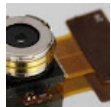




## POPULAR POSTS



Omnivision  
ov3640 i2c  
sccb



Simulating  
keypress  
events on  
Android



Internet Radio  
: Part1 -  
Shoutcast  
Protocol



Sensors on  
Google Glass



Proximity  
Sensor on  
Android  
Gingerbread



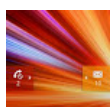
5 ways to  
improve HDD  
speed on Linux



Cost-Effective  
Communicatio  
n



Booting  
Android  
completely  
over ethernet



Android  
Double-  
buffering,  
Page-Flip and  
HDMI

Quilt : This patch can be  
reverse applied

## SEARCH THIS BLOG...


[Tweet](#)

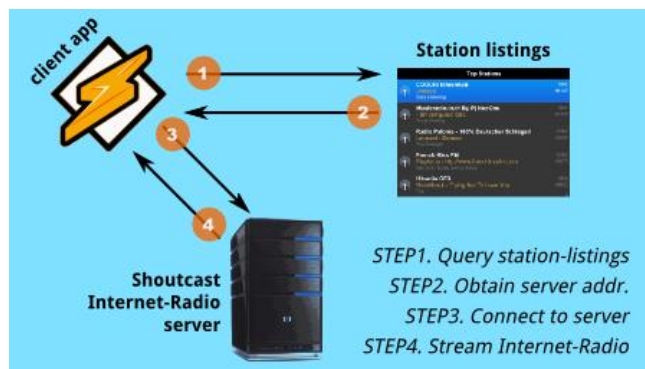
## Internet Radio : Part1 - Shoutcast Protocol

>> [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). **Icccast** 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

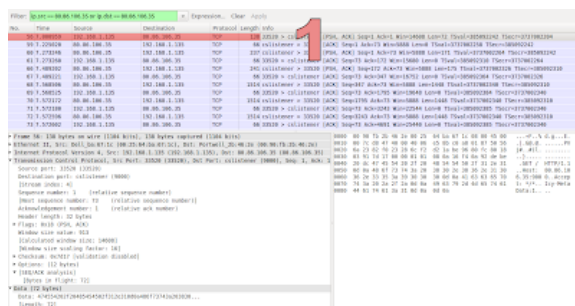
The client app connects to a station listing/agggregator on the internet and obtains a list of stations alongwith their details like genres, language, now-playing, bitrate among other things.

## [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/agggregator. 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/agggregator 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/agggregator 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.



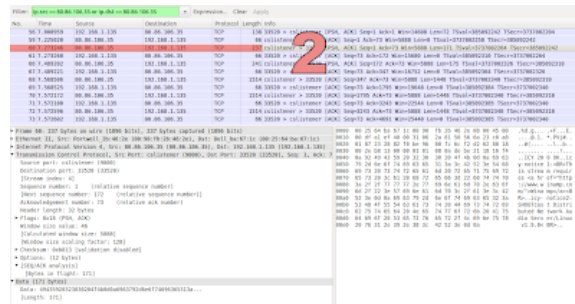
Connection request from shoutcast client (click to enlarge)

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

## READ POSTS ON...

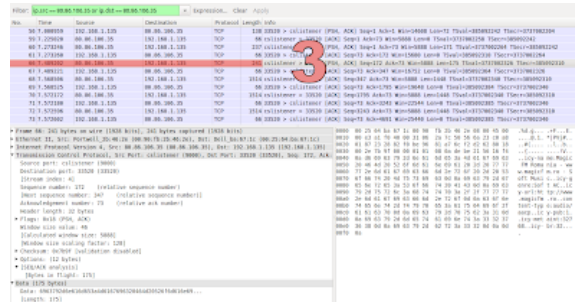
## Programming

( 23 ) [Android](#) ( 9 )  
[CodeProject](#) ( 9 ) [Sensors](#)  
 ( 6 ) [linux-kernel](#) ( 6 ) [Misc.](#)  
 ( 5 ) [linux](#) ( 5 ) [Software Engineering](#) ( 4 ) [Agile](#) ( 3 )  
[Creative thinking](#) ( 3 ) [HDD](#) ( 3 )  
[Magnetic-Field sensor](#) ( 3 ) [git](#) ( 3 )  
[Accelerometer Sensor](#) ( 2 ) [C](#) ( 2 )  
[DroidCon2011](#) ( 2 ) [Orientation Sensor](#) ( 2 )  
[Proximity-sensor](#) ( 2 ) [SATA](#) ( 2 ) [Test-Driven-Development](#) ( 2 )  
[cache](#) ( 2 ) [nGPS](#) ( 2 ) [page-cache](#) ( 2 )  
[product management](#) ( 2 ) [ARM](#) ( 1 )  
[Bonds](#) ( 1 ) [Camera](#) ( 1 ) [Display](#) ( 1 )  
[Finance](#) ( 1 ) [Game](#) ( 1 ) [Google Glass](#) ( 1 )  
[VFS](#) ( 1 ) [Python](#) ( 1 ) [Quilt](#) ( 1 )  
[Shoutcast](#) ( 1 ) [Use-case](#) ( 1 ) [adb](#) ( 1 )  
[chocosticks](#) ( 1 ) [i2c](#) ( 1 ) [internet-radio](#) ( 1 )  
[msleep](#) ( 1 ) [ov3640](#) ( 1 ) [project planning](#) ( 1 )  
[scsb](#) ( 1 ) [softirq](#) ( 1 ) [workqueue](#) ( 1 )



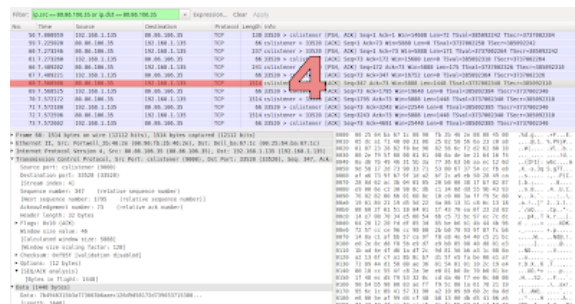
ICY 200 OK reply from shoutcast server (click to enlarge)

## 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).



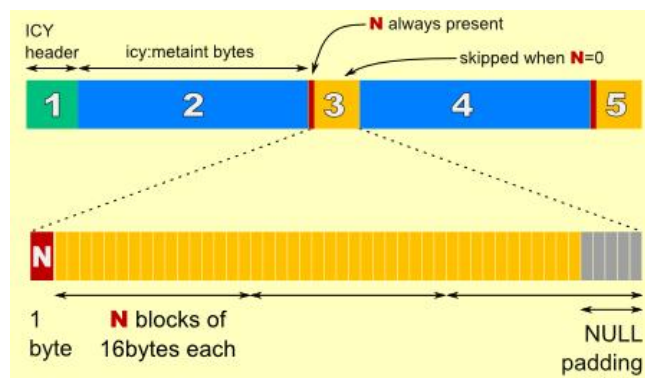
Encoded audio data stream (click to enlarge)



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 additional field in the request header. When set, it's 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-Metadatas.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

00008130	88 02 48 72 F8 82 B9 9C A3 7D 70 0B FD CD 10 3D	? .Hr?????}p.??.=
00008140	94 40 CC 6F 0A D0 30 A8 8A 29 7C 3D F8 EA EC EA	?@?o.?0???) =????
00008150	3A 34 48 F9 25 EF E1 6F 33 B4 02 53 74 72 65 61	:4H7%?7o37,Strea
00008160	6D 54 69 74 6C 65 3D 27 27 3B 53 74 72 65 61 6D	mTitle='';Stream
00008170	55 72 6C 3D 27 27 3B 00 00 00 00 F8 10 00 FF FF	Url='';...?.??
00008180	F9 5C 88 19 C1 64 21 19 D5 43 30 44 C5 3A 02 4D	?\?.?d!?.?C00?:.M
00008190	78 58 53 0A D0 B1 A6 C2 F0 3F EF 7B 76 2E 99 09	xxS.???????(v.?.

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.

```
1  /* listencast.c
2   * A simple shoutcast client that
3   * - Connects to any shoutcast server and
4   * - Logs the audio stream data to stdout
5   *
6   * Date 12-02-2013
7   *
8   * Requires libcurl
9   * $> sudo apt-get install libcurl4-gnutls-dev
10  *
11  * Building listencast.c
12  * $> gcc listencast.c -o listencast -lcurl
13  *
14  * Running listencast
15  * $> ./listencast <ip-addr>:<port> > <test.file>
16  * Attempts to connect to the shoutcast server
17  * running at <ip-addr> on port <port>. If successful
18  * then writes the audio stream into the file <test.file>
19  */
20
21  #include <curl/curl.h>
22
23  int main(int argc, char* argv[])
24  {
25      CURL *curl;
26      CURLcode res;
27
28      /* URL of the radio-station in <ip-addr>:<port> format */
29      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 executed
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

```

listenacast.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. Pretty 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

MY WISHLIST

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

View my profile on

linkedin.com/in/chinmayvs

TheCodeArtist

21k 4 69 130