Specifications for Building an *Application-Level* Firewall Proxy to Support RealAudio® Players

Introduction

This document provides information that enables firewall developers to support 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, Application-Level and Transparent.

Application-Level Proxy Firewalls
An Application-Level Proxy firewall relies on the application inside the firewall having knowledge of the firewall proxy. The Player inside the firewall connects to the specified Proxy, which then connects to the requested Server outside the firewall. 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.

This document describes how to build an Application-Level Proxy for RealAudio Player.

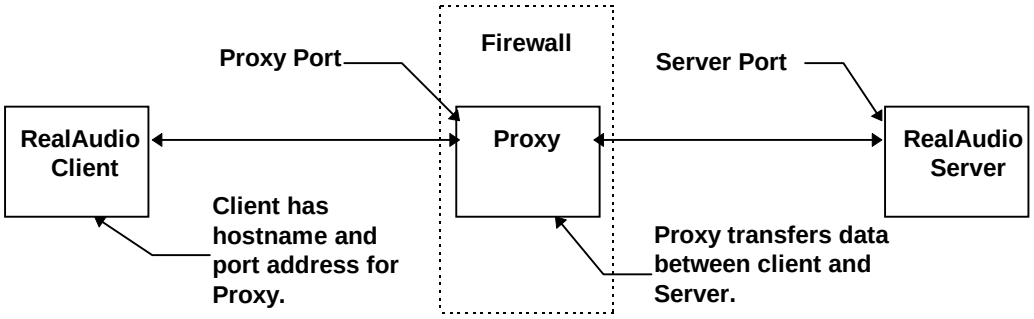


Figure 1 - Application-Level Proxy

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. For information on how to build a Transparent Proxy, please refer to *Specifications for Building a Transparent Firewall Proxy to Support RealAudio* which is available from Progressive Networks at:
www.realaudio.com/help

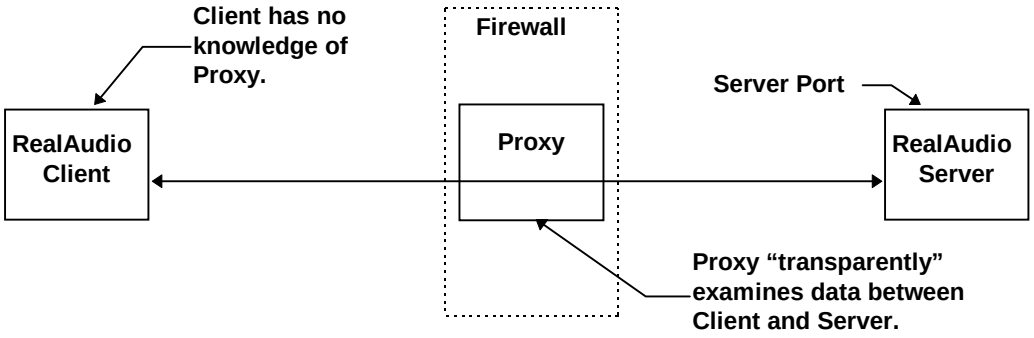


Figure 2 - Transparent Proxy

RealAudio 3.0 Support

RealAudio Player and Server 3.0 offer a new protocol called Robust UDP. Supporting Robust UDP requires additional steps beyond what is required to support other RealAudio protocols.

Although supporting Robust UDP is optional, it gives end users the best audio quality. To support Robust UDP, your Proxy must parse the hello message sent from the Player to the Server after the basic Proxy connection is established. See "Supporting Robust UDP" on page 9.

RealAudio Communications

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.

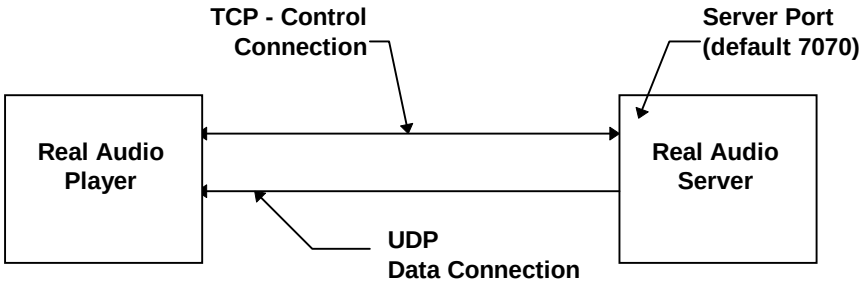


Figure 3 - RealAudio Client / Server Communications : Standard UDP Mode

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.

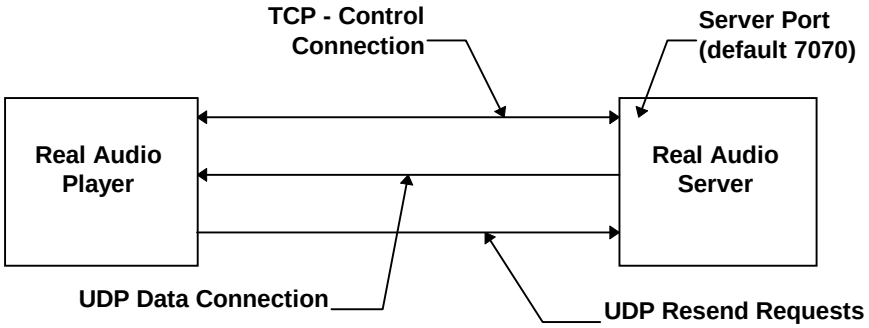


Figure 4 - RealAudio Client / Server Communications : Robust UDP Mode

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.

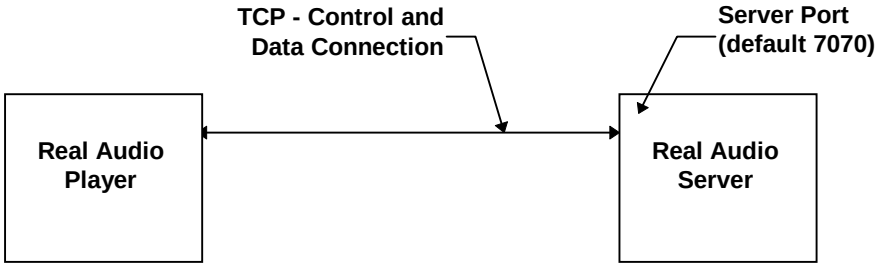


Figure 5 -RealAudio Client / Server Communications : TCP-Only Mode

Real Audio Communication and Firewalls

Application-Level Proxies are supported by RealAudio Player 2.0 and later and require the Player to be configured with the hostname and port number used by the Proxy. The Player connects to the Proxy and passes information to it which is used by the Proxy to connect to the RealAudio Server. Apart from an initial setup protocol between the Proxy and the Player, the dialogue is identical to that used between a Player and Server but all data flows through the Proxy, rather than directly to the Server.

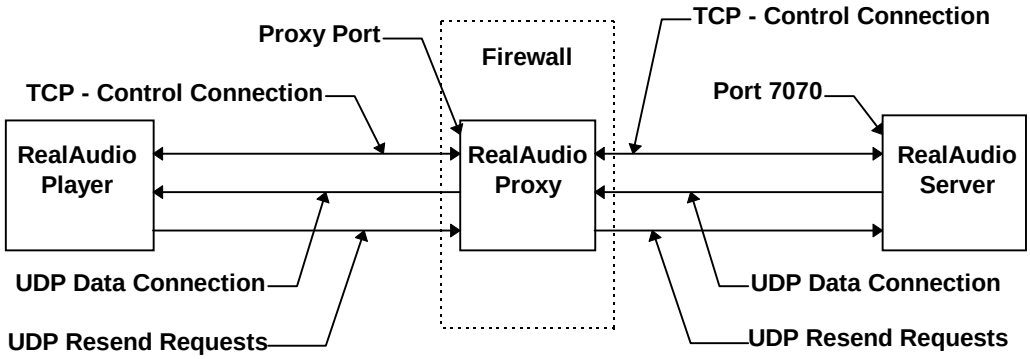


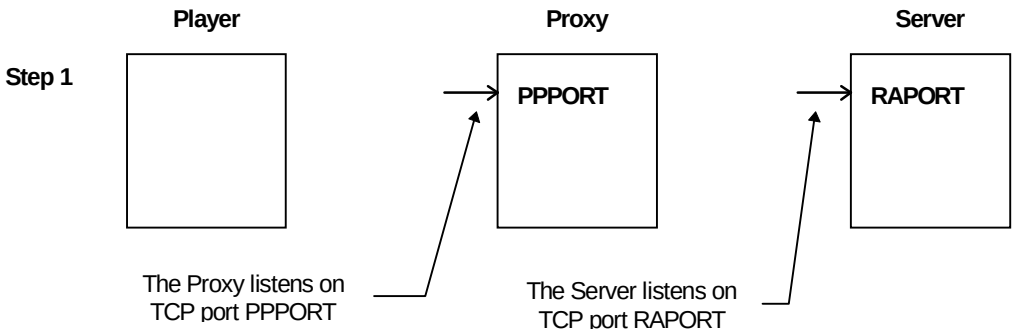
Figure 6 - RealAudio Application-Level Proxy Communications

Application-Level Proxy Handshaking

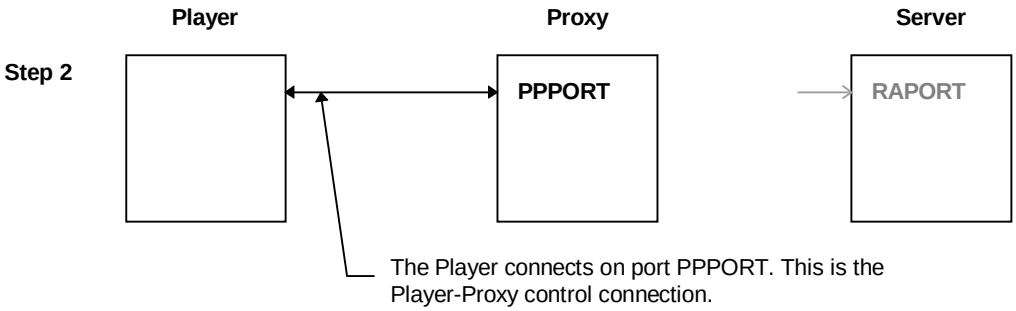
Application-Level Proxies are supported by RealAudio Player 2.0 and later. The interaction between the Player and the Proxy is described with diagrams that show the messages and actions. The connections from prior steps in the diagrams are grayed out when they are not an active part of a step. All descriptions of messages refer to structured RealAudio Proxy Protocol messages, defined in Table 2 - RealAudio Proxy Protocol, Version 1 on page 13.

Handshake and Communications Description

1. The Proxy listens on its defined TCP port, PPPORT (standard is port 7070). All Players within the firewall must be set up with the Proxy hostname and TCP port. The RealAudio Server is outside the firewall and is passively listening for incoming connections on TCP port RAPORT (standard is port 7070).

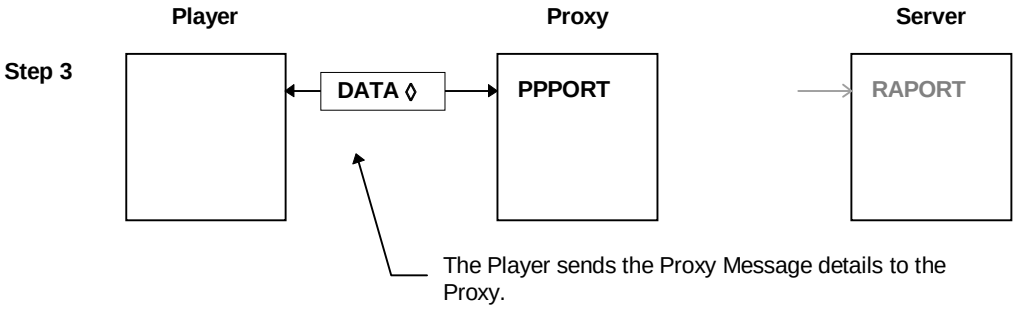


2. The Player connects using TCP to the Proxy on port PPPORT. This is a control connection between the Proxy and the Player.

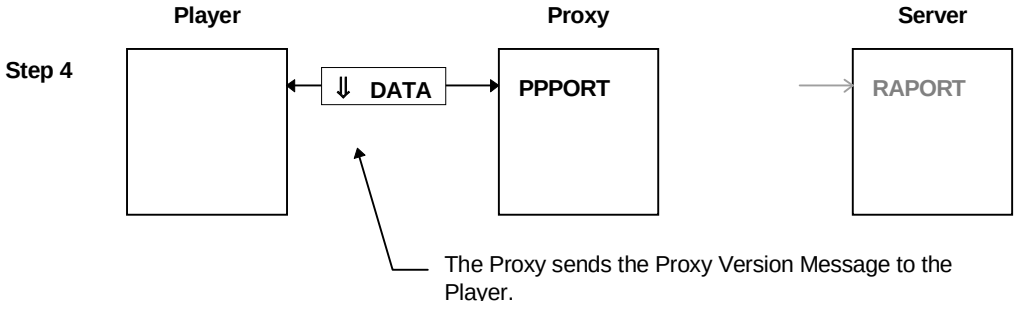


3. The Player sends a set of messages to the Proxy. The Proxy protocol version message must be the first message sent and the End proxy message must be the last. There is no specified order for other messages.

    a) Proxy protocol version
    b) Server hostname
    c) TCP port number for the Server, call this RAPORT
    d) Type of data connection, TCP-Only or UDP (the type of UDP, Standard or Robust, is determined later)
    e) Offset to the location of the UDP data port number in the subsequent communication between the Player and the Server. This offset is relative to the start of the normal Player - Server communication. (Not sent for TCP-Only mode.)
    f) End proxy



4. The Proxy responds to the Player with the Proxy Protocol Version message.

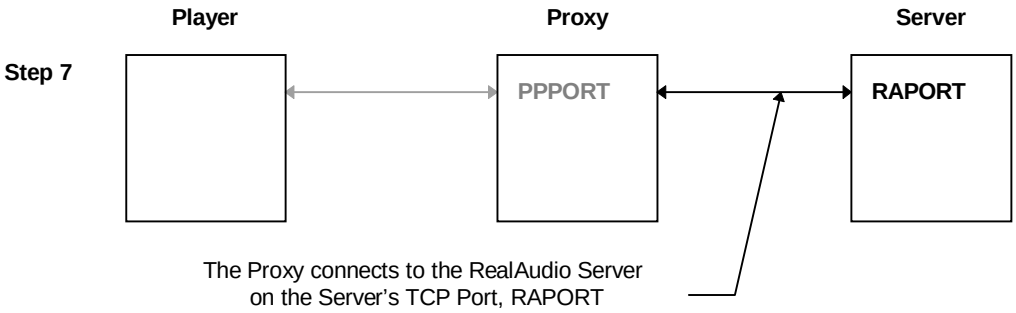


5. If the Proxy protocol version sent by the Proxy is not supported by the Player, the Player and the Proxy terminate the communication and the Player displays an error message.

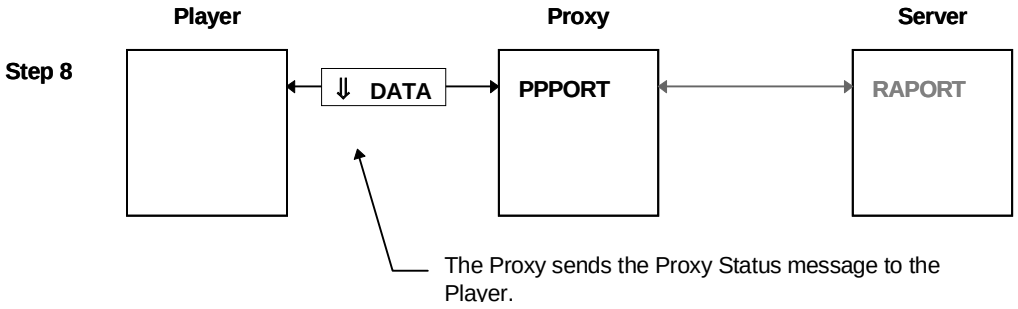
6. The Proxy reads in all incoming messages from the Player and collects and saves the following information:

    a) Proxy Protocol Version
    b) Server hostname
    c) TCP port number for the Server, call this RAPORT
    d) Type of data connection, TCP or UDP (the type of UDP, Standard or Robust, is determined later)
    e) Offset to UDP Port number
    f) Proxy End

7. Once all of the required information in step 6 has been received by the Proxy from the Player, the Proxy opens a TCP connection to the Server on TCP port RAPORT. The hostname and port for the RealAudio Server are those received in Step 6. This completes the TCP Control connection between the Player and the Server through the Proxy.



8. The Proxy sends a status message with a success code to the Player. If all required information is not received or if a error occurs in completing the connection to the server then a status message with an error code and message is sent to the Player.



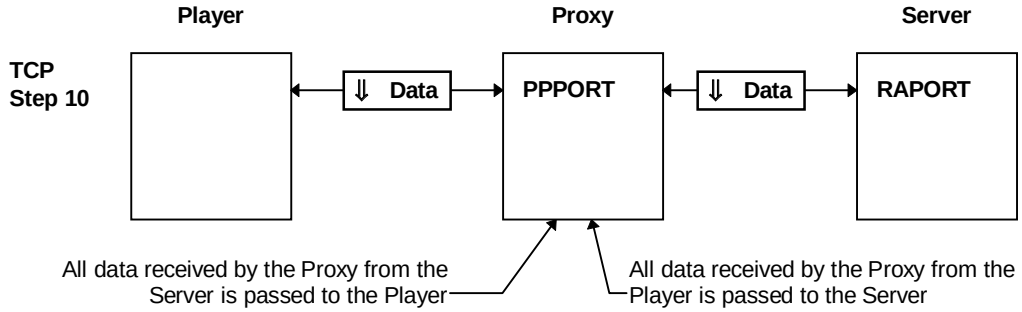
9. The Player examines the Status message code received from the Proxy and if it contains a Status of Success, it then begins it's normal Player / Server interaction. The existing TCP control connection is kept open and is used for the ongoing Player / Server communications.

    If the Status Message code received from the Proxy by the Player is an Error, the Player displays the corresponding error message and any error string supplied with the Status Message. The Player will then close the TCP control connection to the Proxy ending all interactions.

Depending on whether the data connection is UDP or TCP, complete one of the following sets of steps:

TCP-Only Connection

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

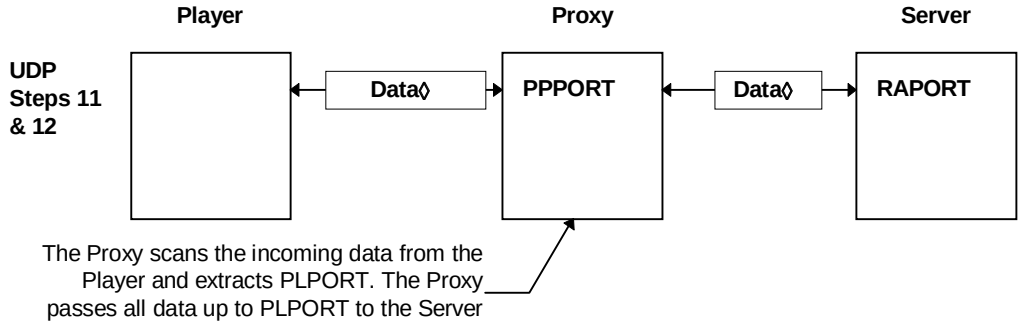


UDP Connection

10. 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 Server to the Player over the TCP control connection.

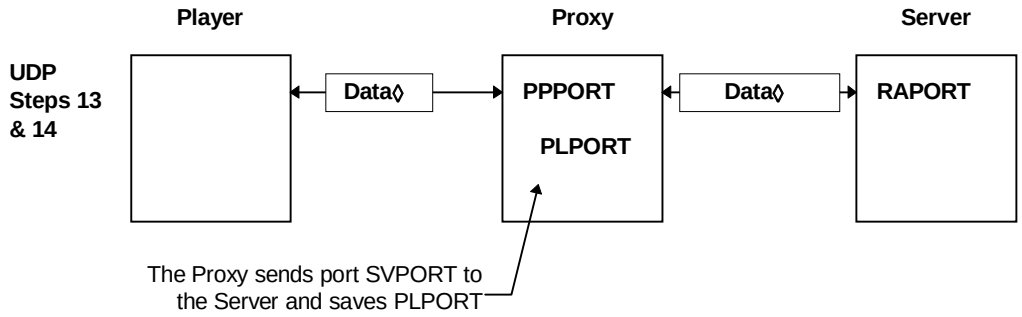
11. After receiving the End Proxy message from the Player, the Proxy begins counting the number of bytes of incoming data from the Player. This enables the Proxy to locate the requested UDP Port number in the data stream sent between the Player and the RealAudio Server. The port number is located after the number of bytes indicated in the UDP Offset Message (that is, if the offset is 4 bytes then the Port number is located in the 5th and 6th byte). The Port number is always two bytes long.

12. The Proxy extracts the Player Port number from the data stream. This Port is used by the Player to receive the incoming Audio Data from the Server. This port is PLPORT.

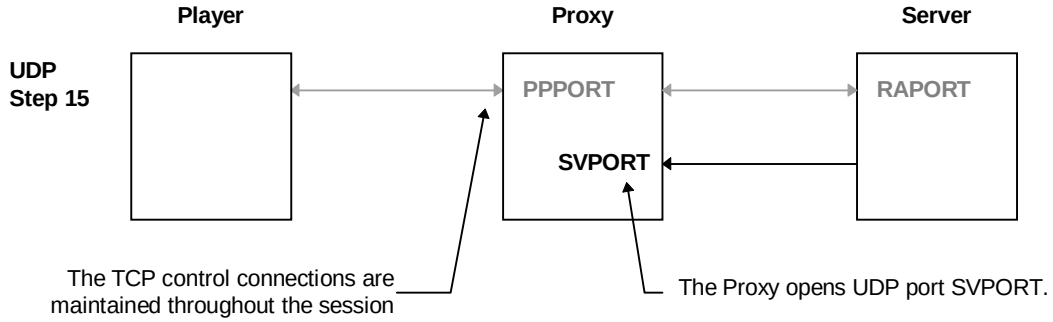


13. The Proxy allocates an unused port number to connect to the Server. This is SVPORT.

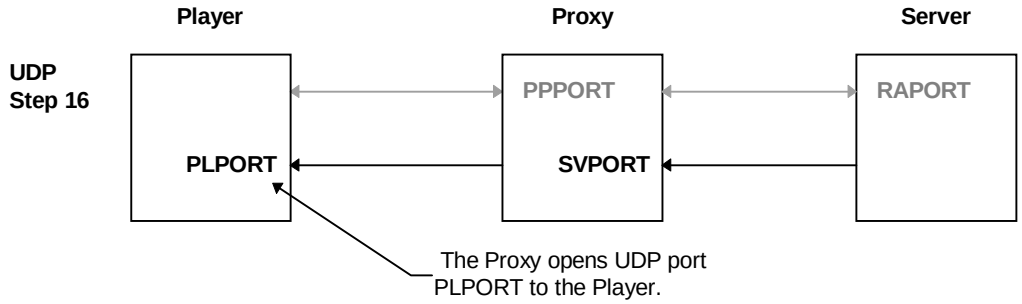
14. The Proxy replaces PLPORT from Step 12 with the new port number (SVPORT) and passes the data stream to the Server.



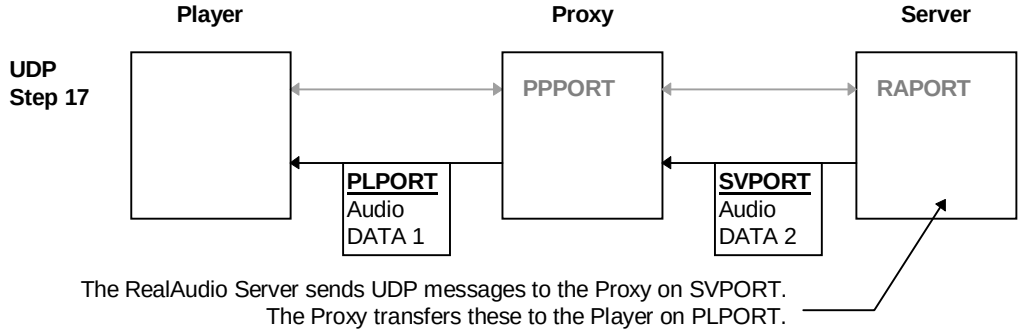
15. The Proxy opens UDP port SVPORT to the RealAudio Server.



16. The Proxy opens a UDP port (PLPORT) to the Player. This completes the basic data connection between the Player and the RealAudio Server through the Proxy.



17. All data received by the Proxy on UDP port SVPORT from the Server is forwarded to the Player on UDP port PLPORT.



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

Supporting Robust UDP

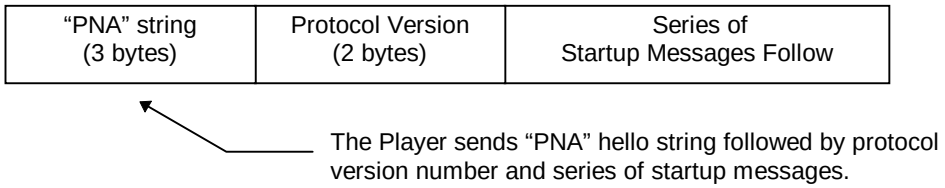
An application-level Proxy can optionally support Robust UDP communication. Robust UDP uses a second UDP connection from the Player to the Server to request resending lost packets. By supporting Robust UDP, your Proxy enables users to receive the best audio quality.

If the Player specifies a TCP-Only connection, the Proxy should skip these steps.

Once basic communication is established through the Proxy, the Proxy needs to parse the messages between the Player and Server to set up the UDP connection for resend requests as follows:

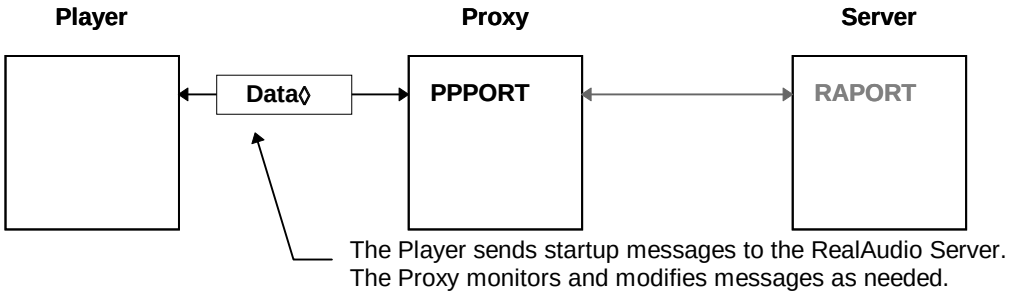
1. After the Player sends the Proxy message, it sends a Hello message to the RealAudio Server that starts with the string "PNA" (hex = 504e41) followed by the protocol version number (a two byte integer). All numeric values are encoded in network byte order.

Step 1



2. The Proxy must detect and modify specific startup messages being passed over the TCP control connection to determine if the connection is Standard UDP or Robust UDP.

Step 2



3. 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 3

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

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

   ID 7 = Robust UDP
   ID 0 = End of start up messages

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.

The Proxy needs to check for two possible message sequences:

Message ID 7 before Message ID 0: **Robust UDP Connection**
Message ID 0 without Message ID 7: **Standard UDP Connection**

Note The Player may send other startup messages before Message ID 0; all messages except Message ID 7 should be passed through unmodified to the Server.

If the connection type is Standard UDP, the Proxy continues passing data unmodified between the Player and Server. If the connection type is Robust UDP, the Proxy completes the following steps:

4. 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.

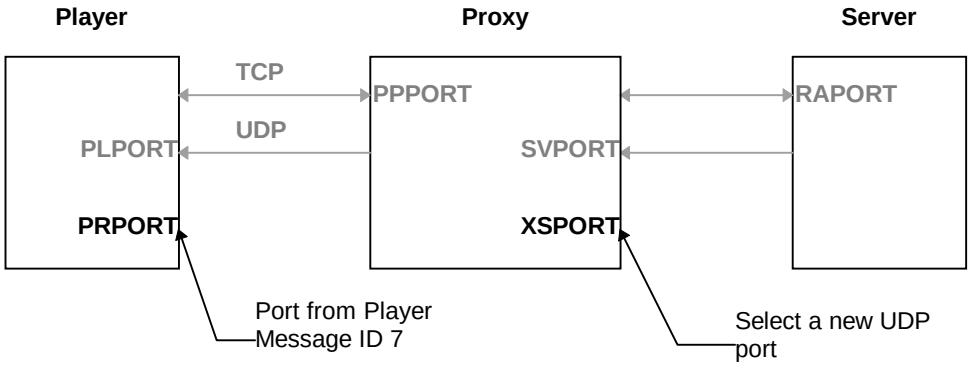
Step 4

Message ID 7 (0x0007)	Message Length (0x0002)	UDP Port Number PRPORT (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.

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

Step 5



6. The Proxy substitutes the port number (PRPORT) extracted in Step 4 with the new port number (XSPORT) and passes the modified message to the RealAudio Server.

Step 6

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

7. 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).

Step 7

Server Hello Message - 9 Bytes

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

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

Step 8

Robust UDP Response - 10 Bytes

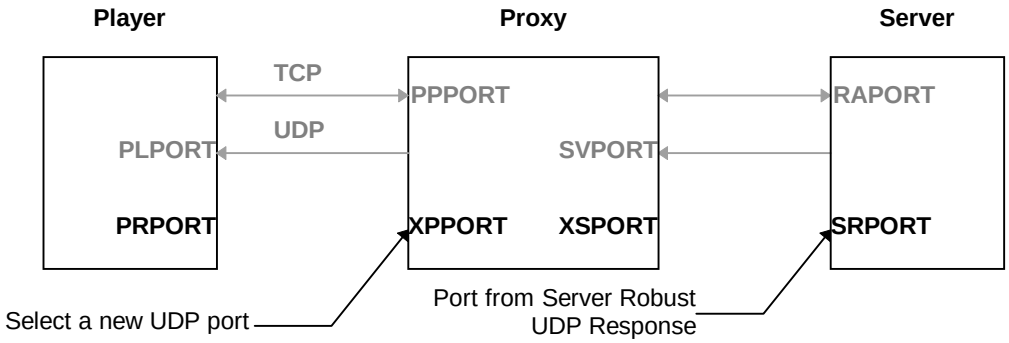
"O" (0x4f)	Data Length (0x08)	Opcode 7 (0x0007)	Server Port SRPORT (2 byte integer)	Player Request ID (4 bytes)

   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.

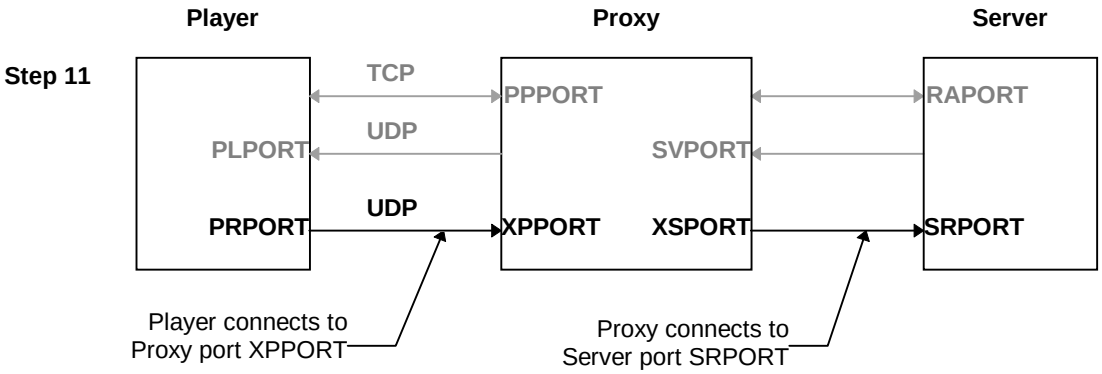
9. The Proxy allocates a new UDP port number to connect the Player to the Proxy. This is XPPORT.

Step 9



10. The Proxy substitutes the Port number (SRPORT) extracted in step 8 with the new port number (XPPORT) and passes the modified Robust UDP Response message to the RealAudio Player on the TCP Control Connection.

11. 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.



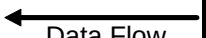
12. The Proxy passes unmodified all bytes received from PRPORT on the Player to SRPORT on the Server.

13. 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.

Message Definitions for Proxy Server Interaction

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



2 Bytes	1 Byte	n Bytes
Length of opcode & data portion of message	Opcode, defines message type	Optional Data

All numeric fields are in Network byte order

Table 2 - RealAudio Proxy Protocol, Version 1

Opcode	Op Name	Data Length Bytes	Data Contents	Purpose
0	end	0 (implied)		Signals end of Player / Proxy communication.
1	version	1	1	Specifies the version of the Proxy protocol
2	hostname	string length	host name as ascii string	Specifies the RealAudio server hostname. This can be an ascii IP address
3	port	2	port number	Specifies the port number for the RealAudio Server.
4	port_off	2	port offset	Specifies the number of bytes to the location of the Port number in the incoming data stream once Player / Proxy communications have completed.
5	use_tcp	1	0 or 1	Specifies whether the data connection will be TCP or UDP.
6	status	1+string length	error code + ascii string	Provides a status on operations. This includes a 1 byte code plus a optional ascii string.

The following table lists the current set of status values defined in the protocol. If a player receives a status that is not included in this table it will display the supplied string and terminate the Proxy interaction.

Table 3 - RealAudio Proxy Protocol Status codes

Status Number	Status Description
0	Success
1	No Connect, could not connect to RealAudio Server
2	Bad Info, Bad information was passed in the Proxy dialog
3	Bad Version, the RealAudio Proxy Protocol version number does not match
4	Bad Opcode, a message with an invalid opcode was received

The following table details an example of each message. All numeric values are encoded in network byte order as integers All strings are in ascii and are not nul ('\0') terminated. The middle three columns of the table can be taken as an example message layout Where the data is listed as None this signifies that no data is sent with this message.

Table 4 - Sample RealAudio Proxy Protocol Messages

Op Name	Length 2 Bytes	Opcode 1 Bytes	Data	Notes
end	1	0	*None*	
version	2	1	1	network byte order
hostname	18	2	www.realaudio.com	ascii
port	3	3	7070	network byte order
port_off	3	4	15	network byte order
use_tcp	2	5	0	network byte order
status	36	6	2Permission Denied, Invalid Username	1 byte number followed by ascii string

Pseudo Code of Player Proxy Interactions

The following section uses a C Style pseudo code to describe the operations in both the Player and the Proxy during a Firewall interaction. All descriptions of messages refer to structured RealAudio Proxy Protocol messages. These messages are defined in Table 2 - RealAudio Proxy Protocol, Version 1

Player Pseudo Code

```
Player-side: {
  player_side_dialog;
  if success
    normal server_player stuff
  else
    error processing
}
```

Proxy Pseudo Code

```
Proxy-side: {
   proxy_side_dialog;
   if success {
     if (use_tcp == 0 {
       open a udp port

       while (off < port_off) {
         copy every byte from the player to
             the server
       }
       read two bytes.
       send the port number of the socket we
         just opened.
     }
     do {
       read from player => send to the server
       read byte server => send to the player
       if (use_tcp == 0) {
         read audio packet from server => send
           to player
       }
     } while (all tcp connections are open);
     close all tcp connections;
   } else
     close the connection;
}
```

```
player_side_dialog: {
  send version message
  send hostname message
  send port message
  send port_off message
  send use_tcp message
  send end message
  await message
  if (message != version or an
```

```
proxy-side-dialog: {
   send version message;
   await message;
   if (message != version or unknown version) {
     send status = bad first message or
                unknown version
     return bad_player;
   }
```



```
        unknown version)
      return bad_proxy;

    await message
    switch (message) {
      case status:
          if (status == error)
            return status;
          done = 1;

      default:
          return invalid message

    } while (!done);

    send end
    return success;
  }
```

```
    done = 0;
    do {
      await message;
      switch (message) {
        case hostname: stash hostname;
        case port: stash port;
        case port_off: stash port_off;
        case use_tcp: stash use_tcp;
        case end: done = 1;
      }
      if (all required information received)
        done = 1;
    } while (!done);

    connect to the server's host:port

    if (connection OK)
      send status = success;
    else
      send status = failure;

    return success;
  }
```