norp User's Guide

Abstract

The NRL Nack-Oriented Proxy (*norp*) project includes software for an RFC 1928 SOCKS5-compatible proxy server daemon that is able to use the RFC 5740 Nack-Oriented Reliable Multicast (NORM) transport protocol for efficient and robust data transfer between *norp* proxy instances. The *norp* proxy automatically supports conventional SOCKS TCP proxy operation when a remote *norp* peer is unavailable. This software was developed by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Networking Research Group. The NRL reference implementation of NORM used here is available from http://www.nrl.navy.mil/itd/ncs/products/norm.

Table of Contents

1. Overview

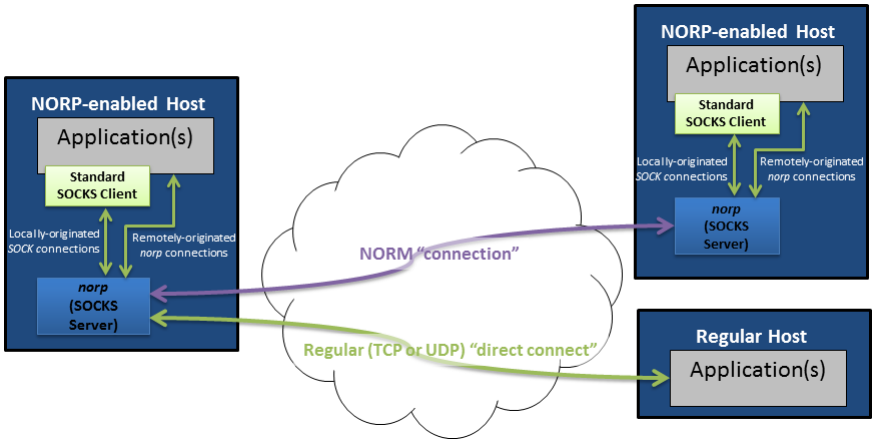
The *norp* application ....

2. Theory of Operation

The *norp* program acts as a SOCKS proxy server. It supports the SOCKS5 "CONNECT", "BIND" and "UDP-ASSOCIATE" proxy methods for conventional SOCKS proxy operation. The current *norp* implementation does not require (or support) any client authentication. Future versions may provide authentication or other access control mechanisms. The current *norp* implementation only provides NORM transport for the SOCKS TCP "CONNECT" requests. SOCKS "UDP-ASSOCIATE" over NORM will be supported in a future version.

Unlike a conventional SOCKS server, it is expected that the *norp* daemon can be installed and run as a local "loopback" server that is co-resident on the host running applications that wish to take advantage of NORM transport benefits. The *norp* daemon implements its own signaling protocol that will automatically determine, upon TCP (or UDP) connection establishment, if a remote destination is also similarly "*norp*-enabled" and establish a NORM transport connection as the proxy connection. Otherwise a "business-as-usual" TCP (or UDP) connection is established on the application's behalf and thus compatibility with "non-*norp*" hosts is also supported. Figure 1, “NORP Concept of Operation” illustrates this high level concept of operation.

Figure 1. NORP Concept of Operation



TBD - provide some more details on norp signaling for peer detection and NORM session establishment

Note that as an alternative to making proxied connections directly to connection destination addresses as illustrated above, a remote *norp* peer "correspondent" can be specified as part of the `forward` command or, for SOCKS connections, with the `correspondent` command (see command descriptions below). Future versions of *norp* will include more sophisticated "routing" options for different destinations and traffic types.

2.1. SOCKS Loopback and Intermediate System

As noted above the principal use case for *norp* is to act as a local, "loopback" SOCKS server that can be used in conjunction with a properly configured SOCKS client. In this way, all of the configuration parameters are localized and implicit and no precoordinated configuration with *norp* peers (or non-*norp* hosts) is required other than using a common UDP port number for NORP signaling.

However, there may be use cases where it may be desirable to deploy *norp* on intermediate systems at the connection originating site (or domain) and/or the destination site(s) (or domain(s)). This is easily supported by the *norp* design and future norp versions will provide configuration options for this type of deployment.

2.2. NORM Protocol Usage

TBD - describe how the NORM streaming capability is used in a flow-controlled, positively-acknowledged fashion to provide a reliable TCP proxy function. Also describe the NORM congestion control options here.

3. Usage

Typically, *norp* can be run in its default configuration with no command-line options required. However, a number of options are available via the command-line. This is a summary of *norp* usage:

```
norp [interface <ifaceName>][address <publicAddr>][sport <socksPort>][port <norpPort>]
     [norm {on|off}][id <normId>][nport <normPort>][cce | ccl | rate <bits/sec>]
     [limit <bits/sec>][persist <seconds>][segment <segmentSize>]
     [correspondent <remoteNorpAddr>][forward <tcpPort>,<destAddr>/<destPort>[,<remoteNorpAddr>]]
     [version][debug <level>][trace][dlog <debugLog>][lport <localNorpPort>][rport <remoteNorpPort>]
```

The *norp* program command-line options include ...

Table 1. norp Command-line Options

`interface <interfaceName>`	The given `<interfaceName>` specifies the name (or IP address) of the host network interface *norp* uses as its "public" proxy address. Currently a single interface may be designated for an instance of *norp*. Future version of *norp* may allow for multiple interfaces to be designated depending upon the source and/or destination address of SOCKS proxy connections.
`address <publicAddr>`	This is similar to the "`interface`" command, but allows a specific address to be set. For example, hosts with multiple addresses assigned may wish to use a specific address for proxy functions.
`sport <socksPort>`	This command is used to specify the port number on which the *norp* server listens for SOCKS client connections. The default port is currently port number 7000.
`port <norpPort>`	This command is used to specify the UDP port number used for *norp* session setup signaling. The default *norp* UDP signaling port is 7001. The configured *norp* port number (and NORM port number) MUST be unblocked by any network firewalls between *norp* peers. The given port number is used by *norp* to listen for remote connection request and is used as the destination port to signal remote *norp* peers.
`norm {on | off}`	By default, *norp* attempts to signal the SOCKS connection endpoint to setup a NORM transport connection to handle reliable data transfer for the TCP connection being instantiated. This command with the "`off`" argument will disable this function and *norp* will act as a conventional SOCKS proxy server.
`id <normId>`	By default, *norp* will attempt to self-configure a NORM protocol node identifier using the IP address of the server host. This command allows a specific NORM node identifier value to be set. It is generally not necessary to explicitly set this value for *norp* unicast proxy connections.
`nport <normPort>`	This command can be used to specify a UDP port number that will be used for NORM protocol transport connections. The default NORM port number used by *norp* is 7002. The configured NORM port number (and *norp* UDP signaling `port` number) MUST be unblocked by any network firewalls between *norp* peers.
`cce`	This option enables NORM-CCE congestion control operation that uses Explicit Congestion Notification (ECN) information for NORM protocol end-to-end transmission rate adaption. This is an alternative to the TCP-friendly congestion control mechanism used for NORM by default. Routers in the path of the *norp* peers using the NORM-CCE option MUST be configured for ECN packet marking in response to congestion.
`ccl`	This option enables experimental NORM-CCL ("Loss Tolerant") congestion control operation that uses some simple heuristics to try to differentiate packet loss due to congestion versus duo to channel bit errors. This is another alternative to the TCP-friendly congestion control mechanism used for NORM by default. No special intermediate system configuration is required, and while more loss tolerant than the default TCP-friendly behavior, is not as effective as the NORM-CCE mode of operation.
`rate <bits/sec>`	This option causes *norp* to use a preset and fixed transmission rate for each proxied data flow (e.g. TCP connection). This should only be used when the network connectivity usage is carefully pre-planned and previsioned for the expected (i.e. *a priori* known) flows. At this time, one common transmission rate is used for all flows.

`limit <bits/sec>`	This option sets a limit for the *cumulative* transmit rate for *all* flows that *norp* is proxying. For automated congestion operation, this can also work to "jump start" the usual "slow start" transport rate control by setting the lower bound of rate adjustment based on the limit `<bits/sec> / <numFlows>`. For example, a single flow will immediately "jump" to close the full limit rate, while the second of two flows would "jump" to half of the "limit" rate. Also, by setting a limit based on *a priori* connectivity information, this can avoid rate adjustment "overshoot" and help congestion control operate more effectively as compared to a "blind" situation. IMPORTANT: This options should only be applied when the connectivity path is well known and the impact of the lower bound enforcement here will not adversely impact other network traffic flows. A future option may be provided to further reduce or eliminate the lower bound enforcement that would eliminate this concern in less controlled network deployments. A limit value of "-1.0" (default) disables the limit enforcement.
`segment <segmentSize>`	This option sets the NORM protocol maximum packet payload size where is *<segmentSize>* is in units of bytes. . For *norp* that uses the NORM_OBJECT_STREAM, the maximum NORM UDP payload size is 40 bytes of NORM header plus the configured segment size. The resultant total maximum IPv4 UDP packet size (including IP and UDP headers) is then 28 + 40 + *<segmentSize>* bytes. For IPv6, the resultant maximum packet size is 48 + 40 + *<segmentSize>* bytes. The default NORM segment size, if this option is not invoked, is 1400 bytes, resulting in NORM UDP packets with 1440 byte payloads. Thus, for IPv4 that has 28 bytes of IP + UDP header, this results in a maximum *norp* packet size of 1468 bytes while, for IPv6, the maximum *norp* packet size would be 1488 bytes.
`correspondent <remoteNorpAddr>`	This option causes *norp* to "route" connections through a *norp* peer at the specified *<remoteNorpAddr>*. This is an alternative to the default behavior where *norp* attempts to connect directly to the connection destination addresses.
`persist <seconds>`	This option controls how persistently norp attempts to deliver data to the remote endpoint when the remote endpoint fails to acknowledge reception. A persist value of -1 makes norp infinitely persistent and the corresponding norp session remains in place until all data is delivered. If not, an orphaned session will remain in place if the remote endpoint is permanently disconnected. The default persist value is 120 seconds (2 minutes).
`forward <tcpPort>,<dstAddr>/<dstPort>[,<norpAddr>]`	This command sets up a "preset" TCP proxy (non-SOCKS) port forwarding session by listening on the specified TCP *<tcpPort>* for connections and then connecting to the given remote *<dstAddr>/<dstPort>*. Optionally, a separate remote *<norpAddr>* may be given. Otherwise, a *norp* proxy connection is attempted to the given *<dstAddr>* platform on the *norp* *<port>* (or *<rport>* if specified). Note that multiple such "preset" proxy sessions may be specified on the command-line and each "preset" proxy session can handle multiple connections as needed.
`debug <debugLevel>`	This command can be used to control the verbosity of *norp* debug logging output. Generally, the range of the value is 0-12. A higher value results in more verbose, detailed debug output.
`trace`	This command enables NORM send and receive packet trace logging.
`dlog <fileName>`	This command can be used to direct *norp* debug logging output to a given file. The default *norp* debug logging is to STDERR.
`lport <localNorpPort>`	This command can enable single host, loopback testing by a having *norp* listen on a different port number than which it uses as the destination port for remote *norp* peer signaling. E.g., two *norp* instances on a single machine can be set up with unique `<localNorpPort>` values and then use the "`rport`" command to specify each other's destination *norp* port numbers.

`rport <remoteNorpPort>`	This command is intended to be used in conjunction with the "`lport`" command to allow separate specification of the destination port number used for remote *norp* peer signaling.

3.1. SOCKS Client Configuration

TBD - provide overview and examples (for specific SOCKS clients of note such as Dante, Proxifier, etc) of SOCKS client configuration

3.2. Usage Examples

The SOCKS client(s) must be configured to use the *norp* server unless a preset TCP port `forward` is specified. For example the Dante proxy distribution (available from http://www.inet.no/dante/) has a *socksify* command that is installed and can be used to launch existing network applications so their socket communications are directed through the configured server. With Dante, a SOCKS configuration file (typically `/etc/socks.conf`) or the `SOCKS5_SERVER` environment variable can be used to set the server address and port number.

The *norp* "server" is a lightweight module and can be installed on the same end systems requiring the performance benefits of NORM transport. In this case the SOCKS client server configuration is the loopback address and *norp* SOCKS port number (i.e. `127.0.0.1:7000`). The locally installed norp SOCKS server will signal remote network destinations (e.g., upon TCP connection initiation) to determine if the destination is *norp*-capable. If possible, it will establish a NORM-connection to the remote *norp* correspondent that connects to the final destination. Otherwise a direct TCP connection (or UDP relay) will be made to the remote destination.

4. The "NORP" UDP Signaling Message Format

The *norp* proxy uses UDP signaling to confirm presence of a remote *norp* peer and to set up (and tear down) NORM transport protocol sessions to support the proxied TCP (and eventually UDP) transport connections. The norp instance originating a SOCKS session request is referred to here as the "originator" and the remote norp peer to which the request is directed is referred to as the "correspondent". The *norp* "originator" is the server associated with the SOCKS client making a request while the "correspondent" establishes connections with the remote SOCKS destination.

The following UDP payload format is used for NORP signaling:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   msgType     |    reserved   |           sessionId           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          normNodeId                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          normSrcPort          |           normDstPort         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          timestamp_sec                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          timestamp_usec                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           content ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NORP message types include:

SOCKS_REQ The message content contains a SOCKS5 Request message from the "originator" to the "correspondent" *norp* server.

ACK_REQ The message is used to acknowledge receipt of a SOCKS_REQ message. There is no "content"

SOCKS_REP The message content contains a SOCKS5 Reply message from the "correspondent" *norp* server.

ACK_REP The message is used to acknowledge receipt of a SOCKS_REP message. There is no "content".

ORIG_END This message indicates the "originator" *norp* server is terminating the given session. There is no "content".

CORR_END This message indicates the "correspondent" *norp* server is terminating the given session. There is no "content".

ACK_END This message is used to acknowledge receipt of either an ORIG_END or CORR_END message. There is no "content".

TBD - describe NORP signaling and the message format given here.

5. Future Plans

There are a number of additional features and refinements planned for the *norp* implementation. Some of these include:

1. Source / destination configuration and "routing" options

2. Data compression options

3. Security features