams transmitted to a specifed destination address and port.

(TBD) Finish description of raft usage and provide example of use with norm