

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

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.

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.

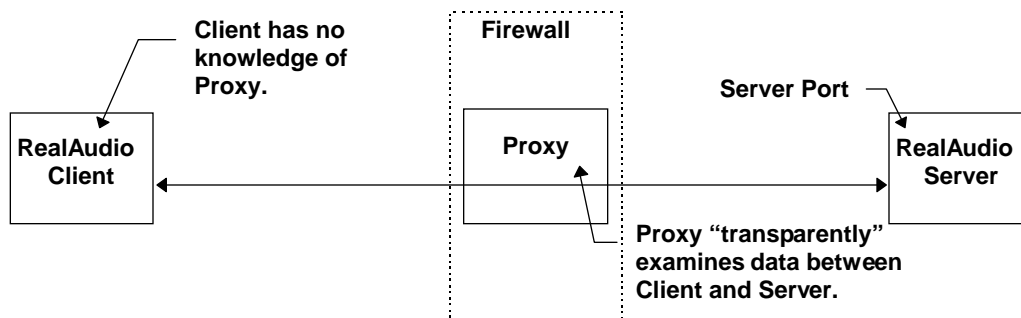


Figure 1 - Transparent Firewall

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*.

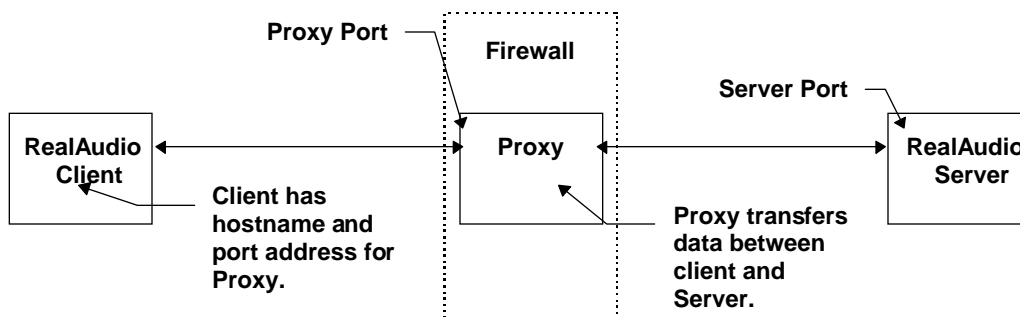


Figure 2 - Application-Level Proxy

RealAudio Communications

RealAudio Client / Server Communications

A RealAudio Client (Player) sets up either one or two network connections with the RealAudio Server. In the standard configuration, utilizing UDP, two network paths are used: a full-duplex TCP connection, used for control and negotiation, and a simplex UDP path from the RealAudio Server to the Player for audio data delivery. The UDP path is used for the audio to allow the RealAudio Server and Player to handle error correction rather than the protocol. This allows a RealAudio Server to provide a better Audio stream when packet loss occurs. 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. The standard TCP connection port on a RealAudio Server is 7070. The following diagrams show both these configurations.

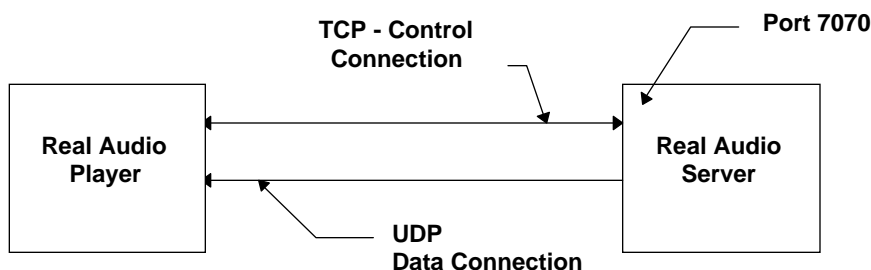


Figure 3 - RealAudio Client / Server Communications : Normal Mode (UDP)

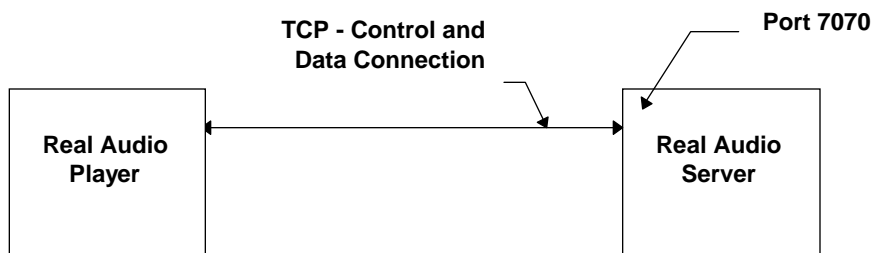


Figure 4 -RealAudio Client / Server Communications : TCP-Only Mode

Real Audio and Firewalls

A Transparent Proxy requires no explicit support in either the Player or RealAudio Server. It must know about the protocol being used between the Player and RealAudio Server but there is no actual connection made to the proxy by either the Player or RealAudio Server. In order for a Transparent proxy to function properly it must be able to detect and modify port requests being sent between the Player and the RealAudio Server. Working at the packet level, the Proxy must be able to capture, scan and modify TCP control messages. Unlike Application-Level Proxies, Transparent Proxies do not require the end user to configure the Player. Transparent Proxies can support both RealAudio 1.0 and RealAudio 2.0 Players.

The Application-Level Proxy can only support by RealAudio 2.0 Players and requires the Player to have been configured with the hostname and port number used by the Application-Level Proxy. In this case the Player always connects to the Proxy and passes information to it which can then be used by the Proxy to find the RealAudio Server, and connect to it. Apart from an initial setup protocol between the Proxy and the Player, the dialogue is identical to that used between a Player and a RealAudio Server but all data flows via the Proxy, rather than directly to the RealAudio Server. The details of producing an Application-Level proxies are covered in a separate document.

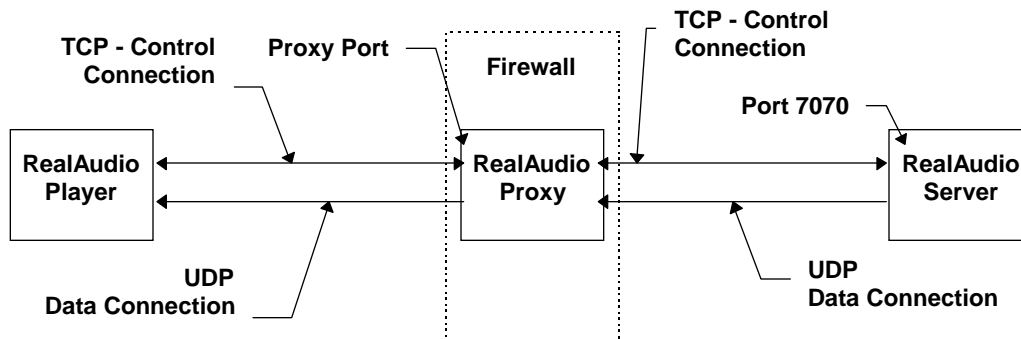


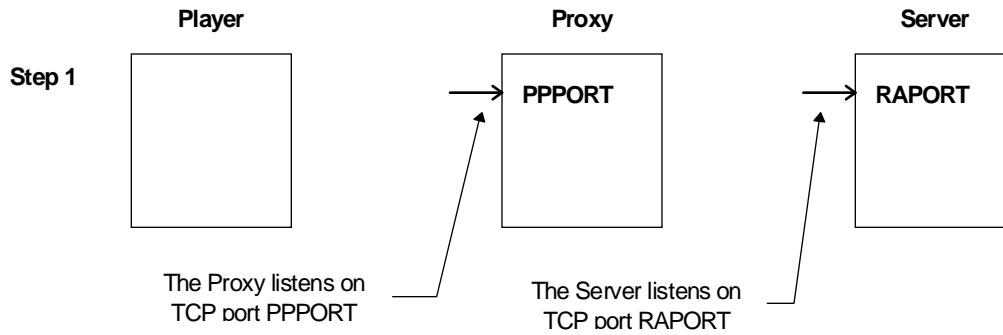
Figure 5 - RealAudio Proxy Communications

Transparent Proxy Handshaking

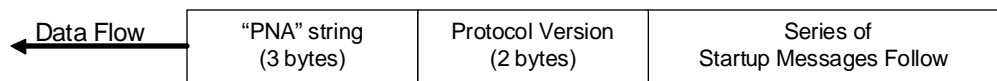
Transparent Proxies, if built to the following specifications, will support both RealAudio 1.0 and 2.0 Players. The interactions between the Player and the RealAudio Server are described in general terms below including schematic diagrams that show the progression of messages and actions in the interaction. 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.



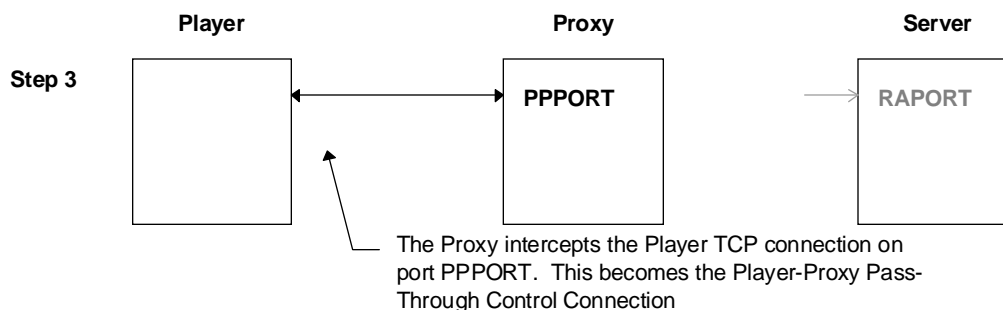
- When the Player attempts an active TCP connect to the RealAudio Server by sending the string "PNA" (hex = 504e41) followed by a two byte integer (in network byte order) that contains the protocol version number. All numeric values are encoded in network byte order. *Please note that firewalls built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



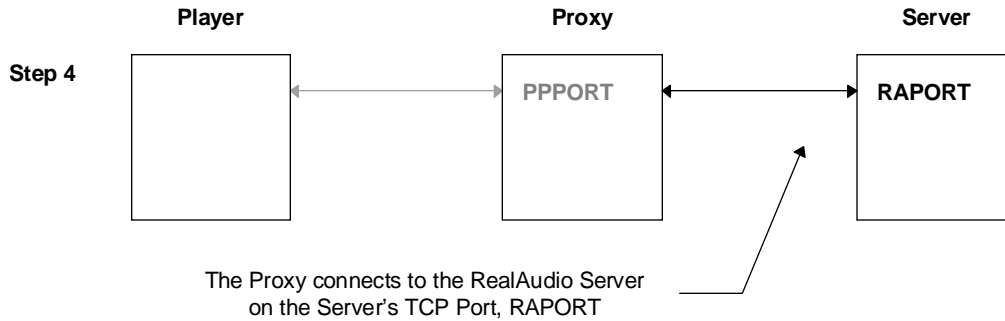
- Step 2**
- The Player sends "PNA" hello string followed by protocol version number and series of startup messages.

IMPORTANT NOTE: This document supports all protocols with version numbers less than 256.

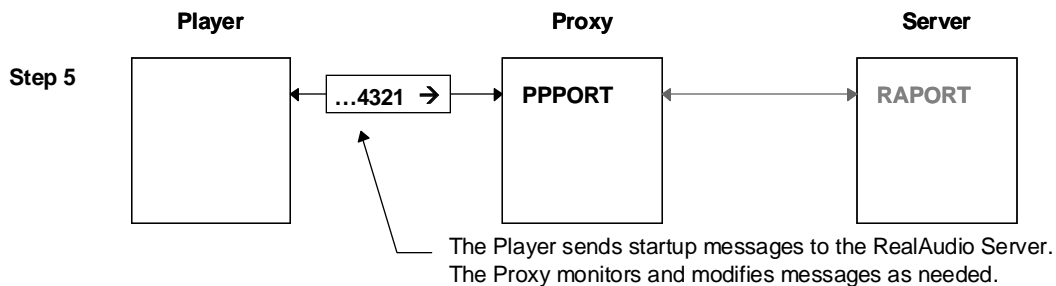
- The listening Proxy identifies this action as a RealAudio request and responds accordingly. The Proxy converts this connection into a pass-through control connection between the Proxy and the Player.



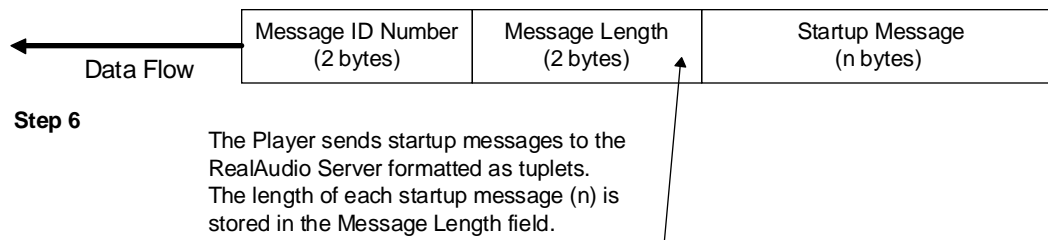
- The Proxy then opens a TCP connection to the RealAudio Server on TCP port RAPORT, using the IP addressing present in the TCP packets coming from the Player. This completes the TCP pass-through control connection between the Player and the RealAudio Server, via the Transparent Proxy.



5. In order to function properly, the Proxy must be able to detect and modify a specific startup message being passed over the pass-through control connection. In particular, the Proxy must determine the type of connection (TCP-Only or TCP & UDP) requested and respond accordingly.



6. Startup messages are formatted as tuples designating: 1) a message identifier (2 byte integer), 2) the byte length (n) of the message (2 byte integer) 3) the message itself (n bytes long). All numeric values are encoded in network byte order as integers



The only startup messages sent by the Player that the Proxy needs to check for are:

ID# 1 = UDP port request (if not sent, TCP-Only session is established)

ID# 0 =End start up messages (always sent, signals end of start up messages)

All other startup messages should be ignored by the proxy and be allowed to pass through to the RealAudio Server unmodified.

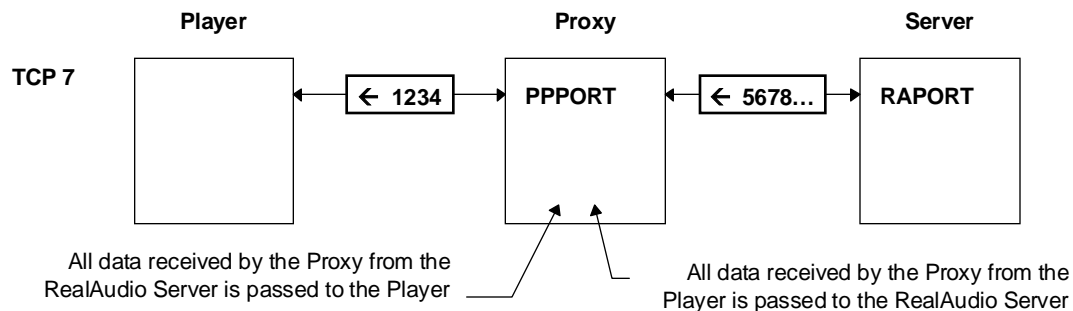
If the proxy detects message ID #1 in the TCP stream it means that the Player is requesting UDP audio delivery. It also means that the message value contains the UDP port number that the Player

expects to receive the data stream. If the proxy detects message ID# 0 in the TCP stream it means that all startup messages have been sent and that play time communications (i.e.: stop, fast-forward, random seek) between the Player and the RealAudio Server will now commence. If the Proxy detects message ID# 0 without first detecting ID# 1, it means that the Player is requesting audio to be delivered over the TCP pass-through control channel.

Depending on whether the audio data connection is **UDP** or **TCP**, follow one of the set of steps below

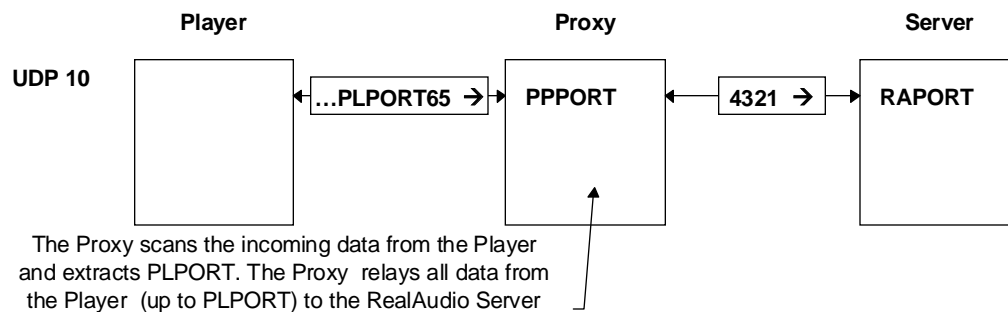
TCP-Only Connection

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 reverse is also true, all data received from the RealAudio Server by the Proxy is transferred to the Player.

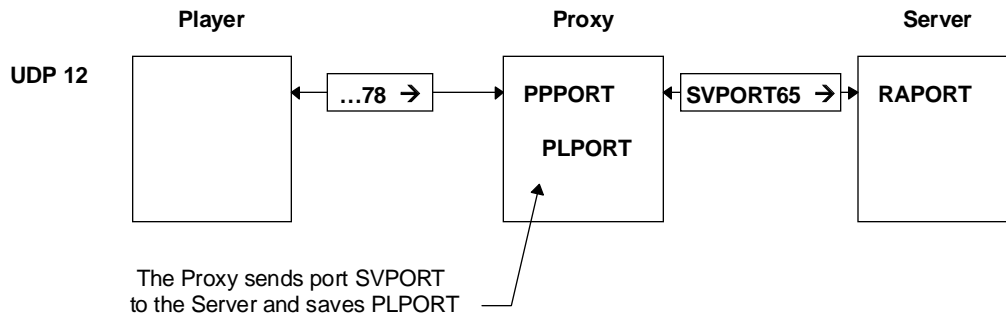


UDP Data Connection

8. The Proxy transfers every byte read from the Player on the TCP control connection to the RealAudio Server on its TCP control connection. The reverse is also true, all data received from the RealAudio Server by the Proxy is also transferred to the Player.
9. 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 (this byte length value should always be equal to 2). The Proxy then reads two additional bytes to obtain the requested UDP port number. The port number is always two bytes long.
10. 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

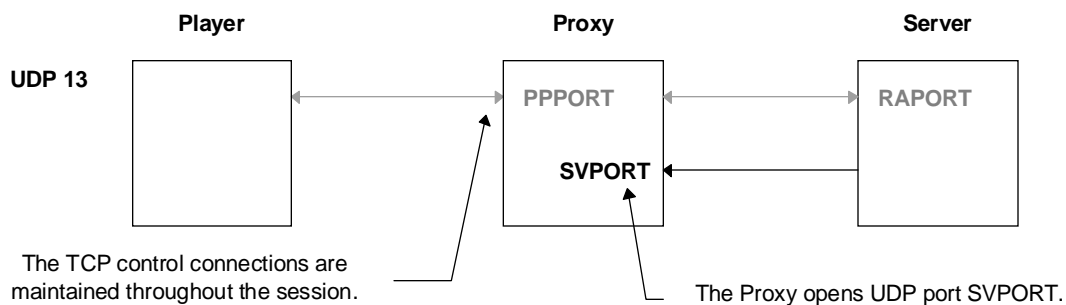


11. The Proxy then allocates a fresh port number to be used in making a connection between the Proxy and the RealAudio Server. This is SVPORT.
12. The Port number (PLPORT) located in step 7 is substituted in the data stream by the new port number (SVPORT) and the data stream is then passed onto the RealAudio Server.

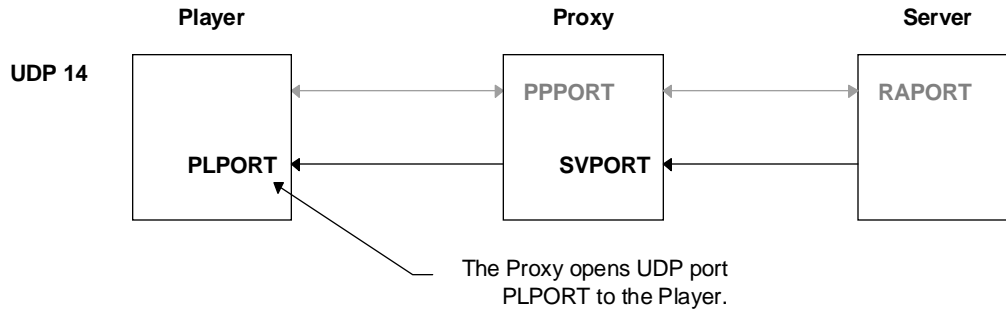


IMPORTANT NOTE: The Proxy should allow every byte sent from the Player, other than the 2 byte UDP port number, to pass through unmodified.

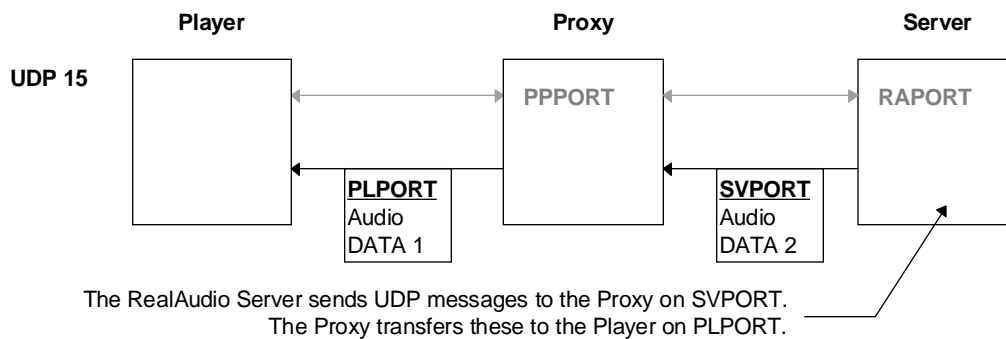
13. The Proxy opens UDP port SVPORT to the RealAudio Server.



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



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



IMPORTANT NOTES

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

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

16. When any of the connections are closed all other open connections in this Server / Proxy / Player interaction should also be closed. This will terminate the Player / Proxy interaction and also 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 setup. |
| 2 | Error Correction | 0 | | Sending this message turns off error correction. Proxies should ignore this message. It is included here only for discussion purposes (see examples below). |

IMPORTANT NOTE: The Proxy should ignore all messages with identifiers other than 0 (end messages) and 1 (UDP requested).

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).

The handshake byte sequence with error correction turned on is as follows:

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.

When error correction is turned off the sequence becomes:

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)

| |
|---|
| IMPORTANT NOTE: This document supports all protocols with version numbers less than 256. |
|---|

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 = 0;
        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;
}

```