Specifications for Building a *Transparent* Firewall Proxy to Support RealAudio® Player

Introduction

This document provides information to allow firewall developers to support the RealAudio Player-Server communications. The information is provided for the sole purpose of designing firewall software which supports RealAudio systems.

There are two types of firewall proxies that can be built to support RealAudio, Transparent and Application-Level.

Transparent Proxy Firewalls

A Transparent Proxy Firewall operates by monitoring network traffic and only letting through connections matching certain protocols. The Transparent Firewall relies on knowing details of all protocols it will support. Transparent proxies can perform their function without the client or server applications being modified or configured. This document provides specifications for building this type of firewall proxy.

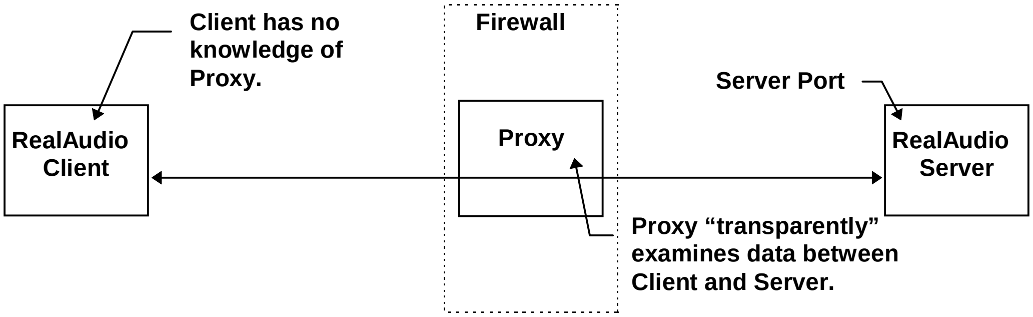

Client has no knowledge of Proxy.
Firewall
Server Port
RealAudio Client
Proxy
RealAudio Server
Proxy "transparently" examines data between Client and Server.

Figure 1 - Transparent Firewall

Application-Level Proxy Firewalls

Unlike a Transparent Proxy Firewall, an Application-Level Proxy firewall relies on the application inside the firewall having knowledge of the firewall proxy. All connections are made to the Proxy with the proxy then connecting to the desired external host. This means that an Application-Level Proxy needs to know about the client application connecting and what its Proxy protocol is, but does not need to know about the low level protocol details. For information on how to build an Application-Level Proxy, please refer to the document entitled, *Specifications for Building an Application-Level Firewall Proxy to Support RealAudio.*

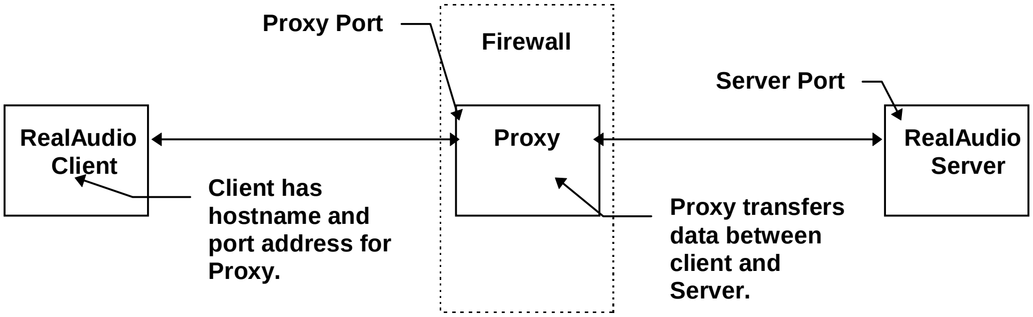

Proxy Port
Firewall
Server Port
RealAudio Client
Proxy
RealAudio Server
Client has hostname and port address for Proxy.
Proxy transfers data between client and Server.

Figure 2 - Application-Level Proxy

RealAudio Client / Server Communications

A RealAudio Client (Player) can use one of three methods for communicating with a RealAudio Server:

- Standard UDP
- Robust UDP
- TCP-Only

Standard UDP

The RealAudio Client (Player) sets up two network connections with the RealAudio Server, as shown in Figure 3. A full-duplex TCP connection is used for control and negotiation. A simplex UDP path from the RealAudio Server to the Player is used for audio data delivery. By using UDP for the audio, the RealAudio Server and Player can handle error correction instead of relying on the transport protocol. This allows a RealAudio Server to provide a better Audio stream when packet loss occurs.

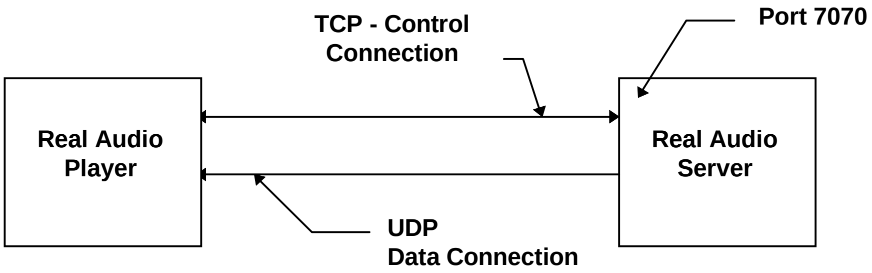


Figure 3 - RealAudio Client / Server Communications : Standard UDP Mode

Robust UDP

The RealAudio Client (Player) sets up three network connections with the RealAudio Server, as shown in Figure 4. A full-duplex TCP connection is used for control and negotiation. A simplex UDP path from the RealAudio Server to the Player is used for audio data delivery. A second simplex UDP path from the Client to the Server is used to request that the Server resend lost UDP audio data packets.

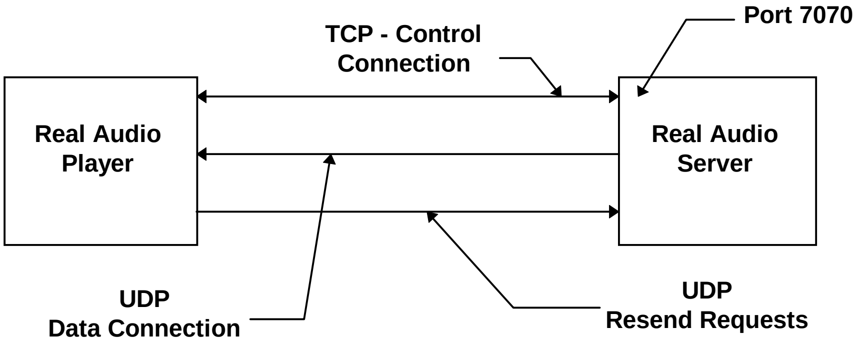


Figure 4 - RealAudio Client / Server Communications : Robust UDP Mode

TCP-Only

In TCP-Only mode, a single full-duplex TCP connection is used for both control and for audio data delivery from the RealAudio Server to the Player, as shown in Figure 5. The standard TCP connection port on a RealAudio Server is 7070.

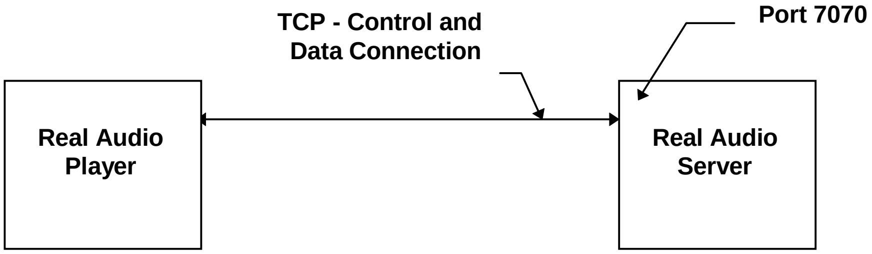


Figure 5 -RealAudio Client / Server Communications : TCP-Only Mode

Real Audio and Firewalls

A Transparent Proxy requires no explicit support in either the RealAudio Player or RealAudio Server. The Proxy must know about the protocol being used between the Player and Server, but no actual connection is made to the Proxy by either the Player or Server.

A Transparent Proxy, shown in Figure 6, must be able to detect and modify port requests being sent between the Player and the Server. Working at the packet level, the Proxy must capture, scan and modify TCP control messages. Unlike Application-Level Proxies, Transparent Proxies do not require the end user to configure the Player for the Proxy. Transparent Proxies can support RealAudio 1.0, 2.0, and 3.0 Players.

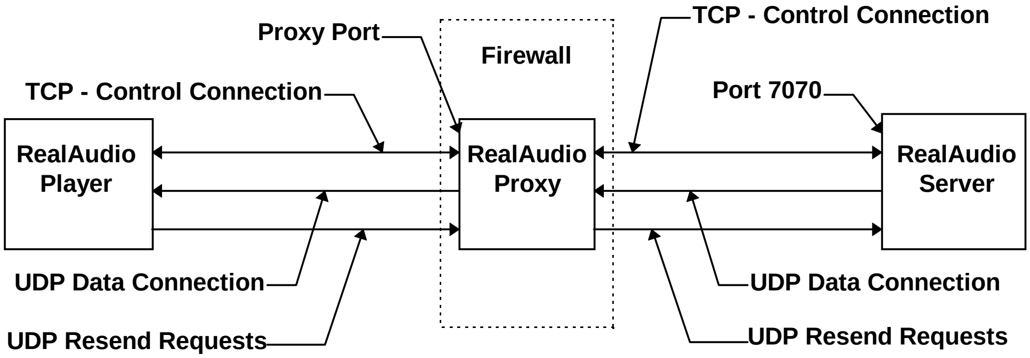


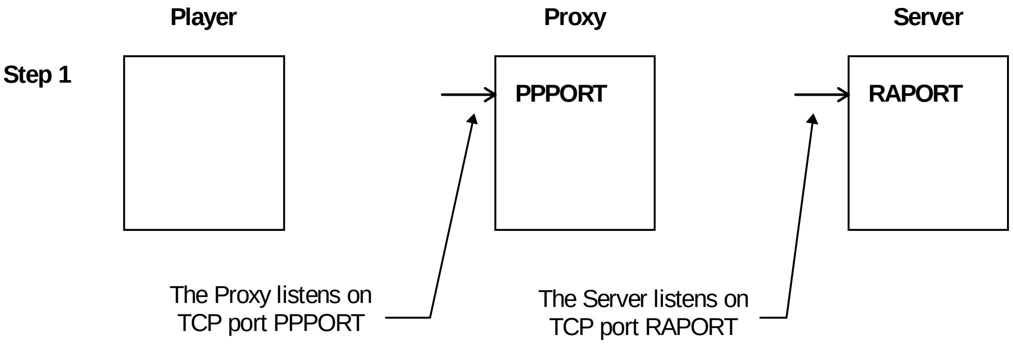
Figure 6 - RealAudio Transparent Proxy

Transparent Proxy Handshaking

Transparent Proxies, if built to the following specifications, will support RealAudio 1.0, 2.0, and 3.0 Players. The interactions between the Player and the Server are described in general terms below including schematic diagrams that show the progression of messages and actions in the interaction. The connections from prior steps in the diagrams are grayed out in subsequent steps where they are not an active part of that step. All descriptions of messages refer to structured RealAudio Proxy Protocol messages. These messages are defined in Table-2 RealAudio Handshake Protocol, that follows this sample description.

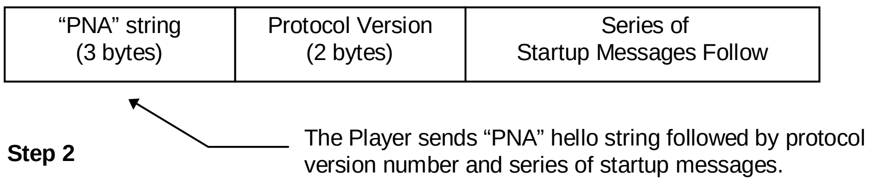
Handshake and Communications Description

1. The Proxy listens on its defined TCP port, PPPORT (standard is port 7070). The RealAudio Server is outside the firewall and is passively listening on TCP port RAPORT (standard is port 7070), for any incoming connections.

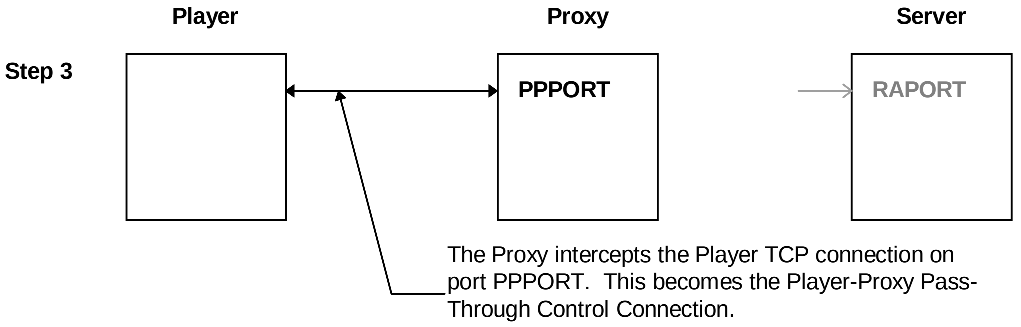


Player
Proxy
Server
Step 1
PPPORT
RAPORT
The Proxy listens on TCP port PPPORT
The Server listens on TCP port RAPORT

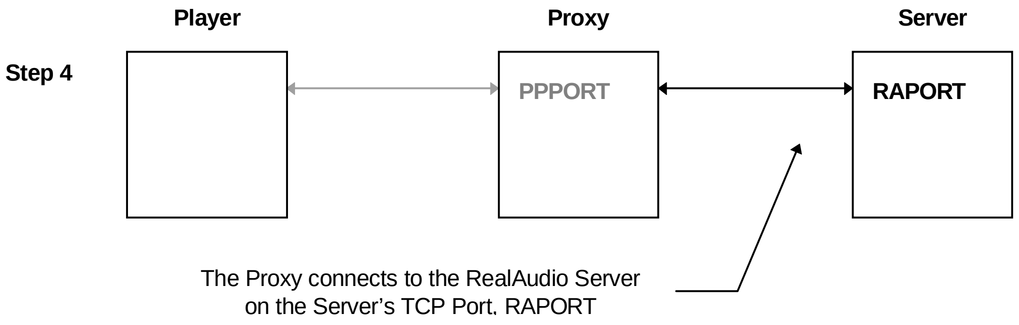
2. When the Player attempts an active TCP connect to the RealAudio Server by sending the string “PNA” (hex = 504e41) followed by the protocol version number (a two byte integer). All numeric values are encoded in network byte order. *Please note that firewalls built to these specifications will support all RealAudio protocol versions numbered less than 256.*


"PNA" string
(3 bytes)
Protocol Version
(2 bytes)
Series of
Startup Messages Follow
Step 2
The Player sends "PNA" hello string followed by protocol version number and series of startup messages.

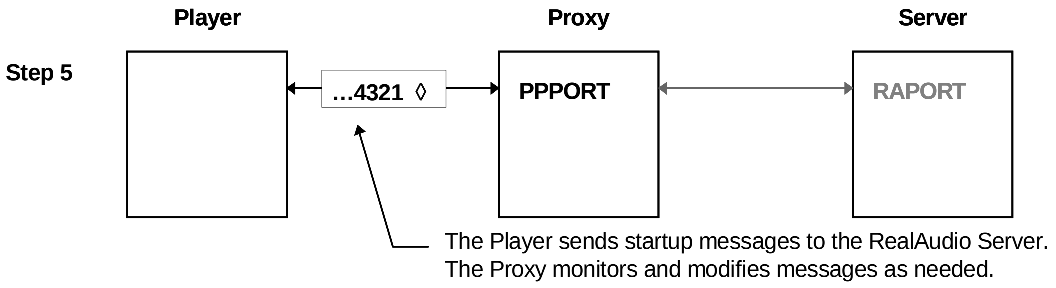
3. The listening Proxy identifies this action as a RealAudio request and responds accordingly. The Proxy converts this connection into a pass-through control connection between the Proxy and the Player.


Player
Proxy
Server
Step 3
PPPORT
RAPORT
The Proxy intercepts the Player TCP connection on port PPPORT. This becomes the Player-Proxy Pass-Through Control Connection.

4. The Proxy then opens a TCP connection to the RealAudio Server on TCP port RAPORT, using the IP addressing in the TCP packets coming from the Player. This completes the TCP pass-through control connection between the Player and the RealAudio Server using the Transparent Proxy.


Player
Proxy
Server
Step 4
PPPORT
RAPORT
The Proxy connects to the RealAudio Server on the Server's TCP Port, RAPORT

5. The Proxy must detect and modify specific startup messages being passed over the pass-through control connection. In particular, the Proxy must determine the type of connection (Standard UDP, Robust UDP, or TCP-Only) requested.


Player
Proxy
Server
Step 5
…4321 ◊
PPPORT
RAPORT
The Player sends startup messages to the RealAudio Server. The Proxy monitors and modifies messages as needed.

6. Startup messages are formatted as tuplets made up of:

    Message identifier (2 byte integer)
    Byte length (n) of the message (2 byte integer)
    Message (n bytes long)

All numeric values are encoded in network byte order as integers.

Step 6

Message ID Number (2 bytes)	Message Length (2 bytes)	Startup Message (n bytes)

Proxy needs to check for the following startup messages sent by the Player:

ID 1 = UDP port request
ID 7 = Robust UDP
ID 0 = End of start up messages

All other startup messages should be ignored by the Proxy and passed to the RealAudio Server unmodified.

Message ID 1 means that the Player is requesting either Standard UDP or Robust UDP audio delivery. The message value contains the UDP port number on which the Player expects to receive the data stream.

Message ID 7 means that the Player is requesting Robust UDP audio delivery. The message value contains the UDP port on the Player computer from which UDP resend requests will be sent.

Message ID 0 marks the end of the startup messages from the Player. The Player and Server continue to communicate over the TCP Control Connection.

Depending on the messages received, follow the corresponding steps that follow:

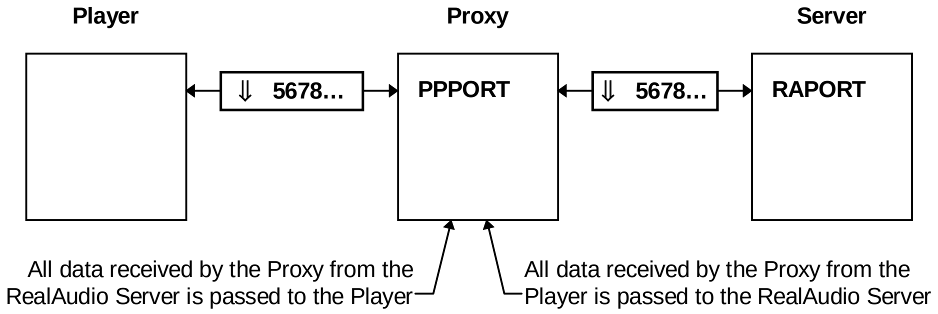
Message ID 1 and Message ID 7 before Message ID 0: **Robust UDP Connection**
Message ID 1 before Message ID 0: **Standard UDP Connection**
Message ID 0 only: **TCP-Only Connection**

Note The Player may send other startup messages before Message ID 0; these messages should be passed through unmodified to the Server.

TCP-Only Connection

7. The Proxy transfers every byte read from the Player on the TCP Control connection to the RealAudio Server on its TCP control connection. The Proxy also transfers all data received from the RealAudio Server to the Player.

TCP 7



All data received by the Proxy from the RealAudio Server is passed to the Player

All data received by the Proxy from the Player is passed to the RealAudio Server

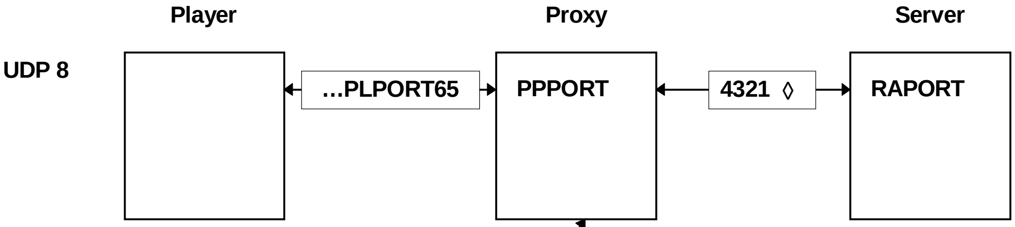
8. When either of the connections are closed, the other open connection in this Server - Proxy - Player interaction should also be closed. This terminates the Player - Proxy interaction and the Server - Proxy interaction.

Standard UDP Connection

The steps for the Standard UDP Connection also apply to a Robust UDP Connection. Additional steps for a Robust UDP Connection are in the next section.

7. The Proxy transfers every byte read from the Player on the TCP control connection to the RealAudio Server on its TCP control connection. The Proxy also transfers all data received from the RealAudio Server to the Player.

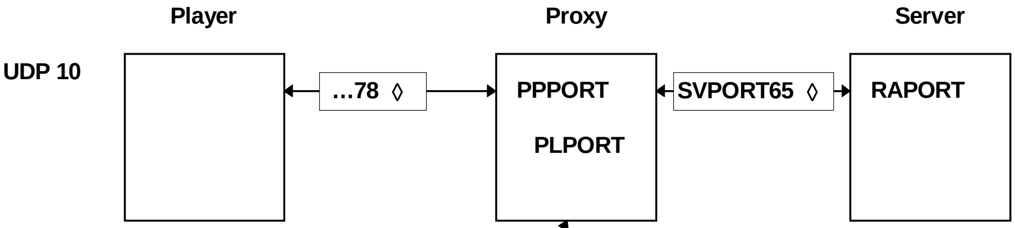
8. After detecting the UDP request message (ID # 1) from the Player, the Proxy then reads the next two bytes to determine the byte length of the UDP port value (always 2 bytes). The Proxy then reads two additional bytes to obtain the requested UDP port number. This port is PLPORT.



The Proxy scans the incoming data
from the Player and extracts PLPORT.

9. The Proxy allocates a new UDP port number to connect the Proxy to the Server. This is SVPORT.

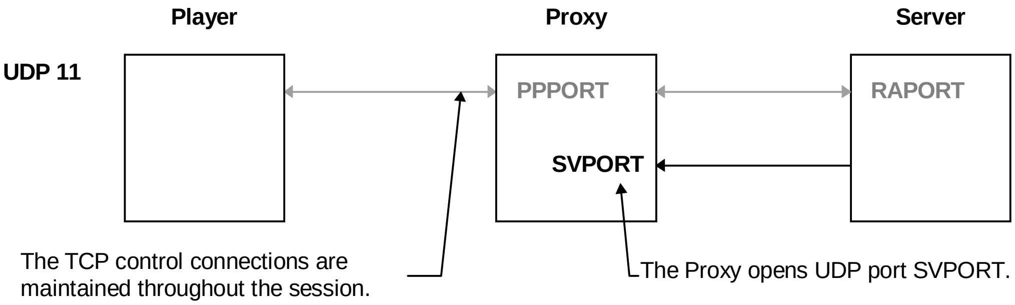
10. The Proxy substitutes the Port number (PLPORT) extracted in step 8 with the new port number (SVPORT) and passes the modified message to the RealAudio Server.



The Proxy sends port SVPORT
to the Server and saves PLPORT

Note The Proxy should allow every byte sent from the Player, other than the 2 byte UDP port number, to pass through unmodified.

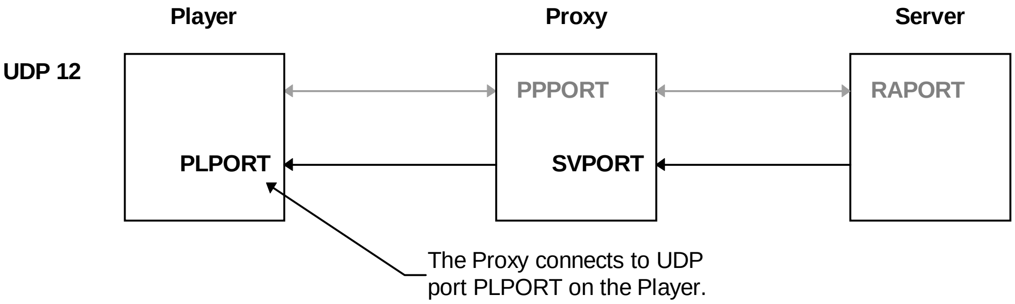
11. The Proxy opens UDP port SVPORT to the RealAudio Server.



The TCP control connections are maintained throughout the session.

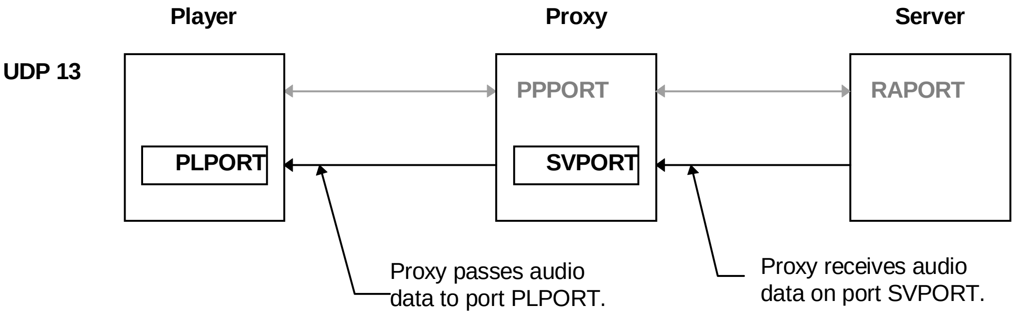
The Proxy opens UDP port SVPORT.

12. The Proxy connects to UDP port PLPORT on the Player.



The Proxy connects to UDP port PLPORT on the Player.

13. The Proxy passes all data received on UDP port SVPORT from the RealAudio Server to the Player on UDP port PLPORT.



Proxy passes audio data to port PLPORT.

Proxy receives audio data on port SVPORT.

14. If the Proxy detects Message ID 7 before Message ID 0, the Player is requesting a Robust UDP session. Continue with the Robust UDP steps. Otherwise, the data connection between the Player and the RealAudio Server using the Proxy is complete.

Note The Proxy should pass unmodified every byte sent through the UDP connection from the RealAudio Server to the Player.

15. When any of the connections are closed all other open connections in this Server - Proxy - Player interaction should also be closed. This terminates the Player - Proxy interaction and the Server - Proxy interaction.

Robust UDP Connection

The Robust UDP Connection steps are a superset of the Standard UDP Connection Steps. For a Robust UDP Connection, start with the steps in the previous section, and then continue with the steps in this section.

15. After detecting the Robust UDP request message (ID # 7) from the Player, the Proxy then reads the next two bytes to determine the byte length of the UDP port value (always 2 bytes). The Proxy then reads two additional bytes to obtain the UDP port number on the Player from which the Player sends UDP resend requests to the Server. This port is PRPORT.

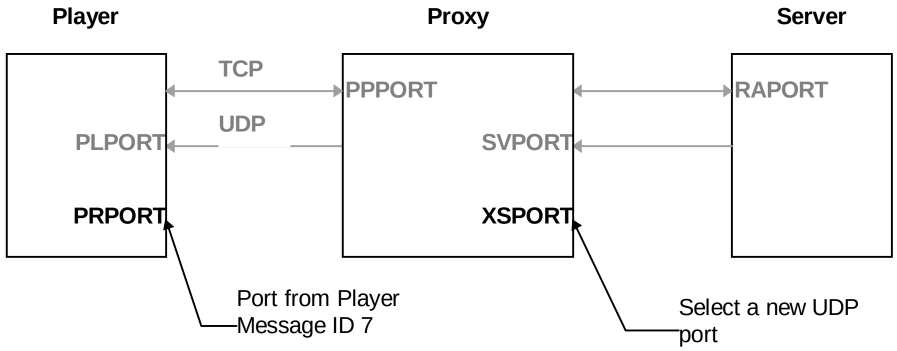
Robust UDP 15

Message ID 7	Message Length	UDP Port Number PRPORT
(0x0007)	(0x0002)	(2 byte integer)

Note Port PRPORT can optionally be used to validate requests; otherwise the PRPORT value is not needed by the Proxy. However, the Proxy does need to check for message ID 7 to know if Robust UDP is requested by the Player, and modify it if it is found.

16. The Proxy allocates a new UDP port number to connect the Proxy to the Server. This is XSPORT.

Robust UDP 16



17. The Proxy substitutes the port number (PRPORT) extracted in step 15 with the new port number (XSPORT) and passes the modified message to the RealAudio Server.

Robust UDP 17

Message ID 7	Message Length	New UDP Port Number XSPORT
(0x0007)	(0x0002)	(2 byte integer)

18. The Proxy scans the messages sent from the RealAudio Server to the Player on the TCP Control Connection. The Server first sends the Server Hello message.

    The Server Hello message is always 9 bytes long, starting with "PNA" (0x504e41).

Server Hello Message - 9 Bytes

Robust UDP 18

"PNA"	Protocol Version	Hello Data
(0x504e41)	(2 byte integer)	(4 bytes)

Immediately following the Server Hello message is Robust UDP response message from the Server.

Robust UDP Response - 10 Bytes

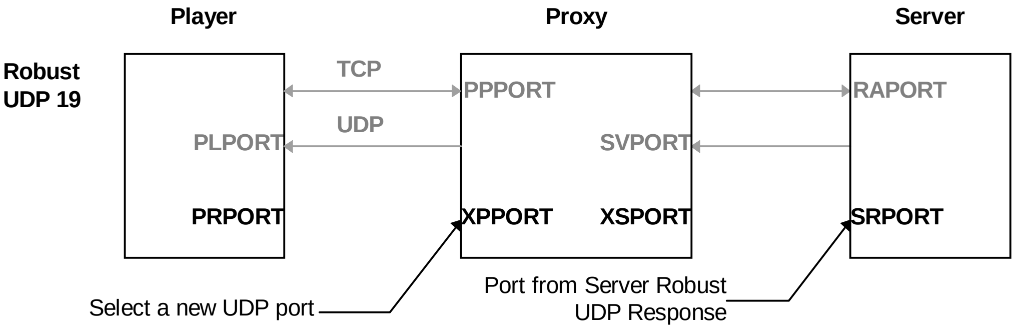
Robust UDP 18

"O"	Data Length	Opcode 7	Server Port SRPORT	Player Request ID
(0x4f)	(0x08)	(0x0007)	(2 byte integer)	(4 bytes)

The Robust UDP Response message from the Server contains the port number on the Server to which the Player sends UDP resend requests. This is port SRPORT. The Robust UDP Response message also contains a 4-byte identifier that the Player uses in UDP resend requests to identify the source of the request. This ID can optionally be used to validate resend requests from the Player.

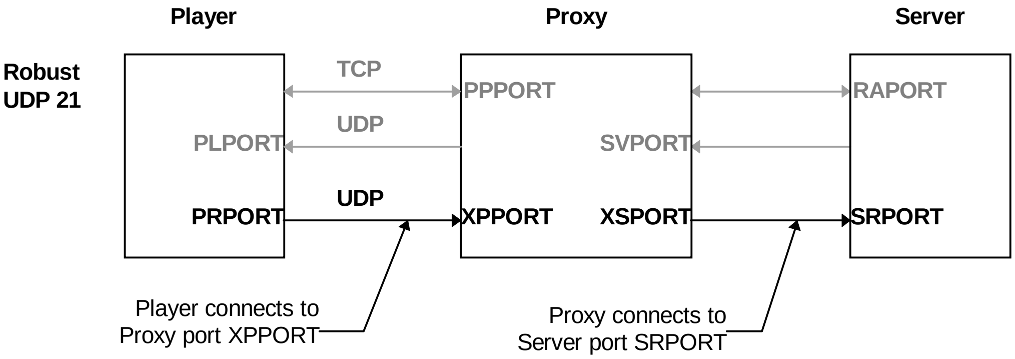
If the Proxy does not receive the Robust UDP response message from the Server immediately following the Server Hello message, the Server does not support Robust UDP. The Proxy should release the UDP port XSPORT and continue the session as a Standard UDP connection.

19. The Proxy allocates a new UDP port number to connect the Player to the Proxy. This is XPPORT.



20. The Proxy substitutes the Port number (SRPORT) extracted in step 18 with the new port number (XPPORT) and passes the modified Robust UDP Response message to the RealAudio Player on the TCP Control Connection.

21. The Proxy opens a UDP connection from XSPORT to SRPORT on the Server. The RealAudio Player opens a UDP connection from PRPORT to XPPORT on the Proxy.



22. The Proxy passes unmodified all bytes received from PRPORT on the Player to SRPORT on the Server.

23. When any of the connections are closed all other open connections in this Server - Proxy - Player interaction should also be closed. This terminates the Player - Proxy interaction and the Server - Proxy interaction.

Startup Message Definitions
Messages are encoded in a standard format. This allows unknown messages to be skipped over and ignored. The standard format is as follows:

Table 1 - RealAudio Proxy Protocol Message Format

← Data Flow

2 Bytes	2 Byte	n Bytes (value of n = Message Length)
Message Identifier	Message Length	Optional Data

All numeric fields are in Network byte order. Data streams in the direction of the arrow.

Table 2 - RealAudio Handshake Proxy Protocol

Opcode	Op Name	Data Length Bytes	Data Contents	Purpose
0	End	0 (implied)		Signals end of startup messages.
1	UDP	2	UDP port number	Specifies the Player UDP port. If this message is not sent, a TCP-Only session is used.
7	Robust UDP	2	UDP port number	Requests a Robust UDP session. Specifies the port from which the Player sends UDP resend requests.

The Proxy should ignore all other startup messages and pass them unmodified to the Server.

An example
The following are examples of startup messages. All numeric values are encoded in network byte order as integers.

The byte offset for the UDP port number will vary depending upon if other options are also being set. Following are two sample connect messages for when error correction is turned on and off. The default setting is to have error correction turned on. Please note that the Proxy should ignore all startup messages except those with identifiers of 0 (end messages) or 1 (UDP port request).

<u>The handshake byte sequence with error correction turned on is as follows:</u>

`PNA050102nn00`

Where
PNA is the three character identifier (3 byte string)
05 is the protocol version number (2 bytes)
01 is the identifier for the UDP port request (2 bytes)
02 is the byte length of the UDP request message (2 bytes)
nn is the actual UDP port number being requested (2 bytes)
00 is the identifier which signals the end of startup messages (2 bytes)

Note: There are no data length or data fields following the End Message identifier.

<u>When error correction is turned off the sequence becomes:</u>

PNA0502000102nn00

Where

PNA is the three character identifier (3 bytes)
05 is the protocol version number (2 bytes)
02 is the identifier to turn off error correction (2 bytes)
00 is the byte length of the error correction message (2 bytes)
01 is the identifier for the UDP port request (2 bytes)
02 is the byte length for UDP request message (2 bytes)
nn is the actual UDP port number being requested (2 bytes)
00 is the identifier which signals the end of startup messages (2 bytes)

Note: In this example the error correction message itself has zero byte length. The action of turning off error correction is inferred from the option being sent. The default setting (used when this option is not sent) is to use error correction.

The handshake byte sequence with error correction turned on and TCP-Only session:

PNA0500

Where

PNA is the three character identifier (3 byte string)
05 is the protocol version number (2 bytes)
00 is the identifier which signals the end of startup messages (2 bytes)

Pseudo Code of Player Proxy Interactions

The following section uses a C Style pseudo code to describe the operation of a Transparent Proxy during a firewall interaction. All descriptions of messages refer to structured RealAudio Proxy Protocol messages. These messages are defined in Table 2 - RealAudio Handshake Proxy Protocol.

```
//
//  The assumption is that this code is called with
//  the player already connected on a standard socket.
//
//
// OS/FW dependent call to determine server we
// really want to connect to.  Gives us server/addr and port we
// should connect to.
//



OSDependentCall(server, port);

connect to server/port;
if failure {
     close connections;
     exit;
}

SetupDialog();
if failure {
     close connections
```

```
    exit;
}

do {
    read tcp data from server => send data to player;
    read tcp data from player => send data to server;
    if ( use_tcp == 0 )
      read udp data from server => send udp data to player;

} while ( all tcp connections are open );

SetupDialog() {

    read first 3 bytes of data from player; // (startup_string)
    if ( startup_string != "PNA" ) {
      // log error?
      return error;
    }
    write first 3 bytes of data to server; // (startup_string)

    read 2 bytes from player; // (version)
    if ( version > 255 ) {
      // log error?
      return error;
    }
    write 2 bytes of data to server; // (version)

    // assume tcp only until we determine otherwise
    // case definitions:  UDPBACKPORT = 1 , END = 0

    use_tcp = 1;
    done = 0;
    do {
      read 2 bytes option_code from player;
      write 2 bytes option_code to server;

      switch (option_code)
      {
            case UDPBACKPORT:
                  read 2 bytes option_length from player;
                  write 2 bytes option_length to server;
                  read 2 byte backport value from player;
                  stash backport;
                  // if we got a backport, then we want udp
                  use_tcp = 0;
                  // setup udp backport connection
                  udpsetup();
                  break;
            case END:
                  done = 1;
                  break;
            case default:
                  read option_info from player;
                  write option_info to server;
                  break;

      }
    } while ( ! done );
}
```

```
udpsetup()
{
    open a udp port for listening to audio from server;
    write 2 byte "new" port number to server;
    setup/create info for sending udp info to player:backport;
}
```