Packet White Paper: 222
Status: Informational
Date: June 2023
Author: Paula Dowie

Remote Host Protocol Version 2

Abstract

Remote Host Protocol (RHP) is one of the Application Programming Interfaces for the "XRouter" Amateur Packet Radio networking software. RHP allows applications, sited either locally to XRouter, or remotely from it, to use XRouter as a multi-protocol "packet engine". RHP allows developers to write Packet Radio applications without having to write a Packet networking stack.

RHP version 1, described in [1] used an extensible binary packet format. This memo describes version 2, which uses JSON messages instead. Version 2 is more verbose than version 1, but is more suited to modern paradigms.

Status of This Memo

This memo provides information for the Packet Radio community. This memo does not propose a standard, but it describes an existing one. Distribution of this memo is unlimited.

Copyright Notice

Table of Contents

1. Introduction

RHP is the acronym for REMOTE HOST PROTOCOL, so called because it allows a "host" application to be located remotely from XRouter, effectively using XRouter as a multi-protocol "packet engine".

The protocol allows applications, sited either locally to XRouter, or remotely from it, to access the XRouter protocol stack at many ISO layers, including layer 7, i.e. XRouter's command line interface.

The original RHP, described in [1], used an extensible binary packet format. It was perfectly adequate, but very few applications were developed to use it.

The world has moved on a long way since 2004. Today's application developers have little desire to craft binary packets, and it is likely that not many possess the skills to do so.

This memo describes RHP version 2 (RHP2), which uses JSON messages instead of binary ones, and outlines the JSON message format.

1.1. Motivation

The motivation for this protocol upgrade was the need to provide a modern socket-like API for the "XRouter" multi protocol Amateur Packet Radio networking software, and thus to stimulate the development of new applications.

Modern applications tend to be browser-based. The age of standalone applications is over, and very few "packeteers" are interested in command-line interfaces.

Whilst XRouter provides both HTTP and REST interfaces, both protocols have drawbacks. HTML pages tend to be fixed-format. The node author designs the pages, and the users are stuck with them. The sysop might be able to tweak the CSS, but the basic layout is set in stone. Node authors make terrible UI designers.

REST APIs are better, because they allow application developers to access the raw data, and display it however they wish. The data is decoupled from its presentation. Very little skill is required to write Javascript, hence there are plenty of UI developers capable of doing so. Hence more chance that someone will write a new UI.

Both HTTP and REST behave poorly when it comes to SESSIONS, as they are stateless by design. There are workarounds, but they are cumbersome and barely satisfactory.

RHP allows developers to write Packet Radio applications without having to write Packet Radio networking stacks. What's more, the applications don't have to run on the same machine or operating system as the Packet stack.

RHP is NOT a user interface. It is a machine-to-machine interface, mainly for ISO layers below the application layer. It treats XRouter as nothing more than a "packet engine".

RHP version 2, sometimes referred to as "RHP2" uses JSON instead of a binary format, and adds the option of operation via Websockets. JSON is easy to read and debug, and modern languages have plenty of inbuilt support for both JSON and Websockets.

1.2. RHP Sockets

At the heart of RHP is the concept of "sockets", i.e. communication endpoints. These are functionally similar to Berkeley (BSD) sockets, but with some extra features.

The diagram below depicts a client using RHP to control a socket, and to exchange data with it. In turn the socket is interacting with a target system using the AX25 protocol. Data is passed between the client and target via the socket, whilst RHP control signals pass only between client and server.

```
                        .--------------.
  .--------.            |  .--------.  |           .--------.
  | client |<------------->| socket |<------------>| Target |
  '--------'    RHP     |  '--------'  |   AX25    '--------'
                        '--------------'
                        Server (XRouter)
```

Sockets must be "opened" before use, and "closed" after use. When a socket is opened, a numeric "handle" is returned, which must be used for all subsequent operations via that socket.

When opening a socket, the client must specify a "protocol family", and a "mode".

The currently available protocol families are as follows:

Mnemonic	Layer	Usage
UNIX	7	XRouter CLI and applications
INET	3/4	TCP/UDP/ICMP/IP/DNS etc
AX25	2	AX25, APRS, Digipeating, custom protocols
NETROM	3/4	NetRom datagrams, streams, custom protocols

The available socket "modes" are as follows:

Mnemonic	Meaning
STREAM	Ordered, reliable octet stream, (LAPB, TCP, CLI etc)
DGRAM	Unreliable datagram (AX25 UI, UDP, NetRom L3 etc.)
SEQPKT	Sequenced, reliable packets (ax25 only)
CUSTOM	User specified protocol
SEMIRAW	Addresses + raw payload
TRACE	Decoded headers plus payload data
RAW	Complete raw packet, no separate headers

Not all modes are available for all protocol families.

Within the AX25 family, stream mode would be used for normal ax25 connections, datagram mode for APRS, trace mode for monitoring packet activity, and raw mode for custom packet tracing / injection.

Within the NETROM family, stream mode would be used for normal L4 connections, DGRAM for NDP (NetRom Datagram Protocol) datagrams, CUSTOM for "protocol extension" frames, SEMIRAW for L3 headers plus payload, TRACE for tracing NetRom from layer 3 up, and RAW for raw layer 3 operations, such as Nodes and INP3 broadcasts.

Within the INET family, STREAM is for TCP connections, DGRAM for UDP, CUSTOM for IP datagrams transporting a specified protocol, TRACE for tracing up from IP layer, and RAW for generating / receiving IP datagrams without using XRouter's packet assembly/disassembly.

Clients may open multiple sockets at once, provided they are not identical. For example, a TRACE socket may be opened to monitor packet activity, while at the same time a DGRAM socket may be opened to send and receive UI frames, while several STREAM sockets may be opened, either to make connections to other systems, or to listen for incoming connections from them.

The only limitation is that a client cannot open more than one TRACE or RAW socket on the same PORT, or more than one DGRAM socket with the same PORT and LOCAL address, or more that one STREAM socket with identical PORT, LOCAL and REMOTE addresses.

Some of these limitations also apply to multiple CLIENTS. For instance, only ONE client may open a STREAM listener on the same port using the same local address. This is because XRouter would have no way of knowing which client to send the incoming packets to.

RHP sockets are "non-blocking". For example, a connect request, INITIATES a connection and returns the result of that operation, but the connection may not actually complete until some time later.

All sockets "owned" by a client are closed when the RHP client connection is terminated.

1.3. RHP2 Messages

Both versions of RHP use a single, persistent, TCP connection between the client and XRouter. The connection normally uses TCP port 9000, but this may be changed using the RHPPORT directive in XROUTER.CFG.

In RHP2, JSON "messages" pass bidirectionally across the TCP stream, to manage packet connections and transfer data in both directions.

The protocol may also be used via a Websockets connection, using the same TCP port number.

The endpoint for RHP2 over Websockets is ws://{host}:{port}/rhp, e.g. "ws://localhost:9000/rhp". Future versions may duplicate this on the regular HTTP port.

Within "normal" RHP2, JSON messages are "framed" by a simple two-byte "frame length", sent high byte first. e.g. the two bytes 0x01 0x20 indicate that the message which follows them is 288 bytes long.

```
          .---------------------------.
          | lenH | lenL | RHP Message |
          '---------------------------'
      Bytes: 1      1     <--- len --->
```

RHP Message Framing

The framing for RHP2 in WebSockets is similar, albeit with a more complex header.

2. Overview of RHP2 Message Types

The most commonly used RHP2 message "types" are as follows:

AUTH (Client to server)
Sends credentials to authenticate the client. Only required if the client has IP address in a public range, and is not whitelisted in ACCESS.SYS.

AUTHREPLY (Server to client)
Indicates success/failure of auth request.

OPEN (Client to server)
Initiates an "active" connection to a specific "target" system, or a "passive" listener to wait for incoming connections.

OPENREPLY (Server to client)
Indicates success/failure of an OpenRequest, and returns a "handle" for further operations.

ACCEPT (Server to client)
Conveys details of an incoming connection. Sent by "listener" sockets only.

STATUS (Bidirectional)
Conveys flags such as "connected" and "busy".

STATUSREPLY (Server to client)
Indicates failure of a status request from client.

SEND (Client to server)
Sends data to a target system with who a connection has been established, if a format dictated by the MODE of the socket.

SENDREPLY (Server to client)
Acknowledges a SEND message.

RECV (Server to client)
Sends data, received from a connected target system, to the client. Each RECV message contains the payload from one packet.

CLOSE (Client to server
Requests closure of a socket or connection.

CLOSEREPLY (Server to client)
Acknowledges closure of a socket or connection.

The following BSD-style message types are also supported:

SOCKET (Client to server)
Opens a socket.

SOCKETREPLY (Server to client)
Indicates success/failure of an SOCKET request, and returns a "handle" for further operations.

BIND (Client to server)
Associates a local address with the socket

BINDREPLY (Server to client)
Indicates success/failure of BIND request.

LISTEN (Client to server)
Initiates "listen" mode on a STREAM socket, awaiting incoming connections.

LISTENREPLY (Server to client)
Indicates success/failure of LISTEN request.

CONNECT (Client to server)
Initiates a connection to a target system (STREAM mode), or binds a remote address (DGRAM mode).

CONNECTREPLY
Indicates success/failure of CONNECT request.

SENDTO (Client to server)
Sends data to a target system (DGRAM mode only)

SENDTOREPLY (Server to client)
Indicates success/failure of SENDTO operation.

3. RHP Message Types In Detail

All RHP2 messages MUST be in JSON format, beginning with the opening curly bracket '{', and ending with the closing curly bracket '}'.

Whitespace (spaces, tabs, newlines, etc.) is allowed within the JSON messages, but is not mandatory. Message examples in this document may include white space for clarity.

All fields of an RHP message are mandatory unless otherwise stated.

The order of fields within a message is unimportant.

All messages MUST have a "type" field.

The "id" field is optional. If present in a request, XRouter ALWAYS replies, returning the same ID in the reply. Using a different ID in each request allows replies to be matched with the corresponding requests, allowing asynchronous "pipelining" of requests.

If the "id" field is omitted, the only replies, other than OPENREPLY, will be those that contain a non-zero error code.

3.1. The AUTH Message

The AUTH request sends credentials to authenticate the client. It is only required if the client has an IP address in a public range, and is not whitelisted by an entry in ACCESS.SYS.

Fields:

Name	Type	Value
"type"	string	"auth"
"id"	integer	(optional)
"user"	string	{user's callsign}
"pass"	string	{user's password}

Example: {"type": "auth", "user": "g9zzz", "pass": "petunias"}

3.2. The AUTHREPLY Message

This message is sent from server to client, in response to an AUTH message or any other request received in unauthorised state.

Fields:

Name	Type	Value
"type"	string	"authReply"
"id"	integer	Same as request (if present)
"errCode"	integer	0 or 14
"errText"	string	"Ok" or "Unauthorised"

If the usercall and password in an AUTH request match an entry in USERPASS.SYS, the value of "errCode" will be 0, and the value of "errText" will be "Ok". Otherwise, the value of "errCode" will be 14, and the value of "errText" will be "Unauthorised".

Examples:

```
{"type": "authReply", "id": 7, "errCode": 0, "errText": "Ok"}
{"type": "authReply", "errCode": 14, "errText": "Unauthorised"}
```

3.3. The OPEN Message

The OPEN request, from client to server, is used to open a socket, and optionally to initiate a connection or start a listener. It performs, in a single operation, the equivalent of the BSD SOCKET, BIND, LISTEN and CONNECT functions, depending on which fields are supplied.

Fields:

Name	Type	Value
"type"	string	"open"
"id"	integer	Serial number of request (optional)
"pfam"	string	Protocol family (see section 1.2)
"mode"	string	"stream", "dgram", "trace" or "raw"
"port"	string	port identifier (port number in XRouter)
"local"	string	Local address
"remote"	string	Remote address (active open only)
"flags"	integer	Option flags

Values for option flags:

0x00	Passive open (listen)

0x01	Trace Incoming frames (modes RAW and TRACE)
0x02	Trace Outgoing frames (modes RAW and TRACE)
0x04	Trace Supervisory frames (mode TRACE only)
0x80	Active open (connect)

Passive open with unspecified remote accepts any call
Passive open with remote address accepts only that addr

Examples:

Open an ax25 connection to GB7GLO from callsign G8PZT-5:

```
{
  "type": "open",
  "id": 22,
  "pfam": "ax25",
  "mode": "stream",
  "port": 2,
  "local": "g8pzt-5",
  "remote": "gb7glo",
  "flags": 128
}
```

Open an AX25 TRACE socket on port 4:

```
{
  "type": "open",
  "id": 1,
  "pfam": "ax25",
  "mode": "trace",
  "port": "4",
  "flags": 7
}
```

3.4. The OPENREPLY Message

This message is sent from server to client in response to an OPEN request. It indicates success or failure of the request, and if the socket was successfully opened it returns a "socket handle" for further operations.

Fields:

Name	Type	Value
"type"	string	"openReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

Example:

```
{
  "type": "openReply",
  "id": 22,
  "handle": 3,
  "errcode": 0,
  "errtext": "ok"
}
```

3.5. The ACCEPT Message

This message is sent from server to client, by "listener" sockets only. It conveys details of an incoming connection.

Fields:

Name	Type	Value
"type"	string	"accept"
"seqno"	integer	Sequence number of this message
"handle"	integer	Socket handle of the listener
"child"	integer	Socket handle of the new connection
"remote"	string	Remote (caller's) address
"local"	string	Local (listener) address
"port"	integer	Xrouter port number of new connection

Example:

```
{
  "type": "accept",
  "seqno": 347,
  "handle": 3,
  "child": 7,
  "remote": "G4FPV-5",
  "local": "g8pzt-1",
  "port": 4
}
```

3.6. The STATUS Message

This message can be used in either direction. It can be sent asynchronously from server to client, to convey flags such as "connected" and "busy". It can also be sent from client to server to request a status message from the server. In this second case, the server only sends a STATUSREPLY message if the request fails.

Fields (server to client):

Name	Type	Value
"type"	string	"status"
"seqno"	integer	Sequence number of this message
"handle"	integer	Socket handle
"flags"	integer	socket flags (server to client only)

Socket flags:

CONOK	1	OK to accept (listeners only)
CONNECTED	2	Downlink is connected
BUSY	4	Not clear to send

Example: (server to client)

```
{
  "type": "status",
  "seqno": 348,
  "handle": 3
  "flags", 2
}
```

Fields (client to server):

Name	Type	Value
"type"	string	"status"
"id"	integer	Serial number of request (optional)
"handle"	integer	Socket handle

Example: (client to server)

```
{
  "type": "status",
  "id": 23,
  "handle": 3
}
```

3.7. The STATUSREPLY Message

This message is only sent from server to client, only in reply to a STATUS request, and only if the request fails.

Fields:

Name	Type	Value
"type"	string	"statusReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code
"errtext"	string	Error text

Example:

```
{
  "type": "statusReply",
  "id": 23,
  "handle": 3
  "errcode": 12,
  "errtext": "Invalid handle"
}
```

3.8. The SEND Message

The SEND message sends data from client to server, for onward transmission to another system.

Fields:

Name	Type	Value
"type"	string	"send"
"id"	integer	Serial number of request (optional)
"handle"	integer	Socket handle
"data"	string	Data to be sent

Additional fields for datagram mode only:

"port"	string	Destination port
"local"	string	Local address
"remote"	string	Remote address

Reserved and control characters in the "data" field MUST be JSON-escaped. The total size of the message MUST NOT exceed 65535 bytes.

Example:

```
{
  "type": "send",
  "id": 23,
  "handle": 3,
  "data": "Hello Fred, are you there?"
}
```

3.9. The SENDREPLY Message

The SENDREPLY message is sent from server to client in response to a SEND message, to convey the result of the operation.

Fields:

Name	Type	Value
"type"	string	"sendReply"
"id"	integer	Matches ID in SEND request (optional)
"handle"	integer	Socket handle
"errcode"	integer	Error number
"errtext"	string	Description of the error
"status"	integer	Status flags (STREAM only)

Status flags:

CONNECTED	2	Downlink is connected
BUSY	4	Not clear to send

Example:

```
{
  "type": "sendReply",
  "id": 23,
  "handle": 3,
  "errcode": 0,
  "errtext": "Ok",
  "status": 2
}
```

3.10. The RECV Message

The RECV message is sent asynchronously from server to client, to convey data that has been received from a remote system. Each RECV message contains the payload from one ax25 packet. This message type is also used to convey TRACE data if the socket mode is TRACE.

Fields:

Name	Type	Value
"type"	string	"recv"
"seqno"	integer	Sequence number of this message
"handle"	integer	Socket handle
"port"	string	Port it was rcvd on (datagram only)
"action"	string	"sent" or "rcvd" (RAW & TRACE only)
"data"	string	Data rcvd from remote system

RAW and TRACE sockets can monitor both sent and received traffic, hence the "action" member.

Example RECV for a STREAM socket:

```
{
  "type": "recv",
  "seqno": 349,
  "handle": 3,
  "data": "Yes I'm here, what's up?",
}
```

Example trace representing "[4] T: G8PZT-1>G8PZT: <RR R F R1>":

```
{
  "type": "recv",
  "seqno": 349,
  "handle": 1,
  "action": "sent",
  "port": "4",
  "srce": "G8PZT-1",
  "dest": "G8PZT",
  "ctrl": 33,
  "frametype": "RR",
  "rseq": 1,
  "cr": "R",
  "pf": "F",
}
```

3.11. The CLOSE Message

The CLOSE message is sent from client to server, to request closure of a socket or connection. It can also be sent asynchronously from server to client to inform the client that a connection has been closed by the remote link partner.

Fields:

Name	Type	Value
"type"	string	"close"
"id"	integer	(client to server only) (optional)
"seqno"	integer	Sequence number (server to client only)
"handle"	integer	Socket handle

Example (client to server):

```
{
  "id": 3,
  "type": "close",
  "handle": 3
}
```

Example (server to client):

```
{
  "type": "close",
  "seqno": 350,
  "handle": 3,
}
```

3.12. The CLOSEREPLY Message

The CLOSEREPLY message is sent from server to client, in response to a CLOSE message, to acknowledge closure of a socket or connection.

Fields:

Name	Type	Value
"type"	string	"closeReply"
"id"	integer	matches the one in the CLOSE request
"handle"	integer	Socket handle
"errcode"	integer	Error number
"errtext"	string	Description of the error

Example of a Successful Close:

```
{
  "id": 3,
  "type": "closeReply",
  "handle": 4,
  "errcode": 0,
  "errtext": "Ok"
}
```

Example of a Failed Close:

```
{
  "id": 3,
  "type": "closeReply",
  "handle": 0,
  "errcode": 12,
  "errtext": "Invalid handle"
}
```

3.13. The SOCKET Message

The SOCKET request is sent from client to server, to open a socket on the XRouter networking stack. It is similar to the OPEN request without the extra fields.

Fields:

Name	Type	Value
"type"	string	"socket"
"id"	integer	Serial number of request (optional)
"pfam"	string	Protocol family
"mode"	string	"stream", "dgram", "trace", "raw" etc

For protocol family and mode values, see section 1.2.

Examples:

Open an ax25 stream socket:

```
{
"type": "socket",
"id": 21,
"pfam": "ax25",
"mode": "stream"
}
```

Open a NetRom TRACE socket:

```
{
"type": "socket",
"id": 1,
"pfam": "netrom",
"mode": "trace"
}
```

3.14. The SOCKETREPLY Message

A SOCKETREPLY message is sent from server to client in response to a SOCKET request. It indicates the success or failure of the request, and if the socket was successfully opened it returns a "socket handle" for further operations.

Fields:

Name	Type	Value
"type"	string	"socketReply"
"id"	integer	Same as request (if present)
"handle"	integer	Handle of newly-created socket (*)
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

(*) The "handle" field is only present if the request was successful.

3.15. The BIND Message

The BIND request is sent from client to server, to associate a local address with the socket. Once bound, the local address may be used in subsequent LISTEN, CONNECT and SENDTO operations.

Fields:

Name	Type	Value
"type"	string	"bind"
"id"	integer	Serial number of request (optional)
"local"	string	Local address
"port"	string	port identifier (port number in XRouter)

The "port" field is only required for AX25. If supplied for other types of socket, that port MUST exist.

The local address MUST NOT be identical to any of XRouter's addresses (using the same callsign with a different SSID is OK).

If the socket is already bound, the request will fail.

3.16. The BINDREPLY Message

The BINDREPLY message is sent from server to client. It indicates the success or failure of a BIND request.

Fields:

Name	Type	Value
"type"	string	"bindReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

3.17. The LISTEN Message

The LISTEN request is sent from client to server. On STREAM sockets it causes the socket to start waiting for incoming connections. On other types of socket, it begins the process of packet reception.

Fields:

Name	Type	Value
"type"	string	"listen"
"id"	integer	Serial number of request (optional)
"flags"	integer	Option flags

Values for option flags:

0x01	Listen Incoming (modes RAW and TRACE)
0x02	Listen Outgoing (modes RAW and TRACE)
0x04	Trace Supervisory frames (AX25 TRACE only)

3.18. The LISTENREPLY Message

The LISTENREPLY message is sent from server to client. It indicates the success or failure of a LISTEN request.

Fields:

Name	Type	Value
"type"	string	"listenReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

3.19. The CONNECT Message

The CONNECT request is sent from client to server. It initiates a connection to a target system (STREAM mode), or binds a remote address (DGRAM mode).

Fields:

Name	Type	Value
"type"	string	"connect"
"id"	integer	Serial number of request (optional)
"remote"	string	Remote address

A local address MUST be bound before CONNECT is issued. XRouter does not auto-bind local addresses.

3.20. The CONNECTREPLY Message

The CONNECTREPLY message is sent from server to client. It indicates the success or failure of a CONNECT request.

Fields:

Name	Type	Value
"type"	string	"ConnectReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

3.21. The SENDTO Message

The SENDTO request is sent from client to server. It sends data to a specified target system. It is allowed only on DGRAM sockets.

If a local address is bound, only the remote address need be supplied and vice versa. If both addresses are bound, the SEND request may be used instead. Addresses that are specified in the SENDTO request take precedence over any bound addresses.

Fields:

Name	Type	Value
"type"	string	"sendto"
"id"	integer	Serial number of request (optional)
"handle"	integer	Socket handle
"data"	string	Data to be sent (*)
"port"	string	Destination port
"local"	string	Local address
"remote"	string	Remote address
"tos"	integer	Type of service (INET only)

(*) Reserved and control characters in the "data" field MUST be JSON-escaped. The total size of the message MUST NOT exceed 65535 bytes.

Example for AX25 DGRAM socket, with bound local address and port:

```
{
  "type": "sendto",
  "id": 23,
  "handle": 3,
  "remote": "g1frd-6",
  "data": "\U0008Hello Fred, are you there?\r"
}
```

3.22. The SENDTOREPLY Message

The SENDTOREPLY message is sent from server to client. It indicates the success or failure of SENDTO operation. A SENDTO might fail if the handle is missing or invalid, or an address is missing or invalid, if the transmit queue is too full and so on. In the latter case the message may succeed if retried later.

Fields:

Name	Type	Value
"type"	string	"sendtoReply"
"id"	integer	Same as request (if present)
"handle"	integer	Socket handle
"errcode"	integer	Error code (0 = no error)
"errtext"	string	Error text in words, e.g. "Ok"

4. Address Formats

The format of addresses supplied in "local" and "remote" fields depends on the address family as follows:

4.1. AX25 Family

The AX25 family uses simple callsigns, e.g. "g8pzt-1".

4.2. NETROM Family

The format for NETROM addresses is:

<usercall>[@nodecall][:svcnum]

Where "svcnum" is the NetRomX "service" number (Reference [2]).

Examples: "g8pzt-1@g8pzt" or "gb7pzt:23"

4.3. INET Family

The format for INET family addresses is <ipaddress>[:port]

Examples: "44.131.91.2" or "192.168.3.22:25"

5. List of Error Codes

You are advised to parse the error CODE, not the error text, as the latter may change in future versions.

Code	Text	Notes
0	"Ok"	No error
1	"Unspecified"	Catch-all error, might be transient
2	"Bad or missing type"	Unrecognised frame type, don't retry
3	"Invaiid handle"	Invalid socket handle, don't retry
4	"No memory"	No memory, try later
5	"Bad or missing mode"	Invalid "mode" in SOCKET or OPEN
6	"Invalid local address"	(in OPEN, SOCKET, or BIND)
7	"Invalid remote address"	(in OPEN or CONNECT)
8	"Bad or missing family"	Unsupported address family
9	"Duplicate socket"	Socket / connection already exists
10	"No such port"	Invalid port number in OPEN or BIND
11	"Invalid protocol"	(in OPEN or SOCKET)
12	"Bad parameter"	Bad or missing parameter
13	"No buffers"	Output queue full (retry later)
14	"Unauthorised"	Request requires AUTHorisation
15	"No Route"	No route to target (L4 / TCP open)
16	"Operation not supported"	e.g. SEND on a TRACE socket

6. Typical Session Flow

This section outlines the general flow of different type of session. It assumes that the client has already established connection with the server, either via direct TCP connection, or via Websockets, and has authenticated if necessary.

6.1. Outgoing Connection

An outgoing connection would proceed as follows:

a) Client sends OPEN message, specifying the radio port, plus source and destination calls (including digipeaters if required)

b) Server immediately replies with OPEN_REPLY, containing a socket "handle" for all subsequent operations on the socket.

c) If the connection succeeds, the server asynchronously sends a STATUS message indicating "connected". If the connection fails, the status message contains "disconnected" instead.

d) The client sends data to the connection using SEND messages, the payload of which is JSON-escaped so it can handle full binary.

e) SEND messages are acked using SEND_REPLY.

f) If the client sends too much data for the AX25 link to handle, the server sends a STATUS message with the BUSY flag set. When clear to send again, the server sends a STATUS message with the BUSY flag unset.

g) Data received from the downlink is sent to the client in RECV messages, with the payload JSON-escaped.

h) If the downlink initiates a disconnect, the server sends a STATUS message to the client with the CONNECTED flag set to "false". The client must now issue a CLOSE to dispose of the socket.

i) Alternatively, if the client wishes to terminate the connection it issues a CLOSE request, and the server responds with a CLOSE_REPLY.

6.2. Incoming Connection

An INCOMING connection proceeds as follows:

a) The client sends an OPEN message, specifying the XRouter PORT number and the local callsign. This creates a "listener" socket.

b) The server responds with a OPEN_REPLY message containing the socket handle (or an error code).

c) If someone connects to the callsign associated with the socket, the server immediately sends an ACCEPT message to the client, containing the socket handle and the callsign of the connectee.

d) The remainder of the connection proceeds as in (d) above.

7. Security Considerations

By default, RHP only allows usage by clients with "localhost" and "LAN" IP addresses. In this context LAN addresses are those in the ranges 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.

Clients with non-LAN IP addresses MUST authorise, by sending a valid AUTH packet before they can use any other RHP commands. The AUTH packet MUST contain a username/password pair which matches one stored in USERPASS.SYS.

Non-LAN clients MAY be granted access without AUTH by including the client's IP address in ACCESS.SYS. Beware of granting such access to ranges of IP addresses, unless you have control of that range.

At the time of writing, authorisation uses plain text usernames and passwords, simply because RHP was never intended for use on the public internet. Alternative authenication methods are planned for future versions.

8. Caveats

As this document is a first draft, it may be vague, incomplete or inaccurate. If it is not fit for purpose, please send feedback, to ensure the next draft is an improvement.

Not every use case has been exhaustively tested, so there may still be bugs in XRouter's implementation of RHP2. Please report them.

9. Feedback

Please send comments, criticisms, suggestions, hate mail etc to the following email address:

g8pzt@blueyonder.co.uk

10. References

[1] Dowie P., "Remote Host Protocol", PWP 144, December 2004.

[2] Dowie P., "NET/ROM Data Multiplexing", PWP 109, July 2001.