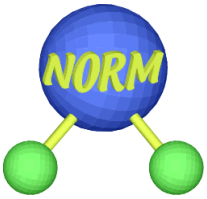

npc User Guide

(NORM Precoder User Guide)

Table of Contents

1. Background

The NACK-Oriented Reliable Multicast (NORM) protocol is capable of supporting robust transmission of content to "silent" receivers that are required or only capable of operating in an emission-controlled (EMCON) manner. This capability is enabled when the NORM sender is configured to proactively transmit Forward Error Correction (FEC) erasure coding content as part of its original data transmission. For NACK-based operation, the FEC repair packets are usually sent only reactively, in response to repair requests (NACKs) from the receiver group. However, hybrid operation with a combination of proactive FEC content and additional reactive FEC repairs as needed is also supported. Similarly, a mix of nacking and silent receivers may be supported with silent receivers capitalizing on the FEC repair information sent proactively and/or reactively. The purpose of the NORM Pre-Coder (**npc**) software utility described here is to support additional robustness for purely-proactive sessions, where the receivers are unable to request repair or retransmission of content.

The Naval Research Laboratory (NRL) reference implementation of the NORM protocol includes support for 8-bit (and very soon 16-bit) Reed-Solomon FEC encoding with additional support for other coding algorithms (e.g., Low-Density Parity Check (LDPC)) planned for the future. The NORM specification allows for different FEC algorithms to be applied within the protocol. The current Reed-Solomon NORM FEC algorithms in the NRL implementation are limited to modest code block sizes (With 16-bit Reed-Solomon coding, larger block sizes will be allowed but very high data rates may not be possible). For channels with random errors, the current NORM FEC codecs are often adequate as there is flexibility in how the encoded data can be partitioned into FEC blocks (a block consists of some number of data segments (packets)) and the number of FEC parity packets that can computed and possibly transmitted per source data block. For channels with large bursts of packet loss (with respect to the configured NORM FEC block size), it is quite possible that the number of lost packets (erasures) that occur within a NORM FEC block may exceed the configured erasure-filling capability. The **npc** utility was created to "pre-encode" (and "post-decode") files for NORM transmission to silent (non- NACKing) receivers by adding additional FEC encoding, and importantly, interleaving of the FEC segments (packets) to re-distribute bursts of packet losses as random losses over the entire file. It is thus most applicable to very large files (with respect to FEC block sizes).

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.

The **npc** tool is designed to use in conjunction with the NORM protocol and accompanying NORM file transfer examples that are part of the NORM source code distribution. However, the encoded file format that **npc** creates can be use with other transports as well. The key is to align the segmentation parameters of the **npc** configuration with that of the intended transport mechanism.

2. Overview

The **npc** utility takes, as input", a file and logically divides it into segments, adding cyclic-redundancy checksum (CRC) to the segments, encoding the source segments with Reeed-Solomon encoding (adding a configurable number of parity segments per FEC source block), and interleaves the source and encoding segments to an output file. The use of the CRC allows erasure to be detected and also provides additional assurance of correct content delivery by possibly detecting bit errors that may have been undetected during transport (i.e., link-layer framing, Internet Protocol (IP), and/or User Datagram Protocol (UDP) checksums). The interleaving by default is a block interleaver using the entire file as a logical block, but a limit on the interleaving size can be set to help increase the speed of the **npc** encoding/decoding process. This may be useful for extremely large file sizes.

3. Usage

The following is a synopsis of **npc** usage:

```
npc {encode|decode} input <inFile> [output <outFile>][segment <segmentSize>]
    {[[block <numData>][parity <numParity>]] | [auto <parityPercentage>][bmax <maxBlockSize>]}
    [imax <widthLimit>][ibuffer <bytes>][background][help][debug <debugLevel>]
```

The **npc** utility may be instructed to either "encode" a file (add FEC content and interleaving to the given <inFile>) or "decode" a file that was previously encoded with **npc**. The ".npc" file extension is suggested to delineate files that are of the **npc** encoded format. Note the "output" filename is optional. By default, **npc** will use the filename of the <inFile> as the output filename, replacing the '.' extension delimiter with a '_' (underscore) and adding the ".npc" extension suffix. The **npc** format includes some minimal "meta-data" in the first encoded <segmentSize> to convey the file size and name of the original file. On decoding, if the "output" file option is omitted, this "meta-data" is used to name the decoded output file.

The optional FEC parameters, <*segmentSize*>, <*numData*>, and <*numParity*> control the logical segmentation, blocking, and amount of FEC parity content added to the file. For use with NORM, it is recommended that the <*segmentSize*> value correspond to the same segmentation size used for NORM transmission. The <*numData*> (source segments per FEC encoding block) and <*numParity*> parameters should be selected to provide erasure filling coverage for the expected transmission packet loss characteristics. Note that when used with proactive NORM FEC transmission, the **npc** encoding provides an "inner" FEC code and interleaving and the NORM protocol provides an "outer" FEC encoding. The "outer" NORM code might be configured to deal with typical random packet loss due to channel BER, etc and the "inner" **npc** interleaving and coding could be correspondingly configured to handle expected burst losses (e.g. outages) that might occur.

The <*auto*> option provides an alternative means for setting the FEC encoding protection level instead of using the <*block*> and <*parity*> options. First the <*auto*> option causes **npc** to select a block size corresponding to the entire input file size (plus the segment of meta data information that npc adds). Then, the <*auto*> option <*parityPercentage*> value is used to set the number of parity packets per encoding block to the given percentage of the automatically determined block size. For exanple, the command "`auto 100`" causes **npc** to set an encoding rate of 100%. I.e., the parity segments sent will equal the number of segments in a bock. Note that the <*parityPercentage*> can even exceed 100% if desired for high levels of loss protection. Also note that the percentage here is _not_ a loss protection percentage with a 100% <*parityPercentage*> value being able to correct up to 50% errored or lost packets within a coding block. With 50% uniform random packet loss, this would result in successful file transfer about 50% of the time as, per Gaussian distribution, burst error probabilities would result in half of blocks arriving with greater than 50% packet loss and half with less than 50% packet loss.

When the **npc** encoder uses the "`auto`" command the **npc** decoder MUST also use the "`auto`" command and be configured with the same <*segmentSize*> and <*parityPercentage*> values. Similarly when the "`block`" and "`parity`" commands are used to explicitly set the <*numdData*> and <*numParity*> at the encoder, the decoder MUST be configured with the same corresponding options and values, again including <*segmentSize*>. And for use with

NORM protocol transport, the <*segmentSize*> parameter SHOULD be matched for best coding gain performance. The NORM block size (numData) and parity (numParity) parameters may be set indepedently. Basically, the NORM protocol proactive erasure coding can be configured to deal with expected short term random packet loss while the **npc** parameters can be configured to counter large burst (or outage) losses. The inner/outer encoding approach that the combination of **npc** and NORM provides, can allow for a sort of multiplicative coding gain to deal well with both random packet loss and bursts or outages with lower FEC overhead. However, when the **npc** coding is configured (e.g., via the "auto" option) to encapsulate an entire file into a single logical coding block, the desired level of loss protection can be simply "dialed into" the **npc** <*parityPercentage*> option. The tradeoff is that the larger FEC block size increases the computational requirement for file encoding and decoding. Future versions of **npc** may provide additional FEC code types

As basic example usage, to encode a file name "originalFile.txt" with the default **npc** naming convention, FEC, and interleaving parameters, use the following syntax:

```
npc encode input originalFile.txt
```

The default npc configuration is XXX.

This will produce and output file named "`originalFile_txt.npc`" in the current working directory, The original file can be recovered (decoded) using the syntax:

```
npc decode input originalFile_txt.npc
```

This will decode the ".npc" file, and in this case produce a file named "`originalFile.txt`" in the current working directory. (The file name information was stored in first "meta data" segment of the ".npc" file). This default naming convention can be overridden by using the npc "output" option. For example, the syntax:

```
npc decode input originalFile_txt.npc output file.txt
```

will produce a file named "`file.txt`" that is identical in content to "`originalFile.txt`".

3.1. Notes

The FEC and interleaving parameters that are used for **npc** encoding MUST be exactly matched to successfully decode the encoded file. I.e., if the defaults are used for encoding, the defaults must be used for decoding. The parameters that must match include <*segmentSize*>, <*parityPercentage*> and <*maxBlockSize*> (or <*numData*> and <*numParity*>), and <*widthMax*>.

It is possible that in some cases it may be beneficial to apply more proactive FEC content with the **npc** program instead of with the NORM transport. The trade-offs are scenario-specific.

The NRL "**norm**" demonstration application has commands included to support transport of **npc** encoded files. The distinction here is that a file that _fails_ NORM transport might still be successfully decoded with **npc**. There are two receiver-side norm demo application options that apply here:

1. The "`saveAborts`" command causes *norm* to not delete (and attempt to postprocess) "aborted" files (files that failed reliable NORM transport).

2. The **norm** "`lowDelay`" command should be applied for silent-receivers to more immediately deliver "failed" files to the application for post-processing (i.e., attempted **npc** decoding)

4. npc Command Reference

The following table describes each of the **npc** commands available in the command-line syntax.

`encode` | `decode`	Determine whether **npc** is to encode or decode the given <*inFile*>. This option is required and only one should be given.
`input` <*inFile*>	Specifies the file to be processed. Required command.

`output <outFile>`	Specifies the name of the output file to be produced. Overrides the default **npc** output file naming convention. Optional.
`segment <segmentSize>`	Sets the segmentation size (e.g., packet payload size) in bytes. Note four bytes of the `<segmentSize>` are used for a 32-bit CRC that **npc** applies to each segment. (Default `<segmentSize>` is 1024 bytes)
`block <numData>`	Specify the number of source data segments (packets) per npc FEC coding block. (Default block sizing is auto)
`parity <numParity>`	Specify the number of FEC parity segments (packets) added per npc FEC coding block. (Default is 2 segments).
`auto <parityPercentage>`	Specifies automatic FEC block sizing with `<parityPercentage>` indicating the percentage of FEC parity segments to include per block. The "auto" block sizing sets the block size as large as possible to treat the entire files as one logical FEC block to maximize FEC performance. The maximum possible block size currently supported by npc are blocks where (numData + numParity) is less than or equal to 65536. The maximum buffer size can be limited by using the `bmax` command.
`bmax <maxBlockSize>`	Limits the maximum block size when the `auto` command is used for automatic block sizing (Default is 65536)
`imax <widthMax>`	Limits interleaving of encoded file to a maximum interleaver width of `<widthMax>` segments. A value of ZERO (or less) defaults to **npc** calculating a block interleaver that encompasses the entire encoded file size. For extremely large files, this option may be beneficial to limit file seeking operations required to interleave the file. If the encoded file size is less than `<widthMax>`*`<widthMax>` segments, **npc** will again calculate its own maximum block size. (Default is 1000 segments interleaver depth (i.e., about 1 Gbyte interleaver size with the default 1024 byte `<segmentSize>` value))
`ibuffer <bufferSize>`	This sets the maximum memory (in bytes) that **npc** allocates for encoding. A larger value allows **npc** to perform file input/output with less seeking and improved encoding/decoding times can be achieved. (Default is 1.5 GByte)
`debug <debugLevel>`	Specifies debug output verbosity. Higher number is more verbose debugging information. (Default is ZERO).
`background`	Sets percentage of received messages that are randomly dropped (for testing purposes). Default = 0.0 percent.
`help`	Displays **npc** usage statement.