Registrare la radio in streaming

Un'analisi delle tecniche usate dagli audio stream ripper per la registrazione delle trasmissioni radiofoniche diffuse via Internet

di Antonino Calderone

La capacità di registrare una radio che trasmetta in streaming non è propria dei programmi per la riproduzione, come per esempio Winamp ([1]). Perché?

La risposta che ci sembra più plausibile è che tale limitazione sia imputabile a possibili implicazioni legali.

Trasmettere musica è lecito a patto che si paghino i diritti d'autore (in Italia la SIAE [2]). Perché allora dovrebbe essere un problema registrare una trasmissione radiofonica, per poi risentirla quando se ne ha il tempo sapendo che i diritti sono stati già pagati da chi trasmette? Sono forse fuorilegge i registratori audio (o video) tradizionali?

È scontato che dal punto di vista legale siamo tenuti a mantenere un comportamento virtuoso, evitando di fare cose che ledano la libertà e i diritti degli altri, ivi compresi quelli degli artisti. In definitiva, possiamo ritenerci liberi di registrare ciò che ci piace a patto di non utilizzare questa pratica per aggirare la legge sui diritti d'autore ([3]) supposto che quello che intendiamo registrare sia sottoposto a tale restrizione. A ognuno la responsabilità di fare scelte sagge.

Antonino Calderone acalderone@infomedia.it

Lavora come software engineer nella divisione Broadband Networks di Ericsson. È inoltre socio fondatore della DigiFacta SW Engineering S.r.l. Si occupa principalmente di sistemi e protocolli per le telecomunicazioni, device driver, internals di sistemi operativi, firmware.

Assumendo superati gli aspetti legali, rimangono quelli tecnici. A questi ultimi, in effetti, siamo maggiormente interessati e dunque costituiranno il nostro argomento di discussione.

La radio via Internet

Dicevamo che Winamp (così come altri software per la riproduzione audio in streaming)

L'**MMS** (Multi Media Server protocol) è uno streaming protocol applicativo proprietario di Microsoft. Questo protocollo è stato concepito per la diffusione in streaming di contenuti multimediali (audio/video). MMS può essere trasportato sia in sessioni TCP che in datagram UDP.

L'**RTP** (Real-Time Protocol) è un protocollo applicativo che fornisce funzioni di trasporto end-to-end per la trasmissione di dati in tempo reale, come lo streaming audio/video; supporta sia la trasmissione in unicast che in multicast.

L'**RTSP** (Real Time Streaming Protocol) è un protocollo utilizzato per fornire un controllo remoto per i server che distribuiscono contenuti audio/video in Rete. Usato con il protocollo RTP, viene impiegato per lo streaming audio/video. RTP e RTSP sono sono standard IETF descritti rispettivamente nelle RFC 1889 e 2326.

RIQUADRO 1 Breve descrizione dei protocolli MMS, RTP e RTSP

può riprodurre, ma non registrare su file lo stream audio proveniente da una radio che trasmetta via Internet. Questo non è di per sé un problema. Esistono in Rete molti cosiddetti "audio ripper" che suppliscono a questa carenza; ma come funzionano questi programmi?

Per rispondere a questa domanda dobbiamo approfondire almeno due aspetti tecnici: il formato dello stream audio e il protocollo usato per la diffusione dello stesso.

Sia per il primo che per il secondo punto non si ha una sola opzione. I formati utilizzati per la trasmissione di stream audio sono diversi. Certamente il formato maggiormente diffuso è l'*MPEG-1 audio layer 3* meglio noto col nome di *MP3* ([4]). Un altro formato con caratteristiche simili all'MP3, assai diffuso, è il formato *WMA* (*Windows Media Audio*) sviluppato da Microsoft. Mentre, tra i protocolli usati per la diffusione sono largamente impiegati l'*HTTP*, l'*MMS* e lo *RTSP/RTP* (si veda il **Riquadro 1** per maggiori informazioni).

Winamp, come molti altri riproduttori, supporta il formato MPEG trasmesso tramite HTTP (e/o sue estensioni). Uno dei vantaggi di usare HTTP è quello che questo protocollo supera l'ostacolo dei proxy e/o dei firewall nella tipiche configurazioni delle intranet di grandi organizzazioni. Questo fatto non è trascurabile per una radio che trasmetta via Internet e voglia raggiungere la più ampia platea possibile. Certamente l'HTTP non è il protocollo ideale per il "broadcasting" anche se funziona abbastanza bene, a patto di non avere grossi problemi di banda.

Eviteremo di sviscerare i misteri dei vari protocolli e dei vari formati, concentrandoci invece su quelli che riteniamo maggiormente diffusi, ovvero l'accoppiata HTTP ed MPEG provando, quando possibile, a generalizzare.

FIGURA 1 Struttura generale di uno stream MPEG audio

MPEG Audio Layer I/II/III framing

Di MPEG si è molto parlato in passato nelle riviste del gruppo; in particolare vorremmo segnalare gli articoli [5] e [6] (che possono essere consultati anche on-line tramite il sito di Infomedia). Non riteniamo utile ripetere quanto autorevolmente scritto, pertanto rimandiamo il paziente lettore che desiderasse avere un'idea sulla genesi e sul funzionamento della codifica MPEG, alla lettura di quegli articoli (e alla consultazione dei riferimenti forniti in calce al nostro).

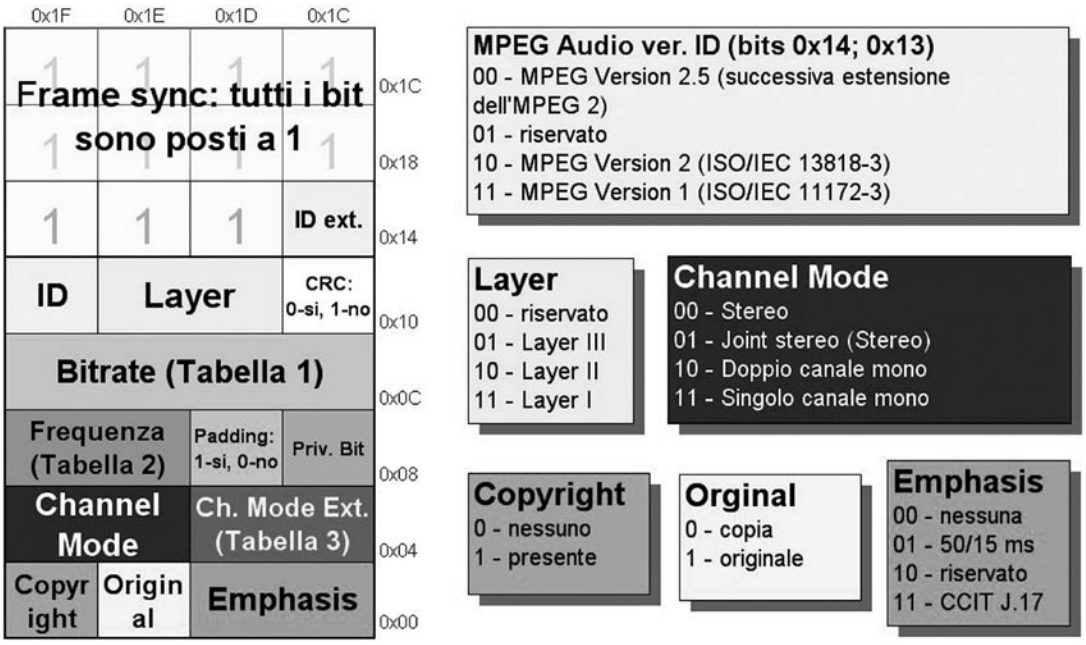
La ragione per la quale non ci occuperemo della codifica MPEG è semplice. Non ci interessa: non è necessario conoscerla per procedere con la registrazione dello stream MPEG su file. Ottenuto il file audio dallo stream lasceremo comunque l'onere della riproduzione al nostro lettore MPEG preferito. Quello che invece ci interessa maggiormente, e che quindi intendiamo approfondire, è come sia strutturato un generico stream (o equivalentemente un generico file) MPEG audio.

Un file MPEG audio non ha un header principale, ma piuttosto una successione di parti denominate *frame* (**Figura 1**). Ogni frame è costituito da un blocco dati preceduto dal proprio header. Per i layer I e II, i frame sono entità completamente indipendenti, il che consente di "tagliare" un file MPEG e di riprodurne le sue parti in modo altrettanto indipendente. Per quanto riguarda il layer III, questo non è sempre vero; comunque nel peggiore dei casi un riproduttore ha bisogno di un numero limitato di frame prima di essere in grado di cominciare la decodifica.

Un *frame header* (**Figura 2**) è costituito da una sequenza di 32 bit, più in dettaglio:

- il campo *frame sync* è costituito da 12 bit (11 nel caso delle estensioni MPEG 2.5), posti a 1.

FIGURA 2 Header di un frame MPEG audio



I frame possono opzionalmente avere una CRC di 16 bit, che se presente, è posta di seguito all'header stesso (precedendo i dati veri e propri);

- l'*id* è un campo rappresentato da due bit (considerando l'estensione MPEG 2.5) attraverso il quale si determina la versione del formato MPEG;
- il *layer* è determinato da due bit e più precisamente: la combinazione 00 è riservata, 01 è usata per il Layer III, 10 per il Layer II, e 11 per il Layer I;
- il successivo *protection bit*, indica la presenza o meno della CRC; vale 0, quando la CRC è presente;
- il campo *bit rate* è codificato con 4 bit, con il significato espresso nella **Tabella 1**;
- i due bit denominati *sampling rate frequency*

TABELLA 1 Codifica del campo bit rate

Bit	MPEG-1 layer I	MPEG-1 layer II	MPEG-1 layer III	MPEG-2 layer I	MPEG-2 layer II e III
0000	*free*	*free*	*free*	*free*	*free*
0001	32	32	32	32	8
0010	64	48	40	48	16
0011	96	56	48	56	24
0100	128	64	56	64	32
0101	160	80	64	80	40
0110	192	96	80	96	48
0111	224	112	96	112	56
1000	256	128	112	128	64
1001	288	160	128	144	80
1010	320	192	160	160	96
1011	352	224	192	176	112
1100	384	256	224	192	128
1101	416	320	256	224	144
1110	448	384	320	256	160
1111	*n.c.*	*n.c.*	*n.c.*	*n.c.*	*n.c.*

Note:
I valori sono espressi in **kbps**; *free* indica il formato libero; n.c. indica che il valore non è consentito.
Per il layer II esistono alcune combinazioni di bit-rate e "*mode*" che non sono consentiti

TABELLA 2 Codifica del sampling rate frequency index

bit	MPEG1	MPEG2	MPEG2.5
00	44100 Hz	22050 Hz	11025 Hz
01	48000 Hz	24000 Hz	12000 Hz
10	32000 Hz	16000 Hz	8000 Hz
11	riserv.	riserv.	riserv.

index forniscono un indice per determinare la frequenza di campionamento (**Tabella 2**);

- il bit di *padding* è usato per "aggiustare" i frame in modo da rispettare il bit rate specificato;
- il cosiddetto *private bit* è un bit utilizzabile a scopo "informativo";
- le informazioni sui *channel mode* (due bit) si riferiscono al canale audio (stereo, mono, ecc...);
- il *mode extension* (due bit) è usato solo per le informazioni nella modalità joint stereo: il range di frequenze del file MPEG è suddiviso in 32 sotto-bande. Per i layer I e II questi bit rappresentano le bande a cui si applica l'intensity stereo; per il layer III, determinano lo stato (on/off) della modalità intensity stereo o m/s (**Tabella 3**);
- gli ultimi campi sono: un bit per l'indicazione sulla presenza (1) del *copyright*;
- il bit *original* vale 1 se il brano è originale; 0 nel caso sia una copia;
- gli ultimi due bit sono riservati al campo *emphasis*, col seguente significato: 00 - nessuna, 01 - 50/15 ms, 11 - CCIT J.17, 10 - riservato.

TABELLA 3 Codifica dei bit del campo "mode extension"

bit	Layer I e II	Layer III	
	Bande	*Intensity stereo*	*MS stereo*
00	da 4 a 31	off	off
01	da 8 a 31	on	off
10	da 12 a 31	off	on
11	da 16 a 31	on	on

Un file MPEG può inoltre avere *tag* in coda ai dati. Uno di questi, denominato *MPEG Audio Tag ID3v1*, è costituito da 128 byte che forniscono informazioni sul brano, come per esempio il titolo, l'artista, il genere e altro.

La precedente descrizione dell'header (si vedano i riferimenti [7] e [8] per approfondire ulteriormente) ci fornisce la caratterizzazione della struttura del file MPEG audio, e ci consente di comprendere come si possa suddividere uno stream continuo, come quello trasmesso dalle radio, in parti auto-consistenti.

Detto questo, è chiaro che la sola conoscenza del formato audio non basta. Per catturare uno stream e "tradurlo" in un file da riprodurre off-line è indispensabile conoscere il protocollo usato per la sua diffusione (almeno limitatamente al dominio di interesse). Vediamo come questo ruolo venga svolto dal protocollo HTTP.

Due parole sull'HTTP e sue estensioni per lo streaming

Il protocollo HTTP (*Hypertext Transfer Protocol*, definito nell'RFC 2616) è il protocollo applicativo usato principalmente per la diffusione di contenuti ipertestuali sul web.

Non ci occuperemo di darne una descrizione esaustiva, assumendo che il lettore abbia familiarità con il protocollo, almeno per le caratteristiche principali, anche se ci sembra utile ribadire alcune cose propedeutiche per quanto verrà detto nel seguito.

L'HTTP segue un paradigma orientato alle transazioni basato, in particolare, sul modello richiesta/risposta: il client inoltra una richiesta a un server che replica con una risposta. Le sessioni di trasporto TCP usate per il trasferimento di questo scambio in genere sono stabilite e terminate per ciascuna transazione. Questo comportamento rende semplice la gestione dei collegamenti ipertestuali che per loro natura permettono di mischiare contenuti appartenenti a fonti distinte. Il modello richiesta/risposta può essere reso più complesso nel caso in cui il client debba interagire con un proxy. Il server invece non sa dell'esistenza di questo intermediario, e non se ne preoccupa.

Un generico messaggio di richiesta HTTP è costituito da tre parti: il metodo della richiesta (*GET*, *HEAD*,

POST, ecc...) seguito da un *URI* e dalla versione del protocollo; una sezione denominata *header* che contiene informazioni aggiuntive sul metodo, sotto forma di attributi (*header field*), e infine, se presente, il corpo del messaggio o *body*. Una risposta è costituita da una *status-line* contenente un codice di tre cifre che fornisce l'esito della risposta stessa; per esempio una status-line con la dicitura "HTTP/1.1 200 OK" indica che la richiesta ha avuto successo. Alla status-line seguono l'*header* e (opzionalmente) il body del messaggio. Questo è un generico *octet stream* (anche se, come diremo tra poco, esiste una sua caratterizzazione che consente al ricevente di classificarlo in modo appropriato). La parte che precede il body, è una sequenza di linee di testo ASCII, ognuna terminata dalla coppia dai caratteri CR (0xA) e LF (0xD). Riassumendo una generica richiesta HTTP ha il seguente formato:

```
<metodo> <URI> <ver. prot.> <CRLF>
<header_field _1>: <valore1> <CRLF>
<header_field _2>: <valore2> <CRLF>
...
<header_field _N>: <valoreN><CRLF>
<CRLF>
<body ... stream di ottetti>
```

Analogamente una generica risposta è formattata come segue:

```
<ver. prot.> <status code> <status msg.>
                                        <CRLF>
<header_field _1>: <valore1><CRLF>
<header_field _2>: <valore2><CRLF>
...
<header_field _N>: <valoreN><CRLF>
<CRLF>
<body ... stream di ottetti>
```

Si noti che l'ultima linea dell'header è individuata da una doppia sequenza <CRLF>. In pratica l'header non ha una grandezza prefissata, ma la sua fine e l'inizio del corpo del messaggio sono individuati da questa sequenza. Il body, invece può sia avere una grandezza predeterminata (un attributo dell'header denominato *Content-Length* è usato appositamente), oppure può essere terminato nel momento in cui viene chiusa la sessione di trasporto per la specifica connessione TCP. Per classificare e trattare correttamente gli oggetti trasferiti, è impiegato un attributo dell'header denominato *Content-Type*. Esiste una codifica dei tipi di dati inviati, descritta nella RFC 1521, che fornisce una classificazione del tipo di contenuto trasferito. Questa classificazione prende il nome di *Multimedia Internet Mail Extensions* (*MIME*) perché in origine fu pensata per i messaggi posta elettronica, e successivamente utilizzata per i contenuti di altri protocolli tra i quali l'HTTP. La classificazione MIME è costituita da un tipo, un sottotipo e opzionalmente un parametro. Per esempio, un normale testo potrebbe essere classificato specificando l'attributo *Content-Type: text/plain; charset=us-ascii*.

I protocolli maggiormente usati per la diffusione audio in streaming sono l'HTTP, l'MMS e lo RTSP/RTP

Il tipo MIME usato per classificare un file audio MPEG è *audio/mpeg* (indipendentemente dal layer utilizzato).

Basandosi sul solo Content-Type non è possibile caratterizzare completamente uno stream audio. Per questa ragione sono state sviluppate una serie di estensioni al protocollo HTTP che mirano a risolvere questa carenza. Tra gli altri, la Nullsoft, azienda creatrice di Winamp, ha sviluppato un'estensione dell'HTTP denominata *SHOUTcast* ([9]) con l'intento di aggiungere all'HTTP il supporto per lo streaming audio e video.

In particolare, l'estensione riguarda nuovi tag usati come campi dell'header; più precisamente essi sono:

- icy-notice – un tag usato per fornire informazioni generiche;
- icy-name – il nome del canale dal quale si riceve;
- icy-genre – il genere che si sta ascoltando;
- icy-url – fornisce l'URL del sito che trasmette;
- icy-pub – indica se lo stream è pubblico (1) o privato (0);
- icy-br – indica il bit-rate espresso in kbit/s.

Quando Winamp vuole "collegarsi in streaming" con una radio inoltra al server della stazione radio una richiesta del tipo:

```
GET <uri dello stream> HTTP/1.0
Accept: */*
User-Agent: <winamp agent>
Host: <host remoto>
```

In altri termini al server remoto viene chiesto essenzialmente di fornire una risorsa, che a priori si sa essere uno stream MPEG. Il server, se tutto va per il meglio, risponde con qualcosa del tipo:

```
HTTP/1.0 200 OK
Connection: close
Content-Type: audio/mpeg
icy-br:<bitrate in Kbit/s>
icy-description:free music from all over
                                    the globe
icy-genre:<genere musicale>
icy-name:<descrizione della radio>
icy-pub:<1 pubblica, 0 privata>
icy-url:<url della radio>
Server: <descrizione del tipo di server>
.... dati .....
.... dati .....
```

Le estensioni al protocollo HTTP come *SHOUTcast* forniscono informazioni ausiliarie sullo stream inviato, come per esempio la descrizione della stazione radio, la banda in termini di bit-rate, e altro. Il riproduttore mostra in genere queste informazioni all'utente tramite la propria interfaccia utente.

Quanto discusso ci porta a concludere che i riproduttori audio basati su HTTP si comportano essenzialmente come normali HTTP client: inoltrano una normale richiesta usando il metodo GET e ottengono (a meno di errori) uno stream, che per sua natura non ha soluzione di continuità (*Content-Length* non specificato). Inoltre comprendono dal *Content-Type* e dagli eventuali *tag* specifici dovuti ad estensioni del protocollo, come trattare lo stream ricevuto.

Wget ed Ethereal: ripper per caso

Un audio stream ripper si comporta rispetto una stazione radio alla stregua di un qualunque altro client ovvero di un riproduttore. Sfruttando la conoscenza del framing del formato audio (pur ignorando il contenuto intrinseco della codifica) è in grado di dividere lo stream acquisito in file distinti e auto-consistenti, e di operare una classificazione di quanto catturato. Un ripper (che non sia anche riproduttore audio), fornisce spesso funzionalità di (transparent) proxy per il riproduttore vero e proprio, rimediando alla necessità di aprire una seconda sessione con la stazione radio, nel caso in cui si desiderasse riprodurre l'audio durante la registrazione.

Se un ripper in soldoni è di fatto un normale client, nel caso specifico (MPEG inviato tramite HTTP) cosa ci impedisce di usare un normale client HTTP, come per esempio wget (del quale si trovano eseguibile, documentazione e sorgenti all'URL [10]), per registrare l'audio in streaming? Evidentemente, se le nostre congetture sono corrette possiamo effettuare un esperimento. Vi proponiamo il seguente.

Un file MPEG audio non ha un header principale: è costituito da una successione di parti denominate frame

Si provi ad eseguire wget da un host che possa raggiungere Internet, eseguendo il seguente comando:

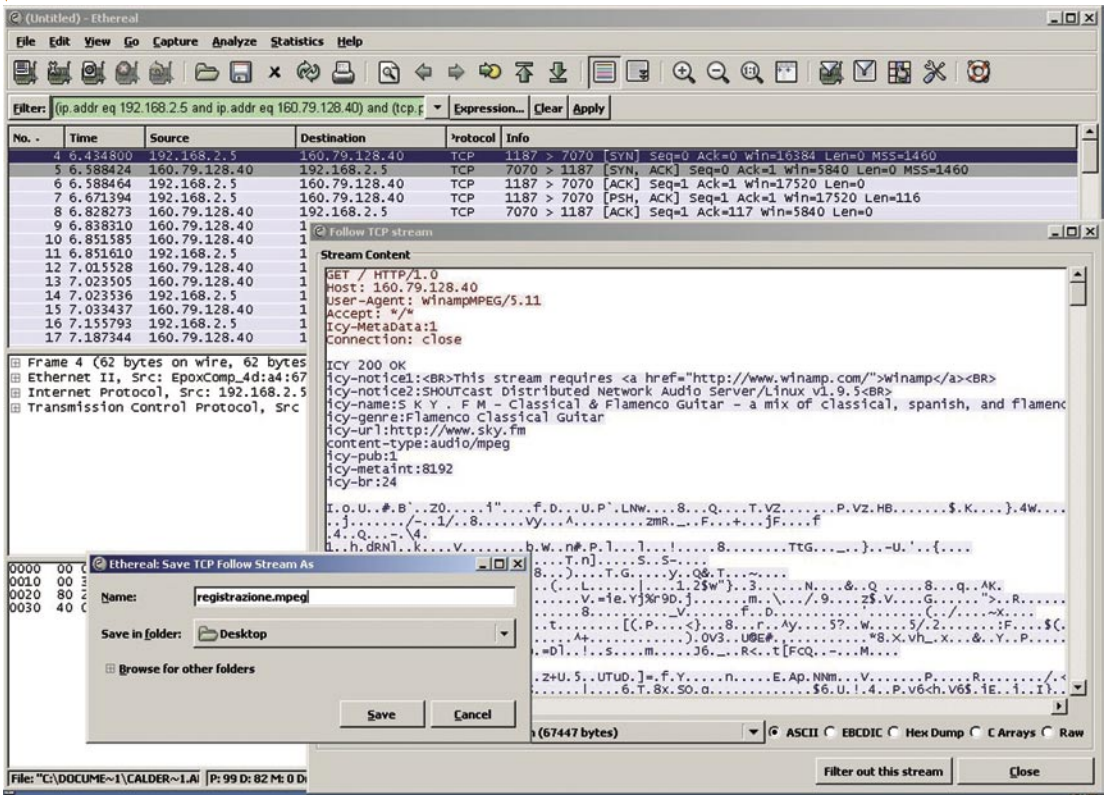
```
$ wget http://audio1.cruzio.com:8000/live32
                                    -O radiolive.mpeg
```

Se la stazione radio (nell'esempio abbiamo scelto, abbastanza casualmente, una stazione radio di Santa Cruz [11]) risponderà correttamente, si dovrebbe ottenere un output simile al seguente:

```
--20:49:39--  http://audio1.cruzio.com:8000/
                                          live32
           => `radiolive.mpeg'
Resolving audio1.cruzio.com... 63.249.95.25
Connecting to audio1.cruzio.com|63.249.95.25|
                                  :8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [audio/mpeg]
    [    <=>    ] 147,000         3.87K/s
```

Specificando il parametro *–O radiolive.mpeg* chiediamo a wget di salvare la risorsa richiesta, cioè lo stream audio nel file *radiolive.mpeg*. Nel caso ci sia di mezzo un HTTP proxy è necessa-

FIGURA 3 Esempio di cattura e salvataggio su file di uno stream MPEG audio con Ethereal



rio impostare la variabile d'ambiente *http_proxy* e specificare il parametro *–proxy=on*, quindi se richiesta l'autenticazione, anche i relativi parametri *--proxy-user=<utente>* *--proxy-passwd=<password>*. Per interrompere la registrazione è ovviamente sufficiente interrompere *wget*. L'esito dell'esperimento ci fornisce la risposta alla precedente domanda.

SHOUTcast è un'estensione del protocollo HTTP che aggiunge il supporto dello streaming audio e video

Le estensioni al protocollo HTTP possono includere anche modifiche che rendono però wget non in grado di comprendere la risposta del server: di solito sono tollerati i parametri dell'header non standard, ma questo non vale per la *status-line*. Se proprio volessimo continuare a usare wget anche in questi casi, un rimedio potrebbe essere quello di modificare questo tool così da permettergli una corretta interpretazione delle risposte non standard. La morale la conosciamo: un programmatore ha sempre una seconda chance, quella di passare all'azione. Ma prima che il lettore intraprendente abbandoni la lettura per passare ai fatti, vorremmo parlare di un secondo metodo di registrazione che fa uso di una tecnica che definiamo di "sniffing".

L'idea è quella di catturare il traffico di rete dell'interfaccia pubblica durante la riproduzione in streaming ad opera del lettore MPEG ed estrarre da questo il contenuto informativo che coincide con il file audio. A partire da questi elementi possiamo condurre un secondo esperimento. Innanzi tutto ci serve un analizzatore di protocolli come per esempio *Ethereal* ([12]). L'esperimento consiste nel catturare i frame dall'interfaccia pubblica durante la riproduzione fatta dal lettore in streaming. I frame catturati appartengono a una sessione TCP. Possiamo chiedere ad Ethereal di ricostruire gli stream della sessione TCP (ci basta individuare almeno un frame della sessione, per esempio quello che contiene la richiesta di GET e usare il comando *Follow TCP Stream* dal

menù *Analyze* tenendo selezionato tale frame). Siamo ovviamente interessati allo stream di dati inviati dal server (la radio) verso il client (il riproduttore). Ottenuto questo flusso, possiamo effettuarne il salvataggio in un file con estensione *.mpeg* (come illustrato nell'esempio di **Figura 3**). Infine, dando in pasto questo file al riproduttore si dovrebbe riascoltare quanto riprodotto durante la cattura.

Vi sarete accorti che il file binario con estensione *.mpeg* ha come preambolo anche la risposta alla GET. Questo fatto non impedisce al riproduttore di eseguire comunque il file: per sua natura un file MPEG può essere riprodotto anche se immerso in uno stream non MPEG; i riproduttori MPEG sono in genere in grado di operare un "parsing" selettivo del frame "buttando via quello che MPEG non è". Quindi per ottenere un file audio MPEG dallo stream catturato, tutto quello che bisogna fare (ed è quello che abbiamo fatto) è salvarlo con un'estensione che consenta al riproduttore di convincersi che dentro troverà un file audio, riproducibile (la generica estensione .mpeg va bene, anche se dentro ci fosse un mp3). Questo potrebbe non essere vero per altri formati. Quindi un eventuale file binario ottenuto ricostruendo lo stream di dati trasmessi dovrebbe subire una post-elaborazione attraverso la quale estrarre le sole informazioni utili alla ricostruzione del file audio privato dell'overhead del protocollo di comunicazione. Nel nostro esperimento siamo stati aiutati anche dal fatto che l'HTTP viaggia in sessioni TCP e che Ethereal sia capace di ricostruire tali sessioni.

Per quanto più complesso, l'approccio di "sniffare" le informazioni dal protocollo di trasporto e riversarle su file è dunque possibile. In realtà più che pensare a questo come un metodo di registrazione da implementare, desideravamo porre l'accento sull'uso dell'analizzatore di protocolli soprattutto come strumento di debugging, utile sia che si intenda provare a scrivere una propria implementazione di un audio stream ripper, sia che si voglia provare a "giocare" con software di terze parti.

Conclusioni

Abbiamo descritto un protocollo e un formato audio compresso scegliendoli in base alla loro popolarità e diffusione, sperimentando con essi le tecniche maggiormente impiegate per la registrazione delle radio che trasmettono in streaming. Quello che il lettore si chiederà è se i metodi descritti per HTTP ed MPEG siano generalizzabili per altri protocolli e formati audio. Noi pensiamo che la risposta sia affermativa: in estrema sintesi abbiamo visto che sono necessari la conoscenza del formato del framing dello stream audio e quella del protocollo per il trasferimento dei dati. Noti questi due elementi, il resto viene da sé.

Ci congediamo indicando che è possibile trovare in Rete vari software completi di sorgenti, come per esempio il programma *streamripper* (che può essere scaricato liberamente all'URL [13]). Chi volesse macinare un po' di codice potrebbe cominciare a dare una sbirciata ai sorgenti di questi tool.

Voglio ringraziare Gianluca Insolvibile che come sempre mi ha fornito materiale e suggerimenti. Il mio collega Alessandro Martin (Alli) che ha ispirato con i suoi lavori la stesura di quest'articolo. Renzo Boni per la pazienza che mi dimostra e per il materiale che mi ha inviato.

BIBLIOGRAFIA & RIFERIMENTI

[1] http://www.winamp.com
[2] http://www.siae.it
[3] http://www.parlamento.it/leggi/00248l.htm
[4] http://www.mpeg.org
[5] G. Insolvibile - "MP3, per chi non lo sapesse…", DEV 82, Febbraio 2001, Gruppo Editoriale Infomedia
[6] G. Insolvibile – "MPEG: la saga continua…", DEV 91, Dicembre 2001, Gruppo Editoriale Infomedia
[7] http://www.mp3-tech.org/programmer/frame_header.html
[8] http://www.dv.co.yu/mpgscript/mpeghdr.htm#MPEGTAG
[9] http://www.shoutcast.com/
[10] http://www.gnu.org/software/wget/
[11] http://www.kusp.org/
[12] http://www.ethereal.com
[13] http://sourceforge.net/projects/streamripper/