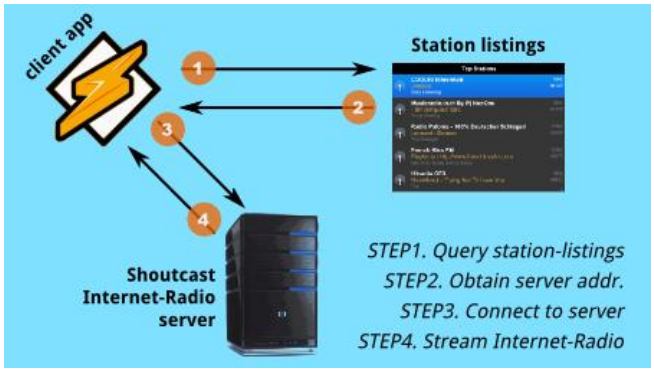
Internet Radio : Part1 - Shoutcast Protocol

>> C , CodeProject , curl , internet-radio , Programming , Shoutcast , streaming

NOTE: If you spent the last decade 100miles below the surface of the earth studying the growth of algae under an antarctic glacier, then you will be surprised to learn that we can now listen to radio over the internet. Just like tuning-in to particular frequencies for particular stations in the olden days; now we can "tune-in" to specific radio stations over the internet. Without further ado, lets jump-in into the world of "Internet Radio".

Most apps claiming to support Internet Radio, in fact support a industry standard - **Shoutcast**. It was a protocol devised by nullsoft in the 90's and first implemented in their popular player **Winamp**(stop reading if you haven't heard of Winamp!). Inspite of being a proprietary protocol with not much documentation to go with, Shoutcast has become the de-facto industry standard for streaming audio. This is mainly due to its simplicity and similarity with the existing hyper-text transfer protocol (dear old http! wink wink). **Icecast** is a similar open-source implementation compatible with Shoutcast.

INITIAL HANDSHAKE BETWEEN SHOUTCAST CLIENT-SERVER



High-level overview of Internet-radio over Shoutcast.

[STEP1] Station Listing

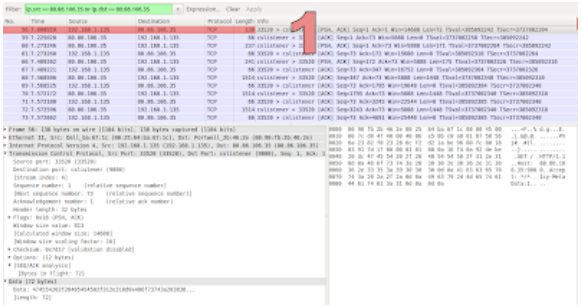
The client app connects to a station listing/aggregator on the internet and obtains a list of stations alongwith their details like genres, language, now-playing, bitrate among other things.

[STEP2] Station Lookup

The user can then select one of the stations as desired. Then the client obtains the ip-address(& port) of the server running that particular station from the station-listing/aggregator. Networking enthusiasts will notice that this step is exactly like a DNS lookup i.e. the client obtains the network address for a particular station name; the station-listing/aggregator acting like a DNS-server for Radio stations. Also note that sometimes the station-listing will provide only a domain-name and then an additional actual DNS lookup is needed to obtain the ip-address of the streaming server. Popular station-listing/aggregator sites like **Xiph**, **Shoutcast.com** and **vTuner** provide huge web-friendly lists of live radio stations.

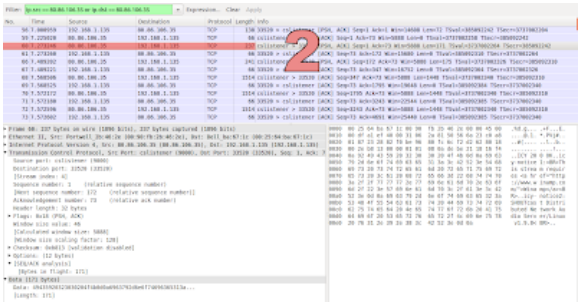
[STEP3|4] Station Connection

1. The client attempts to connect to the server using the ip-address(and port) obtained during station lookup.



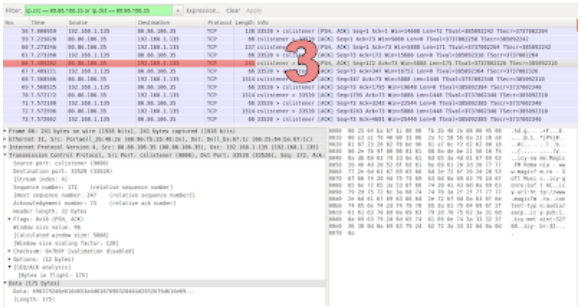
Connection request from shoutcast client (click to enlarge)

2. The server responds with "ICY 200 OK" (a custom 200 OK success code)...



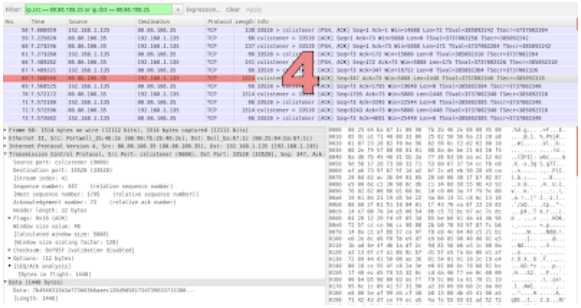
ICY 200 OK reply from shoutcast server (click to enlarge)

3. ...and the stream header...



Shoutcast stream header (click to enlarge)

4. ..and finally the server starts sending encoded audio in a continuous stream of packets(which the client app can decode and playback) until the client disconnects(stops ACK-ing and signals a disconnect).

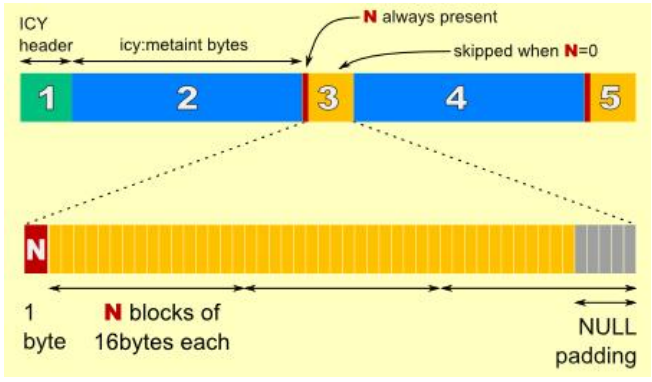


Encoded audio data stream (click to enlarge)

Download the entire WireShark capture of packets exchanged by the shoutcast client and server during initial station connection.

The above steps are similar to what a browser does when it connects to a website (and hence in-browser streaming audio playback of shoutcast streams IS possible).

Shoutcast has subtle differences over http during the **station connection** step above. Shoutcast supports "**Icy-MetaData**" - an additonal field in the request header. When set, its a request to the shoutcast server to embed metadata about the stream at periodic intervals(once every "**icy-metaint**" bytes) in the encoded audio stream itself. The value of "**icy-metaint**" is decided by the shoutcast server configuration and is sent to the client as part of the initial reply.



Shoutcast stream format when ICY:MetaData is set to 1

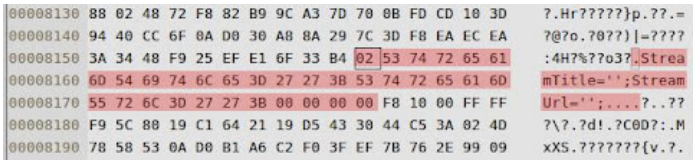
This poses a slight complication during playback. If the received audio stream is directly queued for playback, then the embedded metadata appears as periodic glitches. Following is one such sample recording. This audio clip was retrieved from a radio stream whose icy:metaint = 32768; i.e. the metadata is embedded in the audio stream once every 32KBytes. Stream bit-rate is 4KBps. **So during playback a glitch is present once every 32KB/4KB = 8seconds (0:08s, 0:16s, 0:24s, 0:32s,...).**

Oops, we couldr

~~To view/analyse the stream data in a hex editor, download the actual clip and check out the following offsets~~

> **Update:** Unfortunately the service i was using to host the audio clip has lost it and i was foolish enough to trust them and have no local backups. :(
> R.I.P
> Shoutcast-Metadata.mp3
> 2013-2014
> Here lies a song, cut short in its prime...

[0:08s] 0x0815A - 0x0817A count N = 2, meta = 1+ (16 x 2) = 33(0x21h)bytes
[0:16s] 0x1017B - 0x1017B count N = 0, meta = 1byte
[0:24s] 0x1817C - 0x1817C count N = 0, meta = 1byte
[0:32s] 0x2017D - 0x2017D count N = 0, meta = 1byte



Embedded metadata from 0x0815A to 0x0817a.
Note the first byte is 02 i.e. metadata is 2x16=32(0x20h)bytes following it.

Also note that the first 345(0x159h)bytes of the clip are the reply header of the stream(plain-text in ASCII) sent by the shoutcast server. Technically these are NOT part of the audio stream as well.

NOTE: If you simply want to obtain the audio stream (no embedded metadata) then set the "**Icy-MetaData" field in the request header to 0** or simply do NOT pass it as part of the initial request header.

Finally here is a small bit of code that implements all that we have learnt so far - a simple shoutcast client in a few lines of C, that connects to any shoutcast server and logs the audio stream data to stdout. It uses the curl library to initiate connection requests to the shoutcast server.

```c
/* listencast.c
 * A simple shoutcast client that
 * - Connects to any shoutcast server and
 * - Logs the audio stream data to stdout
 *
 * Date 12-02-2013
 *
 * Requires libcurl
 * $> sudo apt-get install libcurl4-gnutls-dev
 *
 * Building listencast.c
 * $> gcc listencast.c -o listencast -lcurl
 *
 * Running listencast
 * $> ./listencast <ip-addr>:<port> > <test.file>
 * Attempts to connect to the shoutcast server
 * running at <ip-addr> on port <port>. If successful
 * then writes the audio stream into the file <test.file>
 */

#include <curl/curl.h>

int main(int argc, char* argv[])
{
        CURL *curl;
        CURLcode res;

        /* URL of the radio-station in <ip-addr>:<port> format */
        char * webaddr = argv[1];
```

```
30
31          /* init to NULL is important */
32          struct curl_slist *headers=NULL;
33
34          /* Comment the following line to request a pure
35           * audio stream WITHOUT any periodic channel-info or
36           * now-playing info embedded in the stream.
37           */
38          headers = curl_slist_append(headers, "Icy-MetaData:1");
39
40          /* Obtain a curl object handle */
41          curl = curl_easy_init();
42
43          if(curl) {
44                  curl_easy_setopt(curl, CURLOPT_URL, webaddr);
45
46                  /* pass our list of custom made headers */
47                  curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
48
49                  /* Perform the request, res will get the return code
50                  res = curl_easy_perform(curl);
51
52                  /* Ideally code following this line is NEVER execute
53                   * as the radio station will reply with a continuous
54                   * never-ending infinite encoded audio stream.
55                   */
56
57                  /* Check for errors */
58                  if(res != CURLE_OK)
59                          fprintf(stderr,
60                                  "curl_easy_perform() failed: %s\n",
61                                  curl_easy_strerror(res));
62
63                  /* free the header list */
64                  curl_slist_free_all(headers);
65
66                  /* always cleanup */
67                  curl_easy_cleanup(curl);
68          }
69
70          return 0;
71  }
72
```

listencast.c hosted with ❤ by **GitHub** **view raw**

https://gist.github.com/TheCodeArtist/2f1b9fa68197e39ca9bc

Stripping off the comments and the clean-up code following line:50, it comes down to 13 lines of C code. Pretttty neat eh?...

Usage:
$> **sudo apt-get install libcurl4-gnutls-dev**
$> **gcc simple.c -o simple -l libcurl**
$> **./simple** <*shoutcast-server-ip-addr:port*> **>** <*test-file*>

After running the above commands, the ***<test-file>*** will contain the audio stream of that particular internet radio station. The can be played back in any player that supports decoding the stream format(AAC, MP3, OGG etc. depending on the radio station) Make sure to comment out line 38 in simple.c to have a glitch-free(no embedded metadata) audio stream.

This concludes part 1 of the series on how internet radio works. In part2 we will analyse the challenges and issues faced during de-packetising, parsing and queuing the audio stream buffers for local playback. **Stay tuned for updates**.

2 comments :

Unknown 6:27 am, October 27, 2014

Awesome tutorial! Thanks!

Reply

Anonymous 3:54 am, July 25, 2015

Nice one. Did part 2 ever show up?

Reply

Enter comment

<< Newer Post
Sensors on google glass

Home

Older Post >>
Hdd filesystems osync

SITEMAP

Home
About TheCodeArtist
Contact TheCodeArtist

Subscribe to TheCodeArtist feed

Follow @cvs26

linkedin.com/in/chinmayvs

MY WISHLIST

Stardew Valley
The Secret Ray comic by Herge
Bob & Bobette [set of 8 comics]
Das Keyboard [Cherry MX Brown]