POLITECNICO DI TORINO

Collegio di Ingegneria Informatica

Corso di Laurea Magistrale

in Ingegneria Informatica

Tesi di Laurea Magistrale

Estensione delle funzionalità di un encoder multimediale per la gestione dei metadati in tempo reale

Relatore:
Prof. Antonio Servetti

Candidato:
Francesco Ancona (M. 206746)

APRILE 2018

Keywords:
Multimedia, streaming, metadata, GStreamer, encoder

Indice

Indice delle figure

1 Introduzione

Negli ultimi anni abbiamo assistito ad una veloce e rapida diffusione del servizio di streaming Internet. In tal senso le emittenti radio tradizionali si sono mosse verso il digitale, poggiandosi sul protocollo SHOUTcast che risulta già di vecchia data. Nuove tecnologie infatti si stanno affacciando sul mercato, offrendo molte più funzionalità (es. Apple HTTP Live Streaming e MPEG Dynamic Adaptive Streaming over HTTP).

Come sappiamo, la maggior parte dei player odierni sono in grado di leggere i metadati del singolo file audio in ingresso (ad esempio i tag ID3 presenti in un file audio di tipo MP3), ma anche i metadati presenti in un flusso multimediale real-time. La parte di un'architettura di streaming che provvede all'immissione dei metadati nel flusso audio viene sviluppata solitamente nell'encoder, ovvero quel componente hardware/software che rende il flusso audio analogico idoneo alla trasmissione in streaming. Tuttavia, non esistono al giorno d'oggi encoder open source in grado di iniettare i metadati nello streaming multimediale in tempo reale.

L'obiettivo che questo lavoro si è prefissato è quello di implementare, a livello di encoding, un servizio aggiuntivo che preveda l'immissione nello streaming audio dei metadati di ciò che viene riprodotto.

Tramite la modifica di alcuni moduli del framework open source GStreamer, vedremo come sia possibile creare un sistema che permetta allo stesso tempo di leggere l'audio da un ingresso analogico, e di mettersi in ascolto su una porta TCP per ricevere i metadati che si intende inviare nel flusso audio in tempo reale.

2 Catena streaming

In questa sezione si descriverà in breve come funziona un esempio di architettura di streaming grazie alla quale un flusso audio live analogico prodotto da una regia audio viene reso fruibile sulla rete, dove ciascun utente è connesso con la propria tipologia di dispositivo (pc, smartphone, tablet ecc…).

Il flusso audio live riprodotto dalla sorgente, la regia audio, viene codificato, gli vengono aggiunti dei metadati (informazioni come i tag "artista" e "titolo" del brano riprodotto) e viene inviato a un server di streaming tramite il protocollo RTMP.

Il server di streaming è un pc che ha il compito di ricevere il flusso audio codificato e di distribuire agli utenti il contenuto multimediale via Internet, cosicché ciascun utente, tramite il suo dispositivo, sia esso laptop, tablet, o smartphone, potrà fruire del flusso audio attraverso il protocollo di streaming (Apple HLS, RTSP ecc…) più opportuno, a seconda della tipologia di software per il playback (player) utilizzato.

Quanto brevemente descritto avviene attraverso una *catena streaming*, i cui elementi verranno discussi nelle sezioni successive.

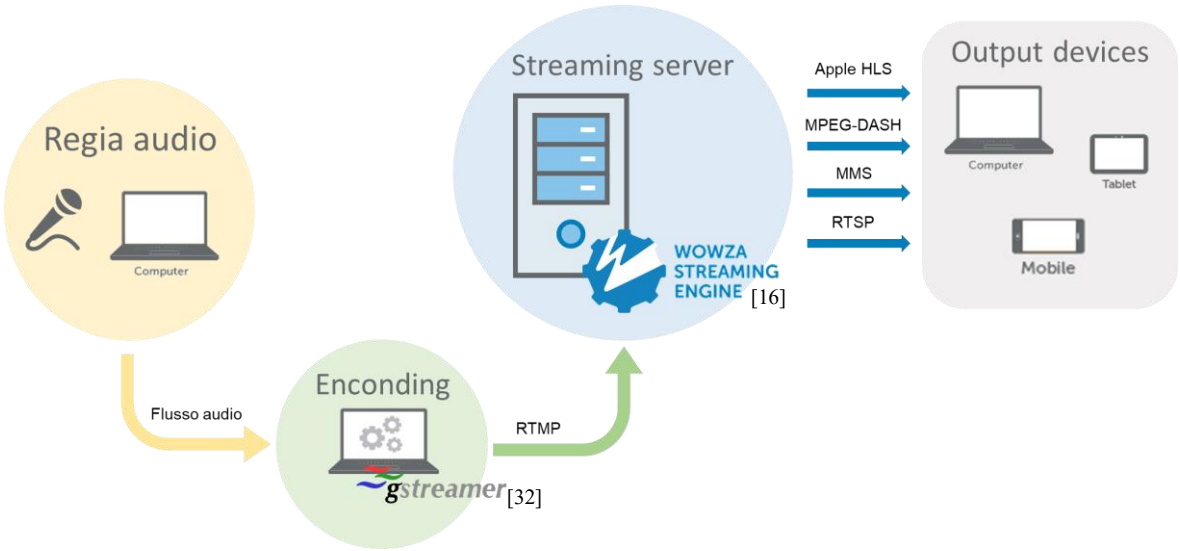

Figura 1 - Catena di streaming

2.1 Regia audio

Il compito della sorgente è quello di creare fisicamente il flusso audio che si vuole trasmettere agli utenti. Questo flusso può essere voce, musica o entrambi.

Il dispositivo preposto a fare da regia audio può essere la scheda audio di un pc o di qualunque dispositivo in grado di riprodurre del contenuto audio, a cui è collegato un microfono, in modo tale che il flusso possa essere composto sia dalla voce dello speaker, sia dal brano che la regia intende riprodurre.

Il flusso multimediale, una volta acquisito dalla scheda audio, può essere direttamente codificato da un encoder presente nello stesso pc, oppure può essere trasferito ad un altro dispositivo dotato di scheda audio e di software preposto alla codifica, collegato tramite un jack, ad esempio, al primo dispositivo.

2.2 Encoder

Il flusso audio grezzo giunge al computer addetto alla codifica. Su questo pc è installato l'encoder che deve assolvere al compito di rendere il formato del flusso idoneo alla trasmissione in streaming.

Nel nostro caso si è scelto di utilizzare come strumento di codifica un particolare *elemento* di elaborazione (plug-in) denominato *FAAC* (acronimo di Free AAC Encoder) [1], integrato come modulo del framework GStreamer.

Nei successivi paragrafi si descriveranno in breve GStreamer e i relativi plug-in utilizzati per acquisire il flusso audio grezzo, codificarlo, suddividerlo in segmenti (buffer) che saranno inseriti all'interno di pacchetti RTMP insieme ai metadati, i quali verranno infine inviati al media server.

2.3 Server di streaming

Affinché gli utenti (i client) collegati in rete possano fruire dello streaming multimediale, essi devono essere connessi al cosiddetto server di streaming, un computer preposto alla distribuzione in rete dei contenuti audio/video codificati che gli sono stati inviati dall'encoder.

In questa tesi si utilizzerà Wowza Streaming Engine come software per far funzionare il media server.

3 Strumenti utilizzati

In questa sezione si descriverà in breve GStreamer, introducendone alcuni dei concetti fondamentali come quelli di “elemento” e “pipeline”. Si vedranno degli esempi pratici di pipeline, e si porrà l’attenzione su quelli che saranno i principali plug-in sui quali si baserà l’implementazione successiva.

Si farà anche una panoramica del protocollo RTMP, su cui si poggia il funzionamento del plug-in utilizzato per l’invio al server dei pacchetti.

Infine si vedrà cosa è Wowza Streaming Engine, il software che gira sul server di streaming.

3.1 GStreamer

Gstreamer [2] è un framework open source che permette la creazione, la codifica, la decodifica e la trasmissione di flussi multimediali, tramite degli elementi collegati opportunamente tra loro. GStreamer consente in generale l’elaborazione di flussi dati di vario tipo, comprensivi di audio, video, metadati e file provenienti da sorgenti differenti.

GStreamer è prevalentemente realizzato in C, in modo da essere portabile e veloce. Aderisce al modello di programmazione GObject, che è un framework per la programmazione orientata agli oggetti in C, fornito dalla libreria multipiattaforma GLib [3].

3.1.1 Gli elementi

Gli *elementi (elements)* [4] costituiscono le unità base di elaborazione di GStreamer e possono essere utilizzati all’interno di una struttura denominata *pipeline*. Ciascun elemento è fornito da un determinato plug-in, che a sua volta è realizzato basandosi sull’uso di librerie già presenti nel sistema operativo del pc in uso, o comunque facilmente reperibili online.

I plug-in su cui si basano gli elementi di GStreamer si suddividono in Good (ben supportati), Bad (il cui codice necessita di una buona revisione), e Ugly (il cui utilizzo può causare qualche problema di compatibilità).

Un elemento può essere caratterizzato da uno o più ingressi e uscite, denominate rispettivamente *sink pad* e *source pad,* attraverso le quali, a partire dal flusso dati ricevuto, si può ottenere un output differente a seconda delle caratteristiche dell’elemento utilizzato.

Ogni elemento può avere nessuna, una o più *proprietà*. Le proprietà possono essere viste come dei parametri di impostazione attraverso le quali è possibile configurare l'elemento, prima dell'avvio della pipeline, al fine di attivare una particolare caratteristica dell'elemento stesso. Talvolta in mancanza di una configurazione specifica da parte dell'utente, possono essere assegnati alla proprietà dei valori di default.

3.1.2 La pipeline

La *pipeline* [5] è una sequenza di elementi collegati tra loro, e costituisce la struttura su cui si basa GStreamer per processare un flusso multimediale.

Il flusso dati che attraversa la pipeline consiste in una combinazione di *buffer* ed *eventi (events)*. Gli eventi contengono informazioni di controllo, come la notifica di ricezione di un metadato, di un comando di seeking (ad esempio per mettere in stato di "pause" la pipeline) oppure la fine dello stream (EOS). Il *buffer* è l'unità base in cui un segnale è suddiviso.

Il segnale viene partizionato in *buffer* già dal primo elemento che fa da source (sorgente). Dopodiché l'elemento successivo riceve il buffer tramite la sua interfaccia sink pad, e lo passa all'elemento seguente tramite il suo source pad, e così via fino all'ultimo elemento che riceverà il buffer.

Una pipeline può essere avviata da terminale tramite il comando *gst-launch*, seguito dagli elementi che compongono la pipeline stessa, disposti secondo l'ordine con cui il flusso deve essere elaborato, e separati tra loro da un punto esclamativo (!).

Ad esempio:

```
gst-launch-1.0 audiotestsrc is-live=true ! faac ! aacparse ! faad !
autoaudiosink
```

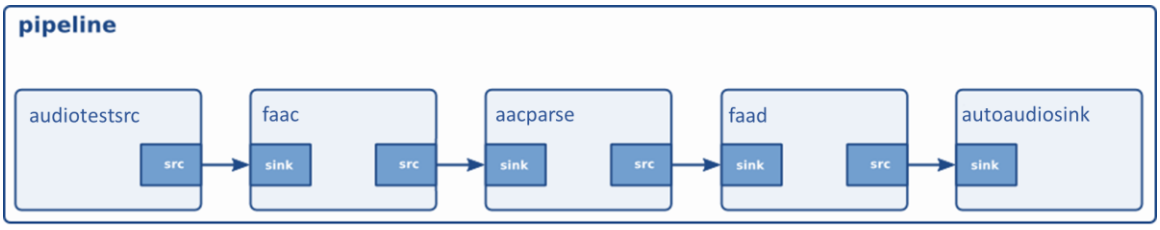

Figura 2 - Rappresentazione grafica di una pipeline GStreamer

In questa pipeline, rappresentata sia graficamente, sia come comando su terminale, e che verrà usata per testare la corretta installazione e configurazione di GStreamer (paragrafo 4.1), l'elemento *audiotestsrc* [6] genera un segnale audio base (una sinusoide) codificato dal plug-in *faac* [1] in un flusso in formato AAC, che passa attraverso il parser AAC [7] (*aacparse*),

per essere infine decodificato dal plug-in *faad* (Free AAC Decoder [8]) ed essere così riprodotto dalla scheda audio grazie all'ultimo plug-in della pipeline, *autoaudiosink* [9].

Un altro esempio di pipeline che vale la pena riportare al fine di introdurre gli altri elementi principali utilizzati nella tesi, è quella a cui si perverrà alla fine delle modifiche dei sorgenti di GStreamer relativi agli elementi *flvmux* e *taginject*:

```
$ gst-launch-1.0 audiotestsrc is-live=true ! faac! taginject port=3000
location="myfile.txt" ! flvmux ! rtmpsink
location="rtmp://localhost:1935/live/testrtmp"
```

Tramite questa pipeline si invia un feed RTMP al server di streaming alla porta 1935.

taginject

Il plug-in *taginject* [10] consente di iniettare i tag nel flusso audio. Nella sua versione originale gode della sola proprietà *tags*, che consente di inserire la lista dei tag unicamente da riga di comando, al momento dell'avvio della pipeline, e senza dare la possibilità di inserire ulteriori metadati dinamicamente, ovvero durante l'esecuzione del flusso multimediale.

In seguito alle modifiche che verranno apportate al modulo di taginject, saranno aggiunte due nuove proprietà, *port* e *location*, che consentiranno rispettivamente di inviare il tag tramite un client TCP (tipo Telnet) alla porta specificata (se non viene specificata nessuna porta, è di default la 4953), e/o di prelevare il metadato da un file di testo indicato.

faac

L'elemento faac [1], che si basa sulla libreria libfaac, effettua la codifica dell'audio in formato AAC [1].

flvmux

flvmux [11] è un elemento che fa dà multiplexer a diversi flussi in ingresso. In questa tesi questo elemento verrà sfruttato soprattutto per la sua capacità di gestire i metadati in arrivo dall'elemento taginject.

flvmux, nella sua implementazione (consultabile e modificabile dal sorgente gstflvmux.c), riconosce il tipo di flusso proveniente dal sink pad[2].

[1] Advanced Audio Coding (AAC) è un formato di compressione audio lossy. AAC è stato standardizzato da ISO e IEC nell'ambito delle specifiche MPEG-2 e MPEG-4. AAC- Oltre che per lo streaming radio, è il formato audio standard per YouTube, player per dispositivi Apple e console come Playstation e Nintendo. [28]

[2] La funzione gst_flv_mux_handle_sink_event() gestisce i vari flussi in ingresso in uno switch-case, che attiva le funzioni adatte a seconda del tipo del flusso dati (audio, video o metadati).

Il flusso audio/video, viene bufferizzato e incapsulato all'interno di frame FLV, all'interno dei quali saranno presenti oltre i dati audio vero e propri, altri campi utili in seguito a formare il pacchetto RTMP.

Se invece il dato in arrivo sul sink pad è un tag, provvede a creare due pacchetti AMF[3]: il primo contenente nel body la stringa "onMetaData" (la quale, vedremo in seguito, fungerà da ulteriore "header", e costituisce un comando che servirà a preparare il server alla ricezione del metadato), il secondo invece conterrà un array, contenente a sua volta la lista dei tag.

rtmpsink

L'elemento *rtmpsink* [12] è l'ultimo della pipeline. Riceve sul suo sink pad i frame inviati dal src pad di flvmux.

Si appoggia alle funzioni della libreria RTMP, *librtmp*, principalmente per connettersi all'url specificato nella proprietà *location*, tramite la RTMP_Connect(), e dopo invoca la RTMP_Write() la quale provvede a riempire i vari campi del pacchetto RTMP, che saranno dimensionati a seconda del tipo di frame.

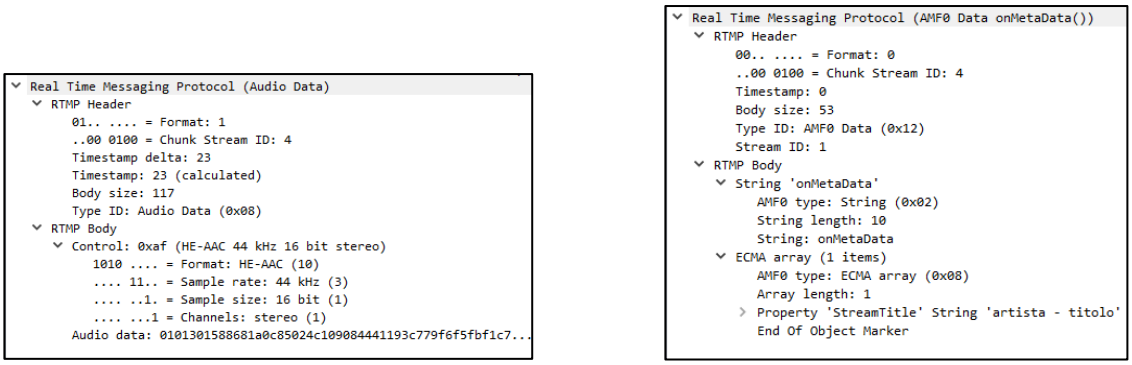

Figura 3 - Catture Wireshark di due pacchetti RTMP a confronto. A sinistra un pacchetto che trasporta dati audio, a destra un pacchetto RTMP che trasporta dati AMF

Nel caso di frame contenente dei metadati, il body del pacchetto RTMP risultante trasporterà al suo interno i due pacchetti AMF creati in precedenza da flvmux, nel seguente ordine: prima quello contenente la stringa "onMetadata", e poi quello contenente l'array con i tag.

I pacchetti RTMP ottenuti saranno infine inviati allo streaming server specificato nella proprietà *location*.

[3] All'interno della funzione gst_flv_mux_create_metadata() vengono inizializzati i vari campi dell'header e il body.

3.2 Il protocollo RTMP

Real-Time Messaging Protocol [13] è un protocollo sviluppato da Macromedia per lo streaming audio, e reso pubblico da Adobe System.

RTMP può essere incapsulato sia all'interno di TCP (porta 1935), sia all'interno di richieste HTTP al fine di superare i firewall. Esiste anche nella versione per connessioni sicure SSL (RTMPS che usa HTTPS) e nella versione cifrata (RTMPE).

Permette di inviare video in formato Flash Video Format (FLV), audio codificato in formato MP3, e può incapsulare nel body anche dei pacchetti Action Message Format (AMF) contenenti istruzioni ActionScript[4].

3.2.1 Formato dei pacchetti

Come possiamo vedere nei precedenti esempi di pacchetti RTMP catturati da Wireshark (Figura 3), un pacchetto RTMP ha un header e un body.

Fra i campi dell'header notiamo il *Body size* che contiene la lunghezza in byte dei dati contenuti nel body, e il campo *Type ID*, che invece contiene un valore che indica se il pacchetto trasporta dati audio (0x08), video (0x09) oppure comandi AMF (0x11, 0x12, 0x14).

3.2.2 Action Message Format

AMF [14] è un formato usato per la serializzazione di istruzioni ActionScript, ma anche per i messaggi scambiati tra il client e il media server per la negoziazione della connessione. Viene solitamente utilizzato insieme a RTMP, ovvero uno o più pacchetti AMF possono costituire il payload del pacchetto RTMP.

Tra i campi più importanti di un pacchetto AMF vi è il campo *Type Marker* il quale contiene un valore esadecimale che indica il tipo di dato contenuto (numero, stringa, booleano, array ecc…).

Nella Figura 3, a destra, vi è un esempio di pacchetto RTMP, con Type ID 0x12, che trasporta a sua volta due pacchetti AMF (rispettivamente con type marker 0x02 e 0x08), il primo contenente una stringa e il secondo contenente un array ECMA[5].

[4] ActionScript è un linguaggio di scripting Adobe Flash, progettato inizialmente per consentire all'utente per gestire i filmati Flash tramite operazioni quali la riproduzione, l'interruzione o saltare da un punto all'altro del filmato. In seguito sono state introdotte anche delle callback per la gestione di eventi, quali la lettura di metadati nel flusso multimediale. [29]

3.3 Wowza Streaming Engine

Wowza Streaming Engine [15] è uno streaming media server sviluppato da Wowza Media System. Il server è usato per lo streaming audio/video live e on-demand su reti IP, per una varietà di dispositivi ad esso collegati, come desktop, laptop, tablet, smartphone, console (videogiochi), e altri dispositivi dotati di connessione alla rete.

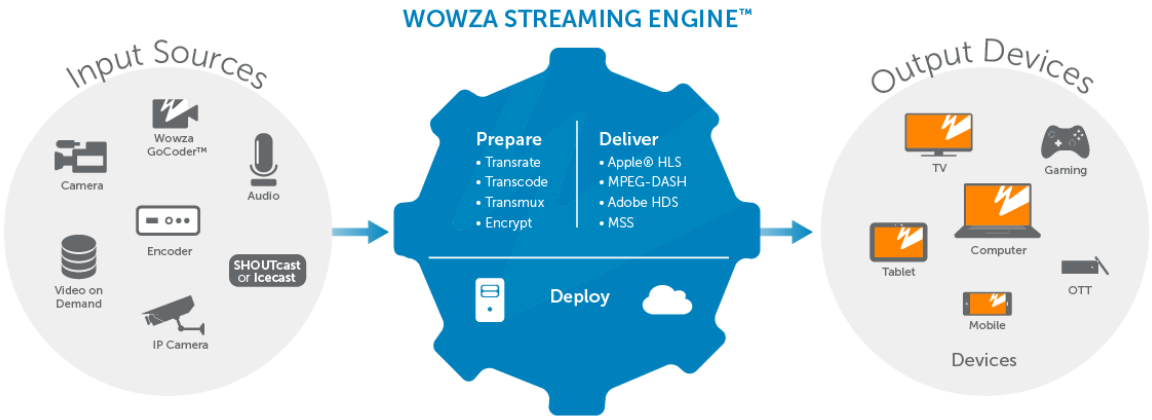

Figura 4 - Schema di funzionamento di Wowza Streaming Engine tratto dal sito ufficiale [16]

Lo streaming di Wowza Streaming Engine è riproducibile da diversi tipi di client per il playback, come i player di Adobe Flash, Microsoft Silverlight ed Apple. È in grado di interfacciarsi con i client ad esso collegato attraverso differenti protocolli di streaming standard, come Apple HLS, HDS, MPEG-DASH, RTSP e Microsoft Silver Light. Inoltre può ricevere flussi audio/video tramite RTP, RTSP, RTMP, MPEG-TS (unicast e multicast) e fare il re-streaming di flussi ICY (SHOUTcast / Icecast)[6]

Ciascun flusso in ingresso può essere gestito in modo adeguato da un'*applicazione*. Un'applicazione [17] di Wowza Streaming Engine è un insieme di opzioni di configurazione, che supporta uno specifico caso d'uso per la gestione e la distribuzione di un determinato contenuto streaming.

[5] ECMA è un linguaggio di programmazione che trova le sue più importanti implementazioni in JavaScript, ActionScript e JScript [36].
[6] ICY è un protocollo sviluppato ad hoc per SHOUTcast, si tratta di una ridefinizione ad hoc del protocollo HTTP. SHOUTcast è una tecnologia per la gestione di dati audio trasmessi da una sorgente a una o più destinazioni su Internet, utilizza i formati audio MP3 o AAC. [30]

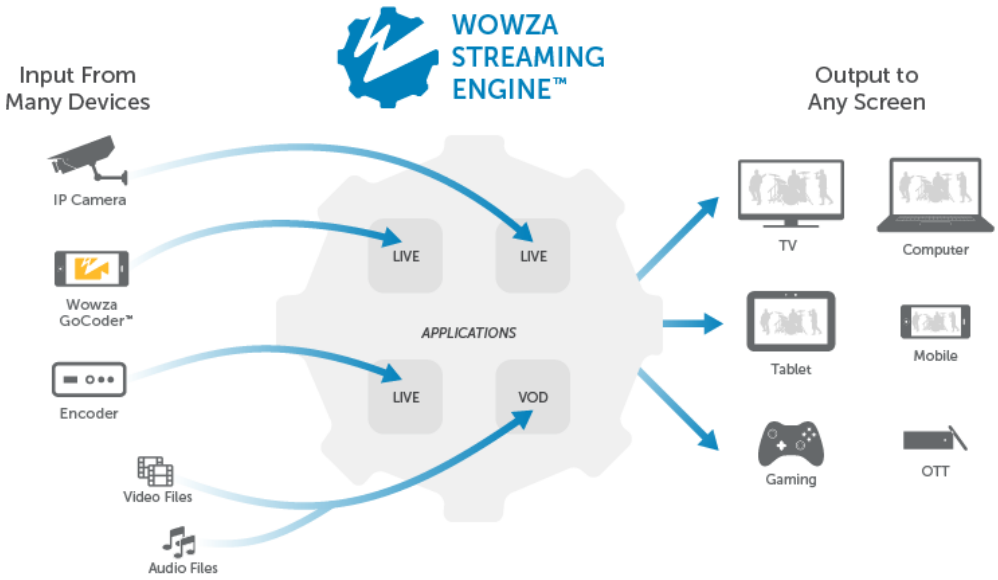

Figura 5 - Workflow tratto dal sito ufficiale Wowza che mostra come vengono gestiti i vari flussi in ingresso

Le funzionalità di un'applicazione possono essere estese da moduli, che sono delle classi Java caricate dinamicamente quando l'applicazione viene avviata.

4 Implementazione su Linux

La prima problematica affrontata è stata quella relativa al download e all'installazione di GStreamer, dei relativi plug-in e delle librerie necessarie al loro funzionamento.

4.1 Download, configurazione ed installazione di Gstreamer

Dal link https://gstreamer.freedesktop.org/src/ si scaricano gli archivi tar relativi ai seguenti package:

- gstreamer-1.9.2
- gst-plugins-base-1.9.2
- gst-plugins-good-1.9.2
- gst-plugins-bad-1.9.2

Si crea una cartella (preferibilmente nella home) col nome "gstreamer", e si scompattano al suo interno gli archivi scaricati.

Per compilare GStreamer è necessario installare i seguenti pacchetti utilizzando preferibilmente Synaptic (alcuni potrebbero essere già installati):

- bison
- flex
- libglib2.0-dev
- g++

Sempre da Synaptic, per far funzionare gli element necessari alle nostre pipeline Gstreamer, bisogna installare i seguenti pacchetti:

- libasound2-dev
- libfaac-dev
- libfaad-dev
- librtmp-dev
- rtmpdump

Dopodiché si compila tutto, accedendo da terminale a ciascuna delle cartelle precedentemente scompattate, nell'ordine gstreamer-1.9.2, gst-plugins-base-1.9.2, gst-plugins-good-1.9.2, gst-plugins-bad-1.9.2, e si eseguono i seguenti comandi:

```
./configure
```

```
make
sudo make install
```

Infine, sempre da terminale, si dà il comando `ldconfig` per aggiornare il file della cache delle librerie a collegamento dinamico.

Al termine di questi passi, è stato possibile testare il tutto tramite il comando su terminale:

```
gst-launch-1.0 audiotestsrc is-live=true ! faac ! aacparse ! faad ! autoaudiosink
```

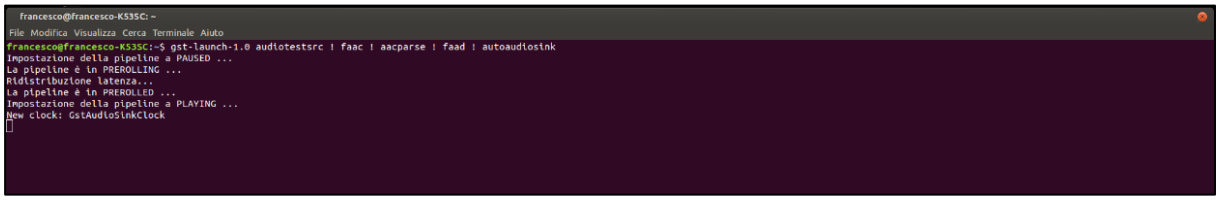

Figura 6 - Screenshot del terminale Linux una volta lanciato il comando gst-launch-1.0 con una pipeline di test

4.2 Configurazione di Wowza Streaming Engine

Una volta scaricato e installato GStreamer con i necessari plug-in, il passo successivo è stato quello di installare e configurare il software Wowza, in modo tale da testare una pipeline GStreamer che invia un feed RTMP al server Wowza stesso.

Nello specifico si è scaricata una prova gratuita del Wowza Streaming Engine, previa registrazione tramite la quale si riceve un codice seriale valido per un mese, seguendo le istruzioni presenti al link seguente https://www.wowza.com/pricing/installer.

Una volta eseguita la procedura di installazione guidata di Wowza, è possibile accedere all'applicazione tramite il link http://localhost:8088/enginemanager, dove si possono configurare il server e le varie applicazioni.

In questo caso si vuole che l'amministratore possa accedere alle proprietà e alle funzioni avanzate, e che non siano previsti meccanismi di autenticazione per i flussi RTMP e RTSP in arrivo.

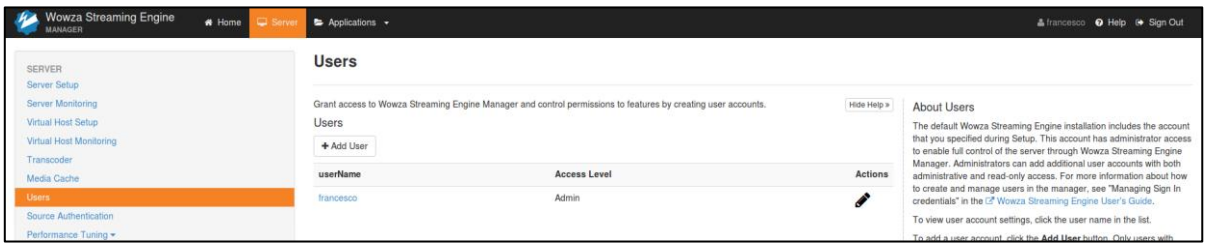

Figura 7 - Configurazione del server Wowza (1)

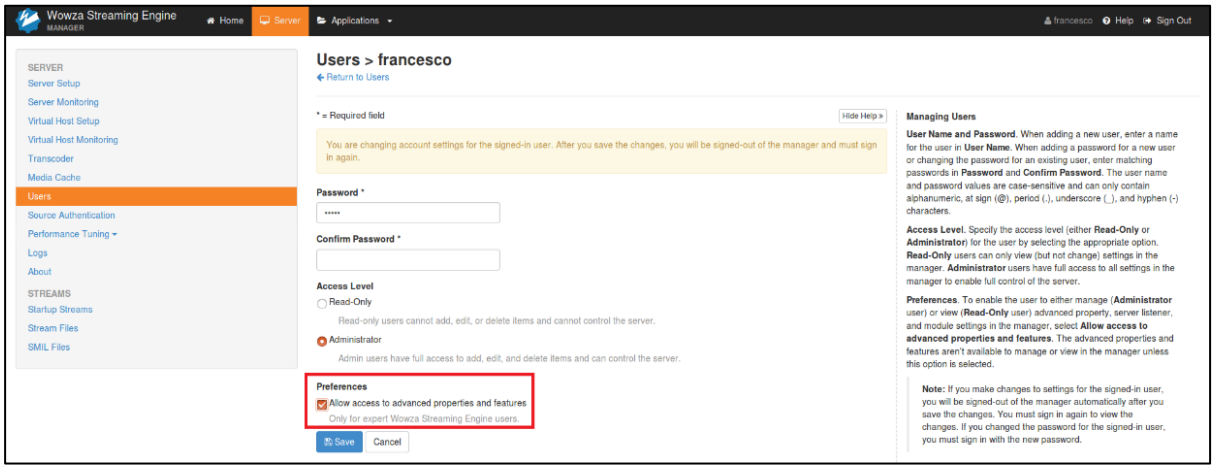

Figura 8 - Configurazione del server Wowza (2)

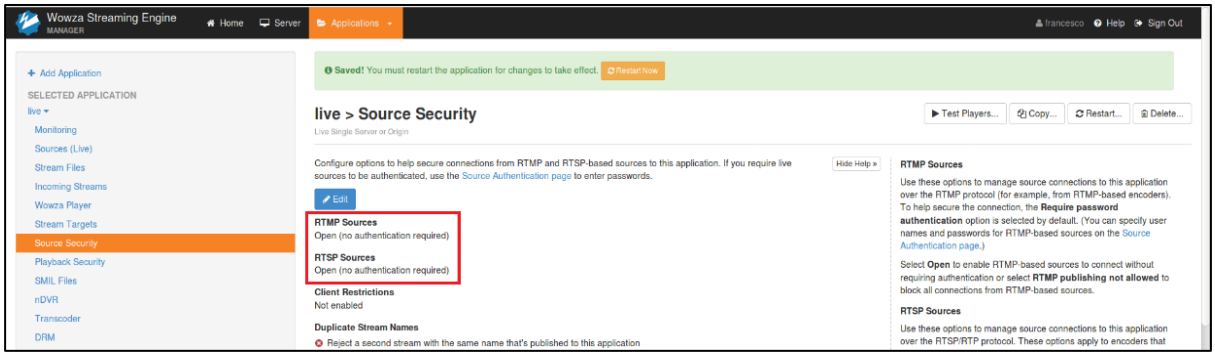

Figura 9 - Configurazione dell'applicazione "live" di Wowza (vengono disabilitati i meccanismi di autenticazione per i flussi RTMP e RTSP)

Terminati questi passaggi, si prova a inviare un feed RTMP con GStreamer all'indirizzo localhost:1935/live/testrtmp, tramite la pipeline:

```
gst-launch-1.0 audiotestsrc ! faac ! flvmux ! rtmpsink
location="rtmp://localhost:1935/live/testrtmp"
```

Figura 10 - Screenshot del terminale dopo aver lanciato una pipeline che manda un feed RTMP al server

Comparirà, se è tutto corretto, uno stream nella pagina "Incoming Streams" dell'applicazione live.

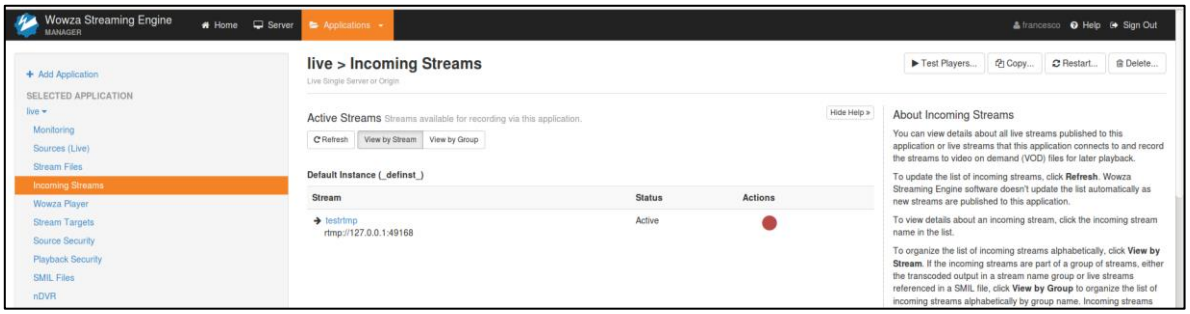

Figura 11 - Streaming RTMP in arrivo sul server Wowza e visibile tra gli Incoming Streams

Inoltre cliccando su "Test Players" si può riprodurre il flusso in arrivo su Wowza da GStreamer.

4.2.1 Aggiunta del modulo java ModuleCupertinoLiveOnTextToID3

Per intercettare i tag inseriti nel flusso live in Wowza, è necessario aggiungere alla nostra applicazione un modulo, ModuleCupertinoLiveOnTextToID3, scritto in codice Java [18]. Una estensione di Eclipse ci permette di integrare tale modulo nella nostra applicazione.

Tramite questo modulo, i metadati contenuti in formato AMF che viaggiano nel flusso live RTMP, vengono intercettati e formattati come tag ID3, per potere essere inviati attraverso un flusso Apple HLS.

Nella funzione onFillChunkDataPacket() preposta al parsing dei pacchetti AMF contenenti i metadati, sono state inserite alcune righe di codice (evidenziate in grassetto nel frammento di codice sottostante) che riportano i tag ricevuti nei file di log di Wowza.

```
public void onFillChunkDataPacket(LiveStreamPacketizerCupertinoChunk chunk,
CupertinoPacketHolder holder, AMFPacket packet, ID3Frames id3Frames)
{
    while(true)
    {
        byte[] buffer = packet.getData();

        if (buffer == null)
                break;
        if (packet.getSize() <= 2)
                break;

        int offset = 0;

        if (buffer[0] == 0)
                offset++;
```

```
            AMFDataList amfList = new AMFDataList(buffer, offset, buffer.length-offset);

            if (amfList.size() <= 1)
                        break;

           if (amfList.get(0).getType() != AMFData.DATA_TYPE_STRING && amfList.get(1).getType() !=
AMFData.DATA_TYPE_OBJECT)
                                    break;

            String metaDataStr = amfList.getString(0);
            AMFDataObj dataObj = amfList.getObject(1);

            if (!metaDataStr.equalsIgnoreCase("onMetaData")) break;

            AMFDataItem textData = (AMFDataItem)dataObj.get("StreamTitle");
            if (textData == null) break;

            String textStr = textData.toString();
            String contextStr = liveStreamPacketizer.getContextStr();

            //…

            WMSLoggerFactory.getInstance().getLoggerObj("metadata").info(CLASSNAME +
".onFillChunkDataPacket["+liveStreamPacketizer.getContextStr()+"] Send metadata string: " +
textStr,"stream", "metadata");

            // Add ID3 tag with text data in chunk based on timecode order
            ID3V2FrameTextInformation comment =
                        new ID3V2FrameTextInformation(ID3V2FrameBase.TAG_TIT2);
            comment.setValue(textStr);
            id3Frames.putFrame(comment);

            break;
      } //fine while(true)
} //fine funzione
```

I passaggi per l'installazione dell'estensione di Eclipse e per l'aggiunta di un modulo in Wowza sono descritti nel seguente link: https://www.wowza.com/docs/how-to-extend-wowza-streaming-engine-using-the-wowza-ide.

Segue uno screenshot dei moduli presenti nell'applicazione "live", al termine della procedura descritta dal link.

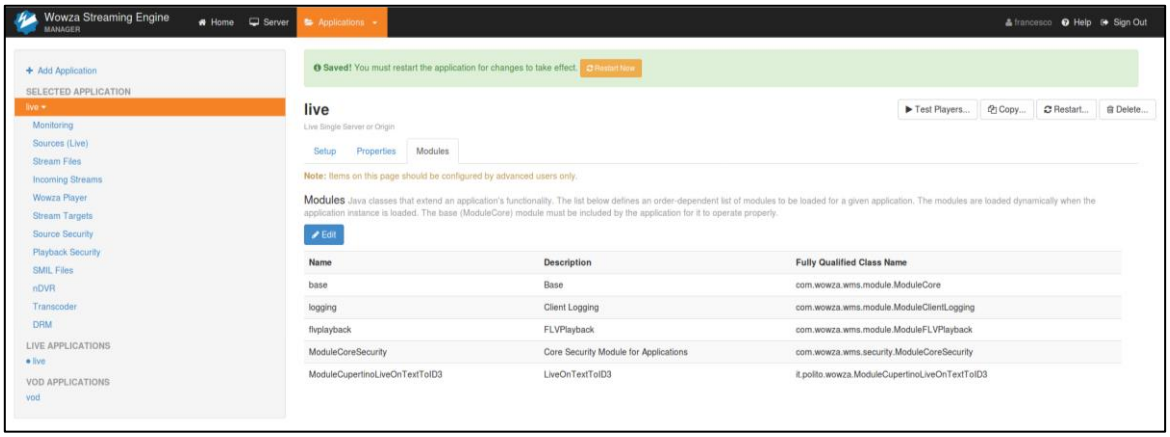

Figura 12 - Lista dei moduli dell'applicazione "live", dopo l'aggiunta del modulo ModuleCupertinoLiveOnTextToID3

4.2.2 Verifica del corretto funzionamento del modulo aggiunto su Wowza

Per testare il corretto funzionamento del modulo aggiunto, si è creata una applicazione che riceve un flusso SHOUTcast che ha già i metadati compatibili con il codice del modulo.

Si segue il procedimento descritto al seguente link https://www.wowza.com/docs/how-to-re-stream-audio-from-shoutcast-icecast, inserendo onair11 come nome dello stream e http://onair11.xdevel.com:9990 nel campo Stream URI.

Per evitare la duplicazione dei messaggi nel file di log, si sono disabilitate alcune forme di streaming abilitate di default da Wowza, tenendo attivate solamente Apple HLS e Adobe RTMP (Figura 13).

Al termine della procedura lo streaming comparirà tra gli Incoming Streams dell'applicazione live (Figura 14).

Al cambio della canzone verrà ricevuto un pacchetto, intercettato dal modulo Wowza, contenente i metadati.

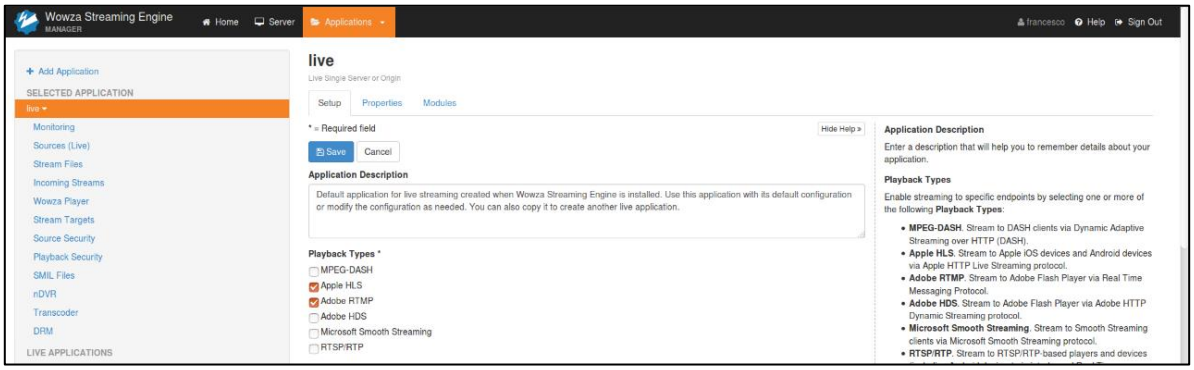

Figura 13 - Configurazione dell'applicazione "live" (vengono abilitati solamente gli streaming HLS e RTMP)

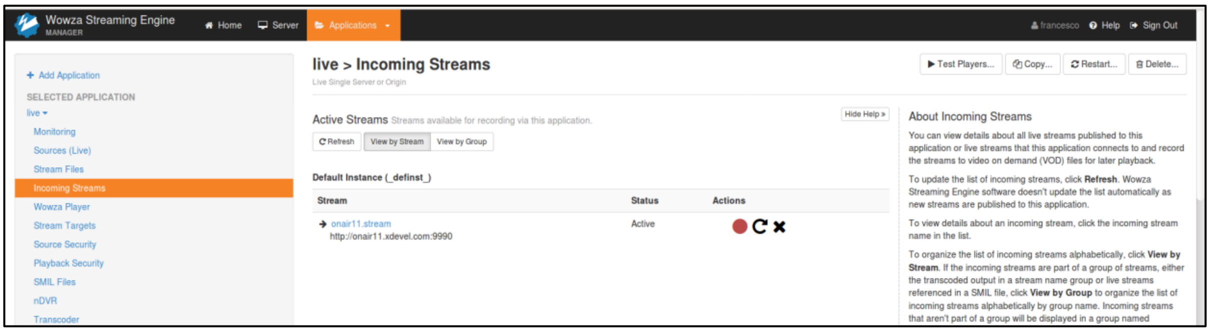

Figura 14- Lo streaming SHOUTcast compare tra gli Incoming Streams in arrivo al server

Di seguito due righe del file di log, presente nella cartella [install-dir]/logs, dove [install-dir] è la cartella nella quale Wowza è stato installato (a sua volta in Linux si trova nella directory usr/local).

- 2017-10-07 13:15:32 CEST **metadata** stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/onair11.stream] **Send metadata string: Dennis-Lloyd - Nevermind** - - - 23.918 -
- 2017-10-07 13:17:52 CEST **metadata** stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/onair11.stream] **Send metadata string: Shawn Mendes - There's Nothing Holdin' Me Back** - - - 163.123 -

A parte, ad esempio, i primi tre campi riguardanti data e timezone, possiamo inoltre notare il valore "metadata" in corrispondenza del campo "event", e il commento contenente il nome del modulo, il nome della funzione, "Send metadata string:", seguito dal titolo della canzone.

4.3 Uso del plug-in taginject per l'inserimento dei tag nello streaming audio

Il passo successivo è stato quello di cercare di inserire il tag nel flusso live tramite il plug-in di GStreamer *taginject*.

Di seguito vediamo un grafico che mostra in grassetto quali saranno i sorgenti di GStreamer coinvolti dalle modifiche.

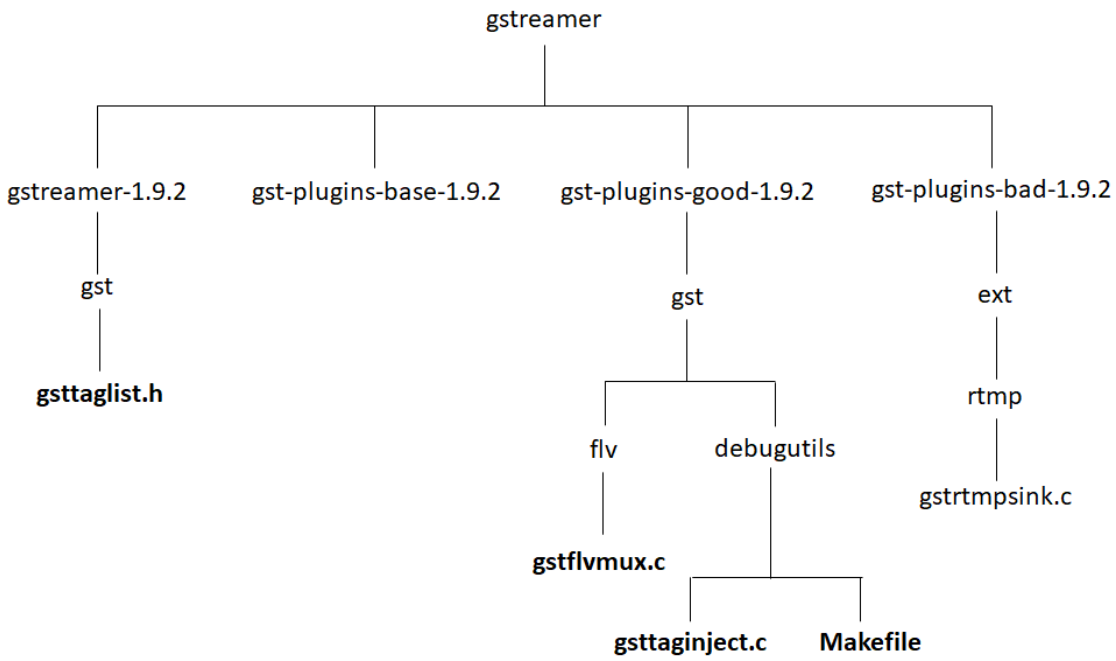

Figura 15 - Sorgenti di GStreamer che saranno coinvolti dalle modifiche. gstrtmpsink.c verrà solamente aggiornato nella data di modifica per includere le modifiche di librtmp.

Di default taginject permette di inserire il metadato relativo al titolo del brano preceduto, tramite la property tags seguita dalla stringa "title=", alla quale a sua volta segue il titolo stesso della canzone [10].

Ad esempio la pipeline GStreamer che si utilizzerebbe in questo caso, è la seguente:

```
$ gst-launch-1.0 audiotestsrc is-live=true ! faac! taginject
tags="title=testsrc,artist=gstreamer" ! flvmux ! rtmpsink
location="rtmp://localhost:1935/live/testrtmp"
```

Una delle problematiche incontrate in seguito a questo approccio consiste nel formato del body, contenente i metadati, generato di default dalla libreria RTMP su cui si appoggia GStreamer per l'inserimento dei tag.

In particolare, mentre il modulo Wowza integrato per intercettare i metadati, si aspetta dei pacchetti RTMP che abbiano come prima stringa del body "onMetaData", invece i pacchetti generati di default da GStreamer, poggiandosi sulla libreria RTMP, presentano nel body un'ulteriore stringa, "@setDataFrame", non riconosciuta dalla funzione del modulo ModuleCupertinoLiveOnTextToID3.

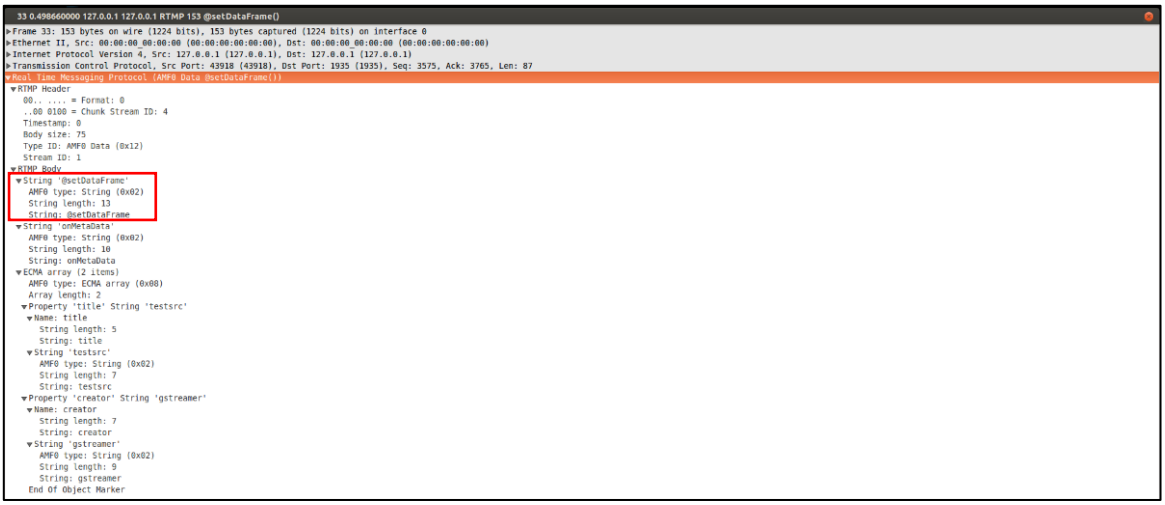

Figura 16 - Screenshot di un pacchetto RTMP, generato da una pipeline GStreamer, contenente i tag inseriti tramite il plug-in taginject. Di default il body contiene a sua volta un pacchetto AMF in più con la stringa @setDataFrame.

In Linux, al fine di risolvere questo problema, ovvero per fare in modo che il body RTMP dei pacchetti che trasportano il metadato inizi direttamente con la stringa "onMetadata", è stato necessario modificare il file rtmp.c della libreria RTMP.

Per la generazione della versione modificata della libreria RTMP su Linux bisogna seguire i passaggi descritti nel paragrafo successivo.

4.3.1 Installazione di RTMPDump e modifica della libreria RTMP

Dal gestore dei pacchetti Synaptic, si installa il pacchetto "rtmpdump" [7]. Dopodiché si installa git (se non è già installato), e si fa il clone (preferibilmente nella cartella Home) di rtmpdump da repository git tramite i seguenti comandi da terminale:

```
$ sudo apt-get install git
$ git clone git://git.ffmpeg.org/rtmpdump
```

È necessario poi installare le seguenti dipendenze, se già non sono state risolte in precedenza, tramite il seguente comando:

```
$ sudo apt-get install build-essential gcc make subversion libssl-dev
```

[7] Software utile per modificare la libreria RTMP, librtmp.

Bisogna scaricare e applicare una patch.

1. Si scarica rtmpdump-2.4.zip dal seguente link: https://github.com/K-S-V/Scripts/releases/download/v2.4/rtmpdump-2.4.zip
2. Si estrae dall'archivio il file Patch.diff e si sposta nella cartella con i sorgenti rtmpdump scaricata nei passaggi precedenti
3. Si applica la patch entrando da terminale nella cartella rtmpdump e digitando il comando
   ```
   $ patch -p0 -i Patch.diff
   ```

Si modifica quindi il file rtmp.c (che si trova all'interno della cartella rtmpdump/librtmp) commentando alcune righe di codice (evidenziate grassetto nel frammento di codice sottostante) presenti all'interno della funzione RTMP_Write().

```
        if (((pkt->m_packetType == RTMP_PACKET_TYPE_AUDIO
              || pkt->m_packetType == RTMP_PACKET_TYPE_VIDEO) &&
           !pkt->m_nTimeStamp) || pkt->m_packetType == RTMP_PACKET_TYPE_INFO)
          {
            pkt->m_headerType = RTMP_PACKET_SIZE_LARGE;
            /*if (pkt->m_packetType == RTMP_PACKET_TYPE_INFO)
                            pkt->m_nBodySize += 16;*/
          }
        else {
            pkt->m_headerType = RTMP_PACKET_SIZE_MEDIUM;
          }

        if (!RTMPPacket_Alloc(pkt, pkt->m_nBodySize))
          {
            RTMP_Log(RTMP_LOGDEBUG, "%s, failed to allocate packet", __FUNCTION__);
            return FALSE;
          }
        enc = pkt->m_body;
        pend = enc + pkt->m_nBodySize;

        /*if (pkt->m_packetType == RTMP_PACKET_TYPE_INFO)
          {
            //enc = AMF_EncodeString(enc, pend, &av_onMetaData);
            pkt->m_nBytesRead = enc - pkt->m_body;
          }*/
```

Ci si posiziona nella cartella rtmpdump/librtmp e si eseguono i seguenti comandi:

```
$ make
$ sudo cp librtmp.so.1 /lib
```

Quindi si compila e si installa la versione modificata della libreria RTMP.

```
$ cd ..
$ make SYS=posix
$ sudo make install
```

Infine bisogna accedere alla directory gst-plugins-bad-1.9.2/ext/rtmp e dare il comando `touch` su uno dei sorgenti presenti nella cartella (ad esempio gstrtmpsink.c), così da impostare la data e ora di ultima modifica del file a quelle correnti, in modo tale che i

successivi comandi `make` e `make install` abbiano come effetto quello di aggiornare la libreria dei plug-in rtmp di GStreamer.

```
$ touch gstrtmpsink.c
$ make
$ sudo make install
```

4.3.2 Modifica di gsttaglist.h e gstflvmux.c

La problematica da affrontare nel passaggio successivo è relativa al fatto che il modulo di Wowza integrato per intercettare i tag, richiede che la stringa che precede il titolo sia "StreamTitle" e non "title" (come di default su GStreamer). Inoltre si vuole evitare che il metadato inserito la prima volta da riga di comando, venga inserito da GStreamer ad ogni frame successivo, come invece avviene di default.

Per risolvere questi problemi, bisogna modificare due moduli di GStreamer:

- Un header appartenente al core stesso di GStreamer, gsttaglist.h, dove sono presenti le macro "define" relative alle stringhe che precedono i tag,
- Un sorgente relativo al plug-in flvmux, dove si vuole che alla stringa che preceda il metadato da mandare venga assegnato il valore dell'identificatore GST_TAG_TITLE definito dalla macro contenuta in gsttaglist.h. Tale file sorgente contiene anche la funzione che viene chiamata ogni volta che viene ricevuto un nuovo frame.

I percorsi dei file sono i seguenti:

- gstreamer-1.9.2/gst/gsttaglist.h
- gst-plugins-good-1.9.2/gst/flv/gstflvmux.c

Modifica di gsttaglist.h

La riga `#define GST_TAG_TITLE "title"` va modificata in `#define GST_TAG_TITLE "StreamTitle"`.

Al termine di questa modifica, affinché vengano coinvolti tutti i moduli di GStreamer che fanno uso di GST_TAG_TITLE, è necessario dare i comandi di `configure`, `make` e `sudo make install` nelle cartelle gstreamer-1.9.2, gst-plugin-base-1.9.2, gst-plugin-good-1.9.2 e gst-plugin-bad-1.9.2 così come è stato fatto nel paragrafo 4.1.

Modifica di gstflvmux.c

Nella funzione gst_flv_mux_create_metadata(), la riga t = "title" deve essere modificata in t = GST_TAG_TITLE.

Inoltre nella funzione gst_flv_mux_handle_buffer (), che viene chiamata ogni volta che il plug-in riceve un frame, deve essere aggiunta la riga in grassetto nel frammento di codice seguente:

```
static GstBuffer * gst_flv_mux_create_metadata (GstFlvMux * mux, gboolean full) {
//…
       t = GST_TAG_TITLE
}

static GstFlowReturn gst_flv_mux_handle_buffer (GstCollectPads * pads, GstCollectData *
cdata, GstBuffer * buffer, gpointer user_data)
{
       //…
       if (mux->new_tags)
       {
              GstBuffer *buf;
              buf = gst_flv_mux_create_metadata (mux, FALSE);

              if (buf)
                     gst_flv_mux_push (mux, buf);

              mux->new_tags = FALSE;
              gst_tag_setter_reset_tags (GST_TAG_SETTER (mux));
       }

       /…

}
```

La chiamata della funzione gst_tag_setter_reset_tags() resetta il tag in modo tale che il metadato venga inserito nel flusso solo quando viene inviato, e non automaticamente ad ogni frame.

Al termine di questa modifica ci si posiziona nella cartella gst-plugins-good-1.9.2/gst/flv, e da terminale si danno i comandi:

```
./configure
make
sudo make install
```

4.3.3 Inserimento del tag con taginject da riga di comando

È possibile a questo punto testare il corretto funzionamento delle modifiche apportate lanciando da terminale la seguente pipeline di GStreamer:

```
$ gst-launch-1.0 audiotestsrc is-live=true ! faac! taginject
tags="StreamTitle=testsrc" ! flvmux ! rtmpsink
location="rtmp://localhost:1935/live/testrtmp"
```

Dopo aver modificato la libreria RTMP, si può notare dalla cattura Wireshark il fatto che adesso i pacchetti generati da GStreamer presentano “onMetadata” come prima stringa del body. Questa modifica ha reso intercettabili i metadati da parte del modulo che è stato precedentemente integrato in Wowza.

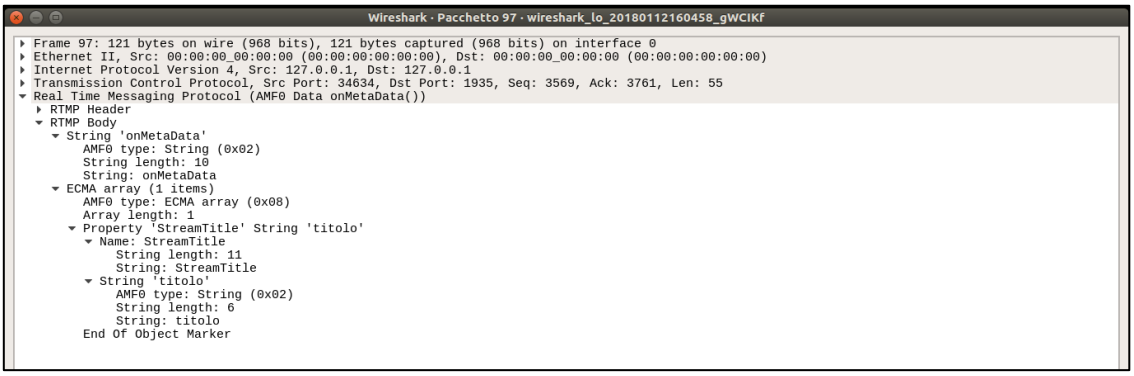

Figura 17- Screenshot di un pacchetto RTMP dopo le modifiche di librtmp e GStreamer

Il tag immesso nello streaming viene correttamente riportato dal file di log di Wowza.

- `2018-01-12 16:08:07 CET metadata stream INFO 200 - - - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/testrtmp] Send metadata string: titolo - - - 637.498 - - - - - - - - - - - - - - - - - -`

4.3.4 Invio di tag tramite un client Telnet

Si vuole a questo punto inviare il metadato nel flusso streaming tramite un client come Telnet. Ne consegue che uno dei plug-in di GStreamer utilizzati nella pipeline dovrà essere modificato in modo tale da avere un server al suo interno in ascolto su una porta che può essere specificata da riga di comando.

Si è deciso di inserire il codice che implementa il server all'interno del plug-in taginject, in modo tale da lasciare invariati gli altri plug-in in quelle che sono le loro principali funzionalità. Ad esempio flvmux dovrà comunque limitarsi a "muxare" i flussi multimediali (audio e/o video) provenienti dalle sorgenti, mentre rtmpsink assolverà esclusivamente al compito di inviare i dati al server streaming tramite il protocollo RTMP.

In quest'ottica si è ritenuto taginject il plug-in ideale a cui demandare l'invio dei metadati, da riga di comando (come visto nei paragrafi precedenti), da un client come Telnet, oppure, come si vedrà più avanti, prelevando la stringa da un file di testo.

Il percorso del file da modificare è:

- gst-plugins-good-1.9.2/gst/debugutils/gsttaginject.c

Modifica di gsttaginject.c

L'idea è quella di modificare taginject in modo tale che venga avviato un server in un thread che venga eseguito in parallelo al plug-in, e di conseguenza in parallelo all'esecuzione dell'intera pipeline di GStreamer.

Il thread viene lanciato all'interno della funzione gst_tag_inject_init() eseguita al momento dell'avvio del plug-in.

```
gst_tag_inject_init (GstTagInject *self)
{
	//…
	my_thread1 = g_thread_new("my_server_thread", my_server_thread, NULL);
	//…
}
```

La funzione eseguita dal thread dapprima crea un socket TCP associato alla porta my_server_port. In particolare all'interno della funzione myTcpServerStartup(): viene creato il socket, viene assegnato un indirizzo locale di rete al socket tramite la funzione *bind*, viene predisposto il socket per ricevere richieste di connessione attraverso la funzione *listen*.

Dopodiché la funzione my_server() provvede a implementare un server TCP che riceve come parametro il file descriptor del socket.

```
my_server_thread(void *param) {
	GSocket* sock;
	sock = myTcpServerStartup(my_server_port);
	my_server(sock);
	return NULL;
}
```

La funzione *my_server*, riceve un puntatore alla struct GSocket e gestisce le richieste di connessione all'interno di un while(1) attraverso la accept. Mentre il while(1) più interno riceve le stringhe inviate dal client collegato.

La g_socket_get_available_bytes() ritorna la quantità di dati in sospeso nel socket senza bloccare. Se vi sono dati in sospeso nel buffer, verranno ricevuti tramite la funzione g_socket_receive() [19].

La stringa, ricevuta da Telnet, che termina con i caratteri "\r\n" di carriage return e new line, verrà troncata alla sua lunghezza effettiva con il carattere terminatore "\0".

Il booleano "changed" impostato a TRUE, segnala che la stringa è stata ricevuta ed è pronta ad essere utilizzata come nuovo tag.

```
static
gint my_server(GSocket* server_socket)
{
	//...
	while(1) {
		Client_socket = g_socket_accept (server_socket, NULL, &err);

		/...
		while(1) {
		  /* read the buffer header */
		  avail = g_socket_get_available_bytes (client_socket);

		  //...
```

```
                if (avail > 0) {
                     rret = g_socket_receive (client_socket, mybuf, 128, NULL, &err);
                            mybuf[avail - strlen("\r\n")] = '\0';
                            changed = TRUE;
                            //...
                     }
                     //...
              }
                //...
                if (!g_socket_close (client_socket, &err)) {
                    GST_DEBUG("Failed to close socket");
                    g_clear_error (&err);
                }
       }
}
```

La g_socket_close() infine provvede a chiudere il socket del client, segnalando eventuali errori. Mentre il server si mette di nuovo in attesa di un nuovo client sulla accept().
Adesso vogliamo che taginject gestisca, tramite una delle sue funzioni, la stringa ricevuta da Telnet. In particolare si va a modificare la funzione gst_tag_inject_transform_ip() [20], che nella versione originale prevedeva la lettura del tag da riga di comando.

```
static
GstFlowReturn gst_tag_inject_transform_ip (GstBaseTransform *trans, GstBuffer *buf)
{
       GstTagInject *self = GST_TAG_INJECT (trans);
       gchar *structure;
       gchar title[128];
       //…
       if(changed) {
              strcpy(title, mybuf);
              structure =  g_strdup_printf ("taglist,%s=\"%s\"", GST_TAG_TITLE, title);
              self->tags = gst_tag_list_new_from_string (structure);

              if (self->tags && !gst_tag_list_is_empty (self->tags)) {
                     gst_pad_push_event (GST_BASE_TRANSFORM_SRC_PAD (trans),
                     gst_event_new_tag (gst_tag_list_ref (self->tags)));
              }
              changed = FALSE;
       }
       //…
       return GST_FLOW_OK;
}
```

Le funzioni dei plug-in di GStreamer che hanno nella signature “trasform_ip”, trasformano in-place (in loco) il buffer da mandare nel flusso della pipeline [20].

GstBaseTransform [21] è una struct utile per quegli elementi, come taginject, che elaborano dati, quindi necessitano in qualche modo di preparare un buffer o una struct, da poter poi inviare al pad source (sorgente) dell’elemento stesso, cosicché tale informazione possa essere ricevuta dall’elemento successivo nella pipeline, tramite la sua interfaccia sink pad. In questo caso l’elemento successivo nella pipeline è flvmux.

Dapprima si effettua un casting della struct trans (di tipo GstBaseTrasform) nel tipo GstTagInject, che è la struct relativa al plug-in e che contiene i campi che vogliamo impostare

prima di passare le informazioni al pad sorgente. Nel nostro caso vogliamo che venga settato il campo tags della struct con la stringa che è stata ricevuta dal server.

"If changed", cioè se il flag changed è impostato a true, a segnalare che una nuova stringa è stata ricevuta, il contenuto mybuf viene riversato in title tramite una strcpy.

La funzione g_strdup_printf() [22] simile alla sprintf() dello standard C formatta la stringa così come se l'aspetta la funzione gst_tag_list_new_from_string() [23], cioè un gchar*, che ha come prefisso "taglist" (in caso contrario ritorna NULL).

La stringa, preceduta da "StreamTitle=" (per i motivi descritti nei paragrafi precedenti), deve essere compresa tra i doppi apici, altrimenti spazi ed eventuali altri caratteri speciali non verrebbero interpretati correttamente e la funzione gst_tag_list_new_from_string() ritornerebbe NULL.

La funzione gst_tag_list_new_from_string() restituisce un puntatore di tipo GstTagList, che viene assegnato al campo tags della struct self.

A questo punto, se self->tags non è NULL e la tag list deserializzata dalla funzione gst_tag_list_new_from_string() non è vuota, la funzione gst_pad_push_event() manda l'evento di "new tag" passato come secondo parametro alla funzione, al pad source specificato come primo parametro.

Infine il boolean changed torna FALSE.

Nella funzione gst_tag_inject_finalize() che garantisce il rilascio delle risorse al termine dell'esecuzione del plug-in, viene effettuata il join del server-thread che ne assicura la corretta terminazione.

```
static void
gst_tag_inject_finalize (GObject * object)
{
	//...
	/**call g_thread_join()**/
	g_thread_join(my_thread1);
	/...
}
```

Per specificare la porta su cui il server deve mettersi in ascolto, è stata aggiunta al plug-in una property, "port", prendendo spunto da un altro plug-in di GStreamer, *tcpserversrc* [24], il quale riceve una stringa da un client (come Telnet) e la invia a al file descriptor specificato.

Nel frammento di codice seguente, la property viene aggiunta nel metodo di inizializzazione del plug-in, similmente a quanto veniva già fatto con la property tags che permette l'introduzione del tag da riga di comando.

Si utilizza la funzione g_object_class_install_property() [25] che prende come parametri: la classe relativa al plug-in taginject al quale si desidera aggiungere la property, un valore intero

che rappresenta l'id della property, e una struct di tipo GParamSpec, costruita tramite la funzione g_param_spec_int().

Il secondo parametro, l'id della property, è un valore che viene settato all'interno del sorgente tramite una variabile enumerativa:

```
enum
{
       PROP_TAGS = 1,
       PROP_PORT = 2
};
```

La funzione g_param_spec_int() che restituisce la struct di tipo GParamSpec da passare come terzo parametro, riceve a sua volta come parametri il nome canonico della property, un nickname del nome, una stringa contenente una breve descrizione della property, e poi dei valori che specificano il valore massimo e il valore minimo che la property può assumere, un valore di default nel caso in cui non venga specificato alcun valore, e un flag.

I parametri della funzione g_param_spec_int() sono stati lasciati invariati rispetto alla property "port" del plug-in tcpserversrc, e prevedono un intero, per la porta, compreso tra 0 e TCP_HIGHEST_PORT (65535), con porta di default, TCP_DEFAULT_PORT, 4953.

```
static void
gst_tag_inject_class_init (GstTagInjectClass * klass)
{
  //...

       g_object_class_install_property (gobject_class, PROP_TAGS,
              g_param_spec_string ("tags", "taglist",
                     "List of tags to inject into the target file",
                     NULL, G_PARAM_WRITABLE | G_PARAM_STATIC_STRINGS));

         g_object_class_install_property (gobject_class, PROP_PORT,
             g_param_spec_int ("port", "Port",
                "The port to listen to (0=random available port)",
                0, TCP_HIGHEST_PORT, TCP_DEFAULT_PORT,
                G_PARAM_READWRITE | G_PARAM_STATIC_STRINGS));

       //...
}
```

La funzione gst_tag_inject_set_property() viene chiamata nel momento in cui viene impostato il valore di una property da riga di comando, e intraprende delle azioni necessarie a gestire il valore inserito.

Uno switch-case seleziona le azioni da intraprendere a seconda dell'id della property ricevuto dalla funzione come secondo parametro, in questo caso PROP_PORT. L'azione intrapresa sarà quella di convertire il GValue ricevuto da riga di comando (terzo parametro della funzione) in un intero, e assegnarlo alla variabile intera my_server_port.

Il caso di default prevede l'inserimento di una property non esistente, e lancia l'errore di "invalid property".

```
static void gst_tag_inject_set_property (GObject * object, guint prop_id,
    const GValue * value, GParamSpec * pspec)
{
switch (prop_id) {
       case PROP_TAGS:

              //...
              break;
       //...

       case PROP_PORT:
              my_server_port = g_value_get_int (value);
       break;

       default:
              G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
       break;
  }
}
```

Modifica del Makefile

Il Makefile presente nella cartella debugutils contenente il sorgente gsttaginject.c, deve essere modificato in modo che il comando "make" generi una nuova versione della libreria di taginject che includa a sua volta la libreria Gnome Input/Output (GIO)[8].

La modifica nel Makefile consiste nell'aggiungere le opzioni in grassetto nelle seguenti due righe:

```
GLIB_LIBS = -pthread -lgobject-2.0 -lgmodule-2.0 -lglib-2.0 -lgio-2.0
GST_BASE_LIBS = -L/usr/local/lib -lgstbase-1.0 -lgstreamer-1.0 -lgobject-2.0 -lglib-2.0 -
lgio-2.0
```

4.3.5 Invio di un tag prelevato da un file di testo

Si vuole che taginject possa prelevare la stringa del tag da un file di testo, specificandone il percorso da riga di comando.

Come nel caso della porta del server, è necessario aggiungere una property che chiameremo "location" tramite la quale il percorso del file possa essere specificato.

[8] GIO fornisce una API che mette a disposizione, tra le altre cose, anche un supporto per la programmazione di rete, che include la risoluzione dei nomi, le API di socket lowlevel e le classi di helper per client e server highlevel. In particolare, sono state utilizzate le funzioni di GSocket della libreria GIO per la realizzazione del server. [31]

Quindi sempre nel sorgente gsttaginject.c si aggiunge un ulteriore valore nell'enumeration, che chiameremo PROP_LOCATION.

```
enum {
       PROP_TAGS = 1,
       PROP_PORT = 2,
       PROP_LOCATION = 3
};
```

Installeremo, analogamente al caso precedente, una nuova property nella funzione gst_tag_inject_class_init().

```
static void gst_tag_inject_class_init (GstTagInjectClass * klass)
{
  //...
         g_object_class_install_property (gobject_class, PROP_PORT,
             g_param_spec_int ("port", "Port",
                "The port to listen to (0=random available port)",
                0, TCP_HIGHEST_PORT, TCP_DEFAULT_PORT,
                G_PARAM_READWRITE | G_PARAM_STATIC_STRINGS));

                g_object_class_install_property (gobject_class, PROP_LOCATION,
             g_param_spec_string ("location", "File Location",
                "Location of the file to read", NULL,
                G_PARAM_READWRITE | G_PARAM_STATIC_STRINGS |
                GST_PARAM_MUTABLE_READY));

       //...
}
```

Bisogna aggiungere poi nello switch-case della funzione gst_tag_inject_set_property(), l'azione che deve essere intrapresa per il valore di "location", dove la stringa ottenuta dal GValue* value tramite la funzione g_value_get_string(), verrà copiata nella variabile di tipo stringa "path" (il percorso del file).

```
static void gst_tag_inject_set_property (GObject * object, guint prop_id,
    const GValue * value, GParamSpec * pspec)
{
  //...
  switch (prop_id) {
       //...

       case PROP_LOCATION:
              strcpy(path, g_value_get_string (value));
              break;
       case PROP_PORT:
              my_server_port = g_value_get_int (value);
       break;
       default:
              G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
       break;
  }
}
```

Analogamente al server-thread, verrà lanciato un thread che controlla, durante l'esecuzione del plug-in e quindi della pipeline di GStreamer, se il file è stato modificato. La funzione

difftime confronta la data di ultima modifica del file con quella precedente, con quest'ultima inizialmente impostata a 0 e aggiornata ogni qual volta il file viene modificato. Verrà letta la prima riga del file di testo, e se la lettura va a buon fine, si procede a copiare la stringa letta nella variabile globale "mybuf" e a settare il flag "changed" a TRUE.

```
my_tag_from_file_thread (void *param) {
    FILE *fp;
    gchar title[128];
    struct stat attr;

    while (1) {
       fp = fopen(path, "r");
       stat(path, &attr);

       if (fp == NULL)
              continue;

       if (difftime(attr.st_mtime,last_modify) > 0) {
              last_modify = attr.st_mtime;

              if (fgets (title , 128 , fp) != NULL) {
                     g_mutex_lock (&data_mutex);
                            strcpy(mybuf, title);
                            mybuf[strlen(mybuf)-1] = '\0';
                            changed = TRUE;
                     g_mutex_unlock (&data_mutex);
               }
       }
       fclose(fp);
    }
    return NULL;
}
```

Il thread verrà lanciato, insieme al server-thread, nella funzione di inizializzazione di taginject e terminerà naturalmente con un join nella funzione di finalizzazione.

```
static void
gst_tag_inject_init (GstTagInject * self)
{
       //...

       /**call g_thread_new()**/
       my_thread1 = g_thread_new("my_server_thread", my_server_thread, NULL);
       my_thread2 = g_thread_new("my_tag_from_file_thread", my_tag_from_file_thread, NULL);
}

static void gst_tag_inject_finalize (GObject * object)
{
       //...

       /**call g_thread_join()**/
       g_thread_join(my_thread1);
       g_thread_join(my_thread2);
}
```

L'accesso alle variabili globali condivise dai due thread è protetto da un g_mutex() in lettura e in scrittura, sia nel thread che legge da file (visto in precedenza), sia all'interno della funzione gst_tag_inject_transform_ip()…

```
g_mutex_lock (&data_mutex);
if(changed) {
	//...
}
g_mutex_unlock (&data_mutex);
```

…sia nel thread del server

```
g_mutex_lock (&data_mutex);
	rret = g_socket_receive (client_socket, mybuf, 128, NULL, &err);
	mybuf[avail - strlen("\r\n")] = '\0';
	changed = TRUE;
g_mutex_unlock (&data_mutex);
```

Compilazione e installazione del plug-in modificato

Dopo aver modificato gsttaginject.c e il Makefile, ci si posiziona nella cartella gst-plugins-good-1.9.2/gst/debugutils, e da terminale si danno i comandi:

```
$ make
$ sudo make install
```

4.3.6 Test della pipeline dopo la modifica del plug-in taginject

A questo punto la nuova versione del plug-in taginject consente di specificare sia la porta (*port*) sul quale il server dovrà mettersi in ascolto, sia il percorso (*location*) dove si trova il file di testo da cui prelevare il titolo.

Segue un esempio in cui verrà specificato come file da cui prelevare il metadato, il file "myfile.txt" nella cartella home, e *3000* sarà invece la porta su cui il server dovrà mettersi in ascolto.

Nel file "myfile.txt" è già presente il testo del primo tag da inviare, "titolo - artista" (Figura 18), e verrà prelevato e mandato nello streaming al momento dell'avvio della pipeline. Il secondo tag "titolo2 – artista2" verrà inviato tramite telnet (Figura 19). Il terzo tag "titolo3 – artista3" viene scritto nel file "myfile.txt" al posto del tag precedente e sarà inviato nel flusso al momento del salvataggio del file modificato (Figura 20). Infine il quarto tag "titolo4 – artista4" verrà di nuovo inviato tramite telnet (Figura 21).

Il corretto invio dei tag nel flusso e la ricezione da parte di Wowza possono essere constatati andando a guardare i log di Wowza, come è già stato fatto nel caso del flusso SHOUTcast.

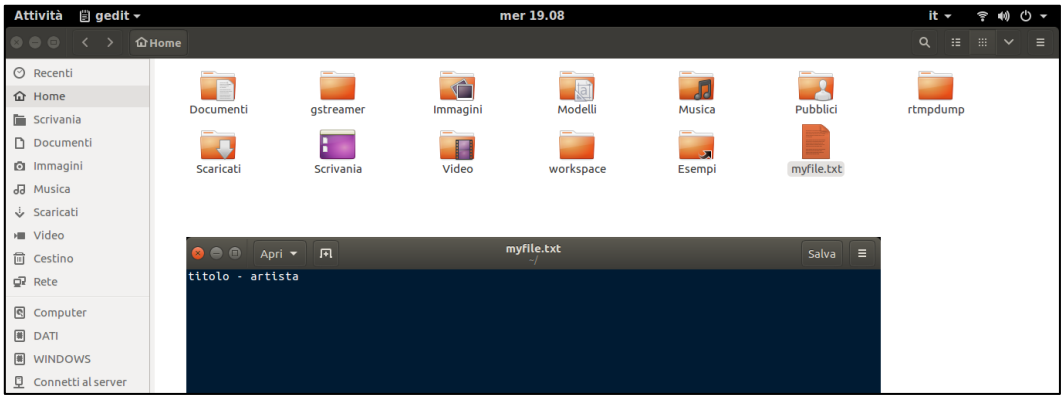

Figura 18 - File "myfile.txt" che viene letto dalla versione modificata di taginject per il prelevamento della stringa una volta salvato il file.

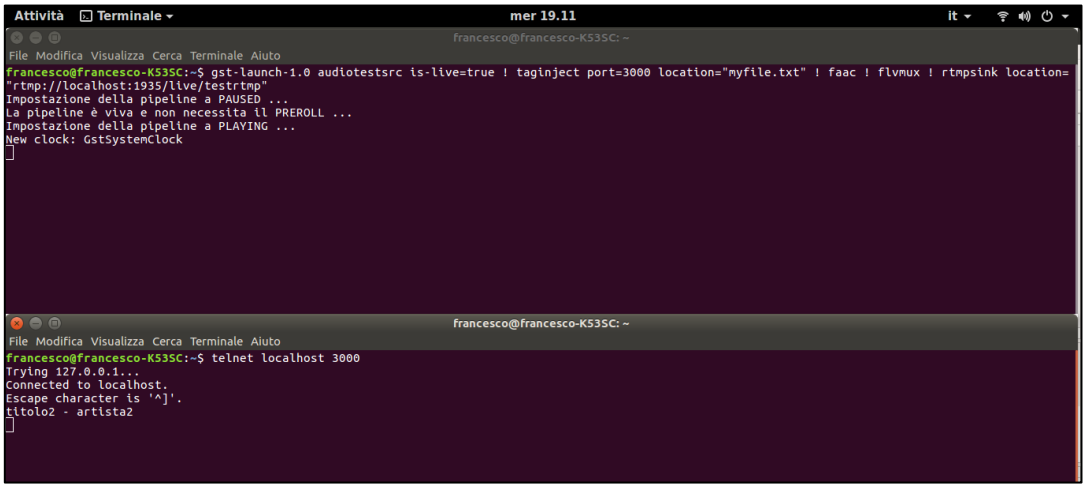

Figura 19- Screenshot di due finestre del terminale.
Dopo aver avviato la pipeline con il server inserito all'interno di taginject, in ascolto sulla porta 3000, e che preleva il tag da "myfile.txt", si invia un altro metadato tramite Telnet al server in ascolto.

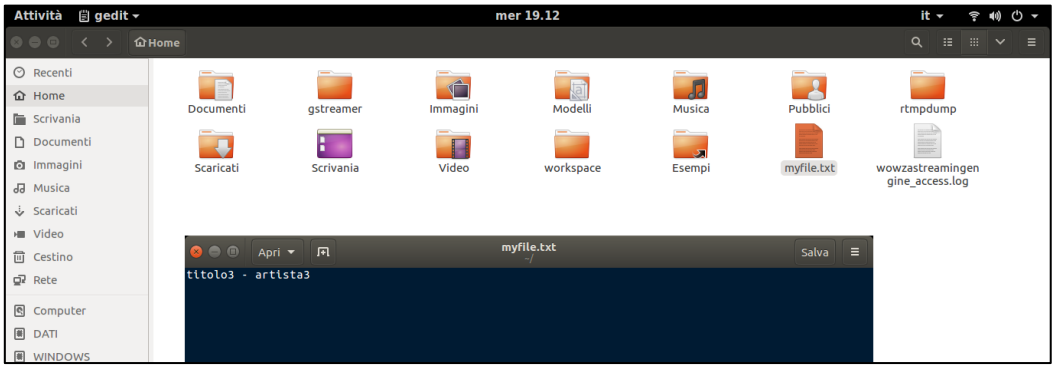

Figura 20 - Screenshot di myfile.txt, che viene modificato, e il salvataggio comporta l'invio di un nuovo tag.

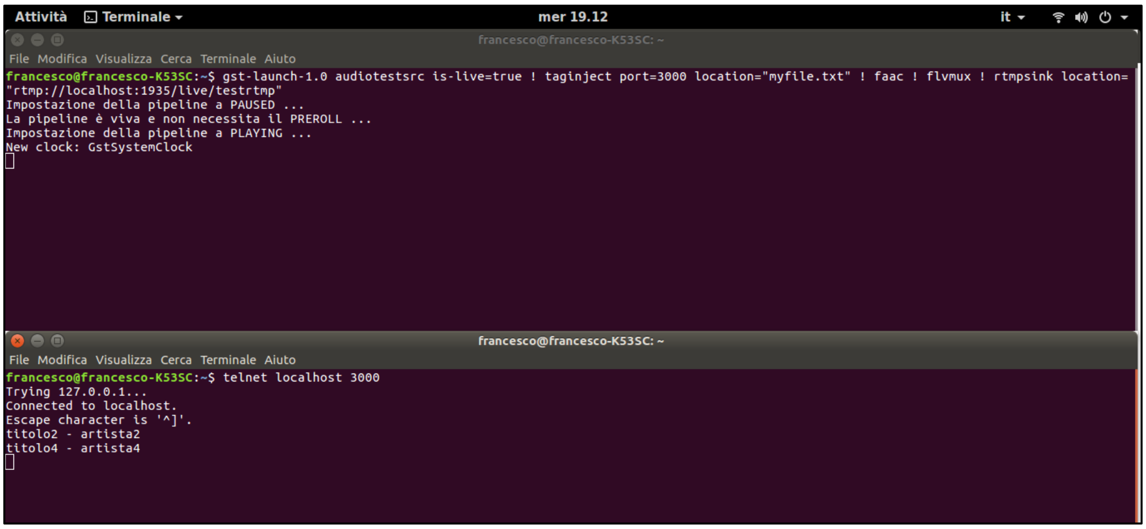

Figura 21 - Viene inviato un'ulteriore tag al server, stavolta tramite Telnet

- `2018-01-10 19:11:30 CET metadata stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/testrtmp] Send metadata string:` **`titolo - artista`** `- - - 390.348 - - - - - - - - - - - - - - - - - - - - - - - - -`
- `2018-01-10 19:11:53 CET metadata stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/testrtmp] Send metadata string:` **`titolo2 - artista2`** `- - - 413.394 - - - - - - - - - - - - - - - - - - - - - - - - -`
- `2018-01-10 19:12:08 CET metadata stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/testrtmp] Send metadata string:` **`titolo3 - artista3`** `- - - 428.091 - - - - - - - - - - - - - - - - - - - - - - - - -`
- `2018-01-10 19:12:35 CET metadata stream INFO 200 - ModuleCupertinoLiveOnTextToID3.onFillChunkDataPacket[live/_definst_/testrtmp] Send metadata string:` **`titolo4 - artista4`** `- - - 455.049 - - - - - - - - - - - - - - - - - - - - - - - - -`

4.4.7 Test con input da scheda audio

Tutti i test descritti fin qui sono stati effettuati usando come audio un “beep” continuo generato dal plug-in di GStreamer *audiotestsrc*.

In realtà, per avvicinarsi quanto più similmente possibile a ciò che avviene nella catena di streaming descritta nel capitolo 2, sono stati condotti anche alcuni test utilizzando come input l'audio catturato dalla scheda audio del pc, tramite il plug-in *alsasrc*[9].

In particolare si sono collegati tramite jack il pc preposto alla codifica e all'immissione dei metadati con un dispositivo che riproduce audio.

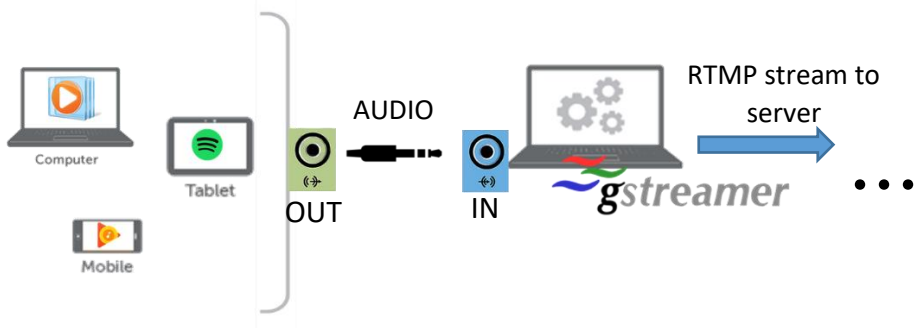

Figura 22 - Schema di configurazione per effettuare i test con audio proveniente da altri dispositivi catturato dalla scheda audio del pc con GStreamer

In una configurazione di questo tipo la pipeline per inviare un flusso audio RTMP, con possibilità di immettere i metadati tramite TCP, su Linux è la seguente:

```
$ gst-launch-1.0 alsasrc ! faac! taginject! flvmux ! rtmpsink
location="rtmp://localhost:1935/live/testrtmp"
```

Su Windows il plug-in equivalente ad alsasrc si chiama *directsoundsrc* basato su DirectSound API di Windows per il driver della scheda audio.

4.4 Autenticazione della sorgente RTMP basata su username e password

In una prima configurazione dell'applicazione live di Wowza, utilizzata per la ricezione e la gestione del flusso RTMP in ingresso, erano stati inizialmente disabilitati i meccanismi di autenticazione della sorgente (Figura 9), al fine di focalizzarsi esclusivamente sulla modifica dei moduli che compongono la pipeline GStreamer.

Vediamo adesso in questa sezione come abilitare i meccanismi di autenticazione su Wowza, e come modificare la pipeline in modo da inviare username e password all'applicazione sul server.

[9] alsasrc è un plug-in che sfrutta l'ALSA API [34]. Si tratta di un software del kernel Linux che fornisce una interfaccia di programmazione al driver della scheda audio. [35]

4.4.1 Configurazione Wowza per abilitare l'autenticazione della sorgente

I passaggi per configurare Wowza sono spiegati in modo completo ed esaustivo sulla pagina https://www.wowza.com/docs/how-to-enable-username-password-authentication-for-rtmp-and-rtsp-publishing.

Per l'esempio seguente sono state usate le credenziali di autenticazione "francesco" e "passwd" rispettivamente come username e password.

Al termine avremo le seguenti impostazioni:

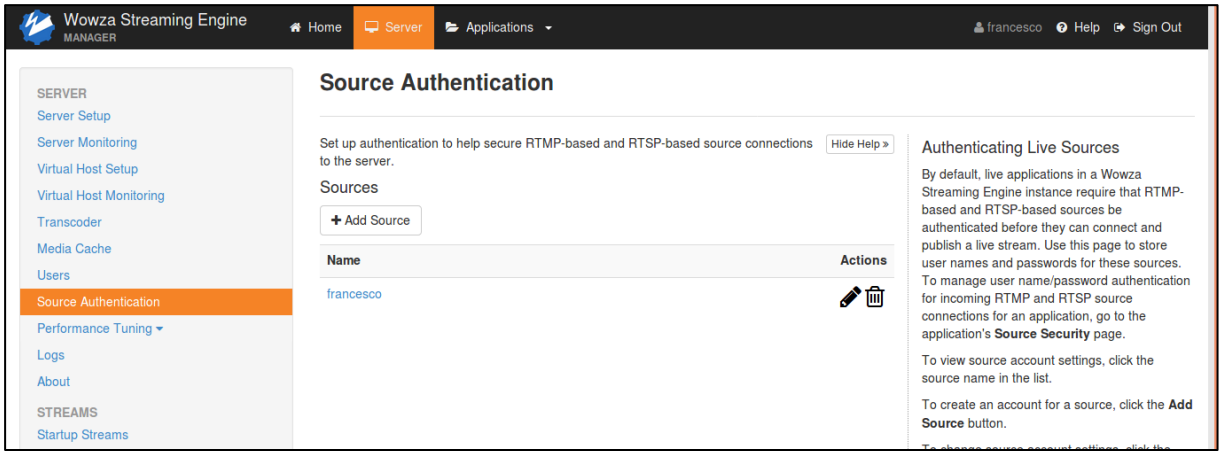

Figura 23 - Credenziali configurate sul server Wowza (username: francesco, password: passwd)

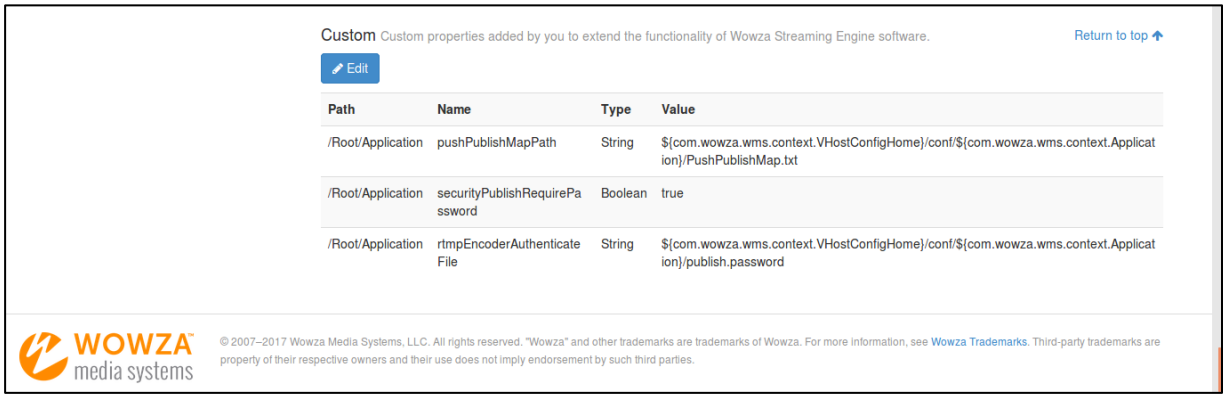

Figura 24 - Moduli aggiunti all'applicazione live di Wowza (scheda Properties)

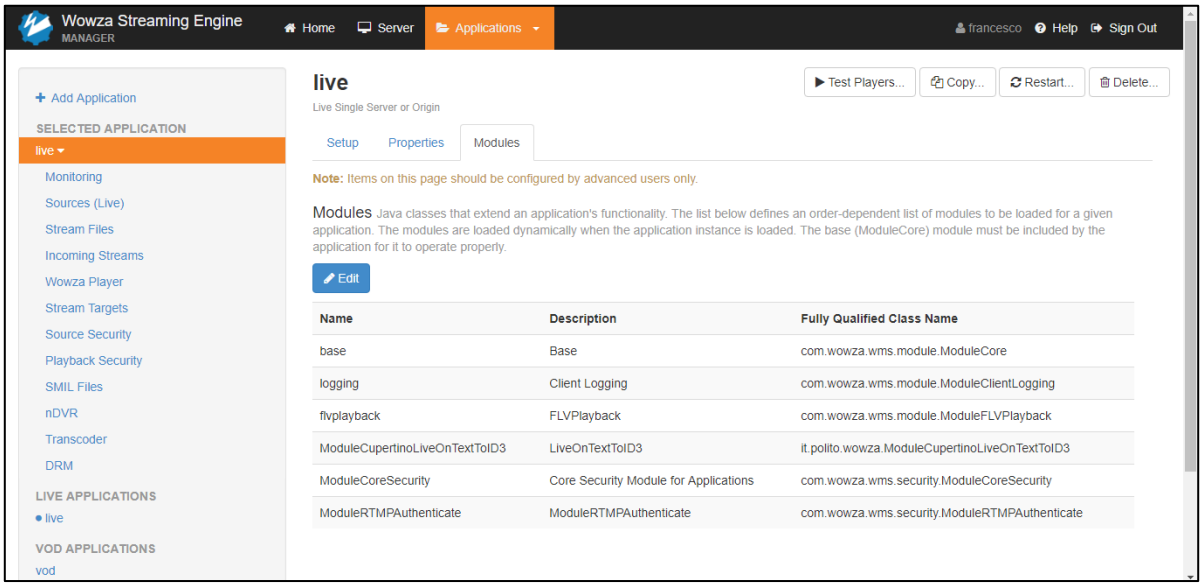

Figura 25 - Moduli per l'autenticazione aggiunti all'applicazione live (scheda Modules)

4.4.2 Avvio della pipeline GStreamer con i parametri di autenticazione

L'unica modifica che bisogna apportare alla pipeline utilizzata per effettuare lo streaming RTMP, affinché vengano inviate al server anche le credenziali per l'autenticazione, consiste nell'introduzione di alcuni parametri dopo l'url nella property "location" di rtmpsink. Ad esempio:

```
gst-launch-1.0 audiotestsrc is-live=true !  faac! taginject ! flvmux ! rtmpsink
location='rtmp://localhost:1935/live/testrtmp pubUser=francesco pubPasswd=passwd
flashver=FMLE/3.0(compatible;FMSc/1.0)'
```

Nello specifico, i parametri che vanno aggiunti sono "pubUser" e "pubPasswd" seguiti rispettivamente da username e password impostati in precedenza nella pagina Source Authentication di Wowza, inoltre va aggiunto anche il parametro "flashver" contenente la versione corrente del Flash Media Encoder.

Nel primo dei due screenshot seguenti vediamo cosa succede se si lancia da terminale una pipeline con password errata ("pass" anziché "passwd"), mentre nel secondo verrà lanciata una pipeline con parametri e credenziali corrette.

Figura 26 - Pipeline con password errata. Viene lanciato un errore, e la connessione fallisce.

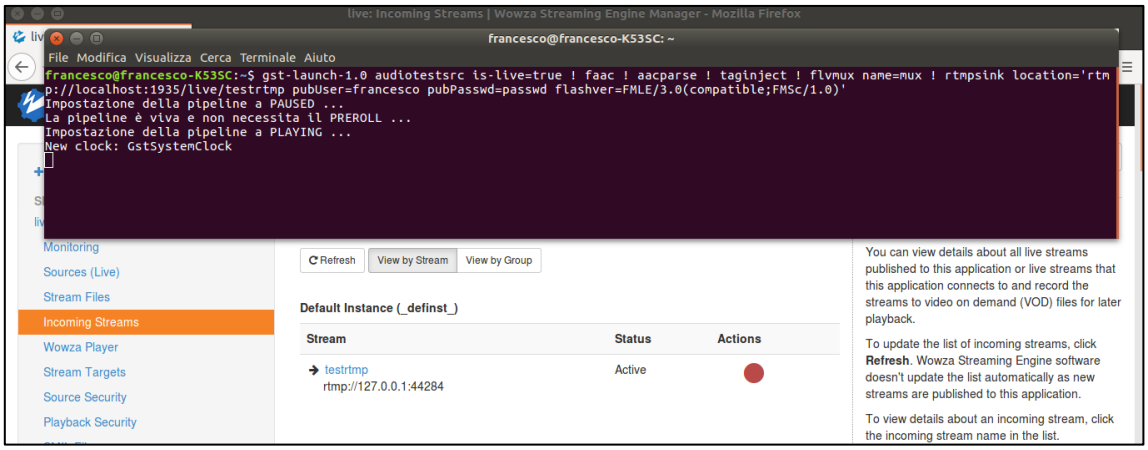

Figura 27 - Pipeline con credenziali corrette. Il flusso arriva a Wowza.

4.3.3 Qualche cenno sul meccanismo di autenticazione della sorgente RTMP

La procedura di autenticazione di una sorgente RTMP si basa su un *meccanismo di autenticazione a sfida*, che consta dei seguenti passaggi.

1. Il client prova a connettersi la prima volta all'url:
   - rtmp://localhost:1935/live/testrtmp
2. Il server inizialmente rifiuta la connessione con un messaggio, che contiene il *rejection code* e una stringa di *description*
   - Rejection code: NetConnection.Connect.Rejected
   - [AccessManager.Reject] : [code=403 **need auth; authmod=adobe**]
3. Il client prova a ricollegarsi all'url modificato con l'authmod e lo username richiesti dal server nel messaggio precedente
   - rtmp://localhost:1935/live?**authmod=adobe&user=francesco**
4. Il server rigetta nuovamente la richiesta di connessione del client con un messaggio contenente una sfida:

- [AccessManager.Reject]:[authmod=adobe]:?reason=needauth&user=francesco **&salt=2PaObA==&challenge=D7djUA==&opaque=D7djUA==**

5. Il client prova a ricollegarsi aggiungendo all'url la soluzione alla sfida (response):
   - rtmp://localhost:1935/live?authmod=adobe&user=francesco&**challenge=jFjdXg==&response=pjVqNoV1dPjNjrKZ8cpo0w==&opaque=D7djUA==**
6. Il server, ricevuta la soluzione, effettua lo stesso calcolo, avendo nel proprio database la tupla username-salt, ed essendogli stata inviata la challenge dal client. Se le due soluzioni coincidono, allora manderà un messaggio contenente 'NetConnection.Connect.Success', lasciando il client libero di poter effettuare il publish dello stream.

Nei passaggi 4 e 5 challenge e opaque sono dei "nonce random", ovvero dei token di sessione utilizzati per evitare il replay attack [10].

In particolare nel passaggio 5, il client calcola il response come segue[11]:

```
response = MD5BASE64[12](user + salt + password);
response += opaque;
response = MD5BASE64(response + challenge[13]);
```

Lo stesso verrà fatto lato-server con il "challenge" inviatogli dal server.

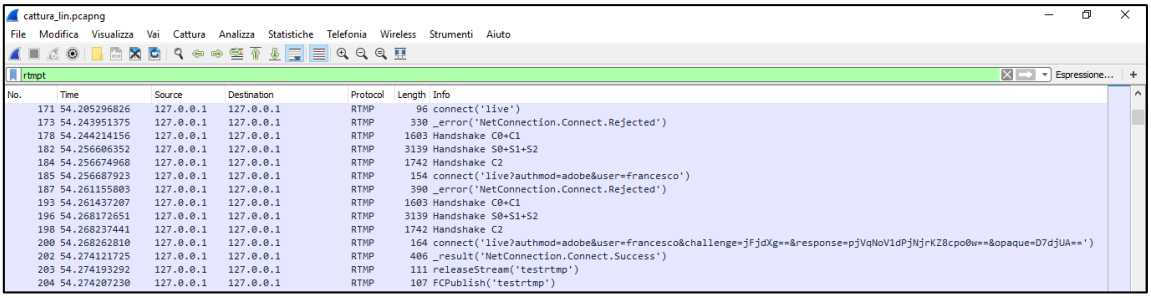

Figura 28 - Cattura Wireshark dei messaggi scambiati da client-server per l'autenticazione della sorgente RTMP

I messaggi per l'autenticazione sono contenuti all'interno di pacchetti RTMP con Type ID=0x14 cioè "AMF0 command", quindi in formato AMF.

[10] Pur senza conoscere la password, un utente malevolo potrebbe "sniffare" direttamente la soluzione alla sfida, e mandarla al prossimo tentativo di accesso. Ma la soluzione "sniffata" non risulterebbe più valida se client e server la calcolassero scambiandosi dei token generati freschi di volta in volta in maniera pseudo-casuale.
[11] Si tratta di pseudo-codice dedotto dalla funzione PublisherAuth() di librtmp.
[12] MD5 è una funzione di hash crittografica.
[13] La "challenge" generata dal client stesso (jFjdXg), e inviata al server nell'url per permettergli di rifare il calcolo (passaggio 5)

5 Implementazione su Windows

Per quanto riguarda la modifica dei moduli di GStreamer su Windows, vedremo come risulti fondamentale l'utilizzo del software MinGW e del sistema "Cerbero".

MinGW [26] è un software che oltre a fare il porting di GCC in ambiente Windows, comprende anche una shell Bourne per Windows chiamata MSYS, da usarsi come alternativa al classico CMD.exe

Cerbero [27] è un framework, reperibile su Git, che rende GStreamer multipiattaforma, ovvero va a ricreare su sistema operativo non Unix-like, tutto quel sottosistema di librerie, moduli sorgenti e package necessari al bulding di GStreamer. Fa uso di particolari script ".recipe" scritti in linguaggio Phyton, che servono a reperire i package online e dare l'avvio ai comandi necessari per il build di GStreamer e dei suoi plug-in.

I seguenti passaggi descrivono come installare GStreamer Editing Services[14] su Windows versione 8.1 (o superiore) a 64 bit.

5.1 Download e installazione del software necessario

È consigliato un editor di testo come Notepad++ per la modifica dei sorgenti, o comunque un editor con funzioni più avanzate rispetto a quello presente di base in Windows che permetta di salvare i file di testo in formato UNIX.

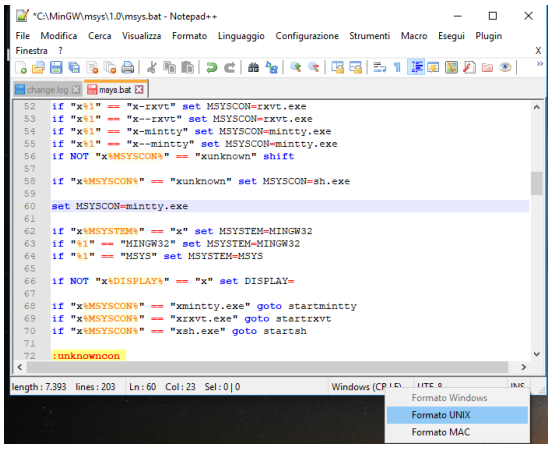

Figura 29 - Esempio di come salvare i file in formato UNIX

[14] "GStreamer Editing Services" è una libreria che rende più semplice la creazione e la modifica di applicazioni multimediali. È basata sul framework GStreamer e ne permette l'utilizzo multi-piattaforma sulla maggior parte dei sistemi UNIX-like, ma anche su Windows. [33]

Per prima cosa si installano i seguenti programmi necessari al download dai repository di Git e alla compilazione dei vari sorgenti che costituiscono GStreamer.

Git

La versione consigliata di Git da installare è a 64 bit, preferibilmente l'ultima disponibile sul sito *Git for Windows*. La directory consigliata nella quale installare git è C:\git. Tra le opzioni di installazione si seleziona "Checkout as-is, Commit as-is".

Python

La versione consigliata di Python da installare è la 2.7.8. Scaricare e installare l'installer MSI per sistemi x86 a 64 bit. Si consiglia di installare Python all'interno della cartella C:\Python27. Tra le opzioni di installazione scegliere di aggiungere Python.exe alle variabili d'ambiene ("Add Python.exe to Path") e "For all users".

CMake

La versione consigliata di CMake è la 3.7.2, da installare attraverso l'installer "win32", nella cartella C:\CMake Anche qui si aggiunge CMake alle variabili d'ambiente ("Add CMake to the System Path for all users").

MinGW

Si scarica l'installer di MinGW. Una volta avviato, in "Basic Setup", si sceglie C:\MinGW come directory di installazione, e si selezionano i seguenti package:

- mingw32-base
- mingw32-gcc-g++
- msys-base
- mingw-developer-toolkit

Si applicano le modifiche cliccando su Installation → Apply changes. Appena completata questa fase, è possibile chiudere la finestra di dialogo.

Si scarica YASM versione 1.3.0-win64, si rinomini l'eseguibile in yams.exe, e si copi quest'ultimo nella cartella C:\MinGW\bin.

Riepilogo dei file da scaricare

- Git, consigliata l'ultima versione dal sito ufficiale "Git for Windows"
- Python 2.7.8
- CMake 3.7.2-win32
- YASM 1.3.0-win64

- MinGW, con i seguenti package: mingw32-base, mingw32-gcc-g++, msys-base, mingw-developer-toolkit

5.2 Configurare la shell MSYS

È consigliabile l'uso di MSYS, una Bourne shell per Windows, più semplice e meno problematica da utilizzare rispetto al classico CMD.exe.

Questa shell si trova già nel pacchetto MinGW, il percorso è:

- C:\MinGW\msys\1.0\msys.bat (si consiglia di crearne un collegamento sul desktop).

È necessario rinominare "fstab.sample" in "fstab" nella cartella C:\MinGW\msys\1.0\etc\, in modo da poter utilizzare il packer manager mingw-get (nella cartella è già presente un file fstab, che deve essere spostato o rinominato prima di poter rinominare fstab.sample).

È consigliabile inoltre installare, tramite mingw-get, MinTTY ossia un terminale virtuale per MSYS che rende più facile il copia&incolla ed è, in generale, maggiormente personalizzabile. Per fare ciò, si apre msys.bat e si digita il comando

```
$ mingw-get install mintty
```

Dopodiché bisogna impostare MinTTY come shell di default, modificando il file msys.bat C:\MinGW\msys\1.0\mys.bat: dopo la linea 58 si aggiunge (Figura 29).

- set MSYSCON=mintty.exe

Alla riapertura di msys.bat, verrà aperta la nuova shell.

Bisogna inoltre aggiungere al file C:\MinGW\msys\1.0\etc\profile, le seguenti righe, in modo da impostare le variabili di ambiente necessarie.

```
export PATH=$PATH:/c/Python27
export PATH=$PATH:/c/Git/bin
export PATH=$PATH:/home/<NomeUtente>/cerbero/
export PATH=$PATH:/c/CMake/bin
export PATH=$PATH:/home/<NomeUtente>/cerbero/build/mingw/w64/bin
export PATH=$PATH:/home/<NomeUtente>/cerbero/build/dist/windows_x86_64/bin

alias cerbero=cerbero-uninstalled
```

Dove <*NomeUtente*> va sostituito con il proprio user name sul sistema.

5.3 Build di Gstreamer

A questo punto si vuole poter modificare e compilare i sorgenti di Gstreamer che si utilizzeranno nella pipeline. A tal proposito saranno propedeutici i seguenti passaggi:

1. Scaricare i tool utili per l'installazione della versione di Gstreamer desiderata (in questo caso la versione 1.9.2) dal git repository.
2. Installare le dipendenze necessarie al bootstrap di cerbero
3. Bootstrap di cerbero
4. Download di faac.recipe e build di faac
5. Modifica dei recipe zlib e gst-plugins-bad-1.0.recipes
6. Build di Gstreamer e dei plug-in base, good e bad

1. Download di Gstreamer dal git repository

Digitare nella shell il commando

```
$ git clone -b 1.9.2 https://github.com/GStreamer/cerbero
```

Al percorso C:\MinGW\msys\1.0\home\<NomeUtente>, troveremo la cartella "cerbero" che contiene tutto il necessario per il build di Gstreamer, tra cui i file.recipe che sono degli script utili per recapitare i pacchetti online (tipicamente archivi in formato zip o tar) contenenti i sorgenti, ed eseguire i comandi necessari alla scompattazione degli archivi scaricati, e alla compilazione dei sorgenti al loro interno.

2. Installazione delle dipendenze propedeutiche al bootstrap di cerbero

Digitare nella shell il comando

```
$ mingw-get.exe install msys-perl msys-patch msys-bison msys-flex
msys-coreutils
```

3. Bootstrap di cerbero

Digitare nella shell il comando

```
$ cerbero bootstrap
```

Talvolta durante il bootstrap può capitare che la procedura si blocchi sulla riga:

```
checking for msgmerge... /usr/home/<NomeUtente>/cerbero/build/build-
tools/bin/msgmerge
```

In questo caso bisogna chiudere la shell msys e assicurarsi, tramite lo strumento "Gestione Attività" di Windows, che non ci siano in esecuzione altri processi sh.exe, e in tal caso individuarli e cliccare su "Termina attività". Dopodiché si riapre la shell e si dà nuovamente il comando di bootstrap. Se dopo questi passaggi il problema si ripresenta, allora si riavvia il pc.

Ridando il comando di bootstrap, la procedura riprenderà direttamente dall'operazione dove si era interrotta.

4. Download di faac.recipe e build di faac

Si scarica faac.recipe dal link

- https://github.com/fluendo/cerbero/blob/master/recipes/faac.recipe

nella cartella C:\MinGW\msys\1.0\home\NomeUtente\cerbero\recipes.

E poi si digita nella shell il comando

```
$ cerbero build faac
```

5. Modifica dei recipe zlib e gst-plugins-bad-1.0.recipes

Occorre modificare il file gst-plugins-bad-1.0.recipe presente nella cartella: C:\MinGW\msys\1.0\home\NomeUtente\cerbero\recipes.

In particolare si cancella "--disable-faac", si aggiunge "--disable-webrtc-audio-processing" all'elenco "configure_options", e si cancella "webrtc-audio-processing" da "deps".

Inoltre l'url per la zlib non risulta più funzionante [15], bisogna quindi modificare il file zlib.recipe presente nella cartella

- C:\MinGW\msys\1.0\home\NomeUtente\cerbero\recipes

sostituendo il vecchio url con il seguente:

- https://sourceforge.net/projects/libpng/files/zlib/1.2.8/zlib-1.2.8.tar.xz.

6. Build di Gstreamer e dei plug-in base, good e bad

Digitare, uno alla volta nella shell i seguent comandi:

```
$ cerbero build gstreamer-1.0
$ cerbero build gst-plugins-base-1.0
$ cerbero build gst-plugins-good-1.0
$ cerbero build gst-plugins-bad-1.0
```

[15] Potrebbe presentarsi lo stesso problema con altri plug-in e librerie. In tal caso si esegua lo stesso procedimento, cercando online un url funzionante, e si riavvii il bootstrap/build del package.

5.4 Modifica dei sorgenti di RTMP e GStreamer

Le modifiche da apportare ai sorgenti GStreamer sono simili, e in alcune casi identiche, a quelle viste nel caso di Linux. Infatti Cerbero, scaricato e *bootstrapato* nei passaggi precedenti, è un sistema per la compilazione multipiattaforma, che raccoglie al suo interno tutti gli strumenti necessari per compilare su Windows dei sorgenti che si appoggiano a librerie open source di Linux, come GIO ad esempio.

Si procederà dapprima a modificare la libreria librtmp per fare in modo che non venga inserita, nei pacchetti RTMP contenenti il metadato, la stringa "*@setDataFrame"* prevista originariamente dalla libreria RTMP, e far precedere il tag direttamente dalla stringa "onMetadata", così come se lo aspetta il modulo del server Wowza preposto ad intercettare i metadati. In questo caso i passaggi differiscono da quelli svolti su Linux, e verranno spiegati nella sezione 4.3.1.

Le modifiche apportate a gsttaglist.h e gstflvmux.c su Windows sono identiche a quelle apportate ai due file nel caso di GStreamer in Linux. In questo caso però al momento della compilazione si sono riscontrati alcuni errori, che possono essere risolti cancellando (o commentando) le righe dei file gstsystemclock.c e gstutils.c che generano errore (sezione 4.3.2).

Infine anche il sorgente gsttaginject.c e il Makefile, possono subire le stesse modifiche effettuate su Linux (sezione 4.3.3) ottenendo così, anche su Windows, una pipeline Gstreamer che permette l'immissione del tag inviato da un client come Telnet o prelevato da un file di testo.

5.3.1 Modifica di librtmp

1. Collocarsi nella seguente cartella C:\MinGW\msys\1.0\home\<NomeUtente>\cerbero\build\sources\local\librtmp-2.4_p20131018
2. Scompattare il file tar al suo interno (bisogna scompattare due volte, infatti la cartella si trova all'interno di due tar annidati)

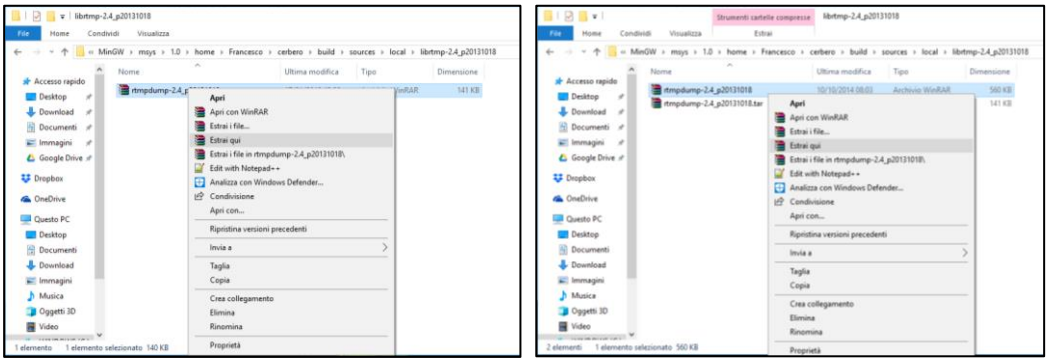

Figura 30 - Scompattazione del tar esterno (1) e del tar annidato (2)

3. Modificare rtmp.c così come descritto nella sezione 3.3.1. rtmp.c si trova dentro la cartella rtmpdump-2.4_p20131018\librtmp
4. Dopo aver eliminato gli archivi tar scompattati nei punti precedenti, ricompattare di nuovo la cartella rtmpdump-2.4_p20131018 in un archivio tar.bz2 (usando il software 7z).

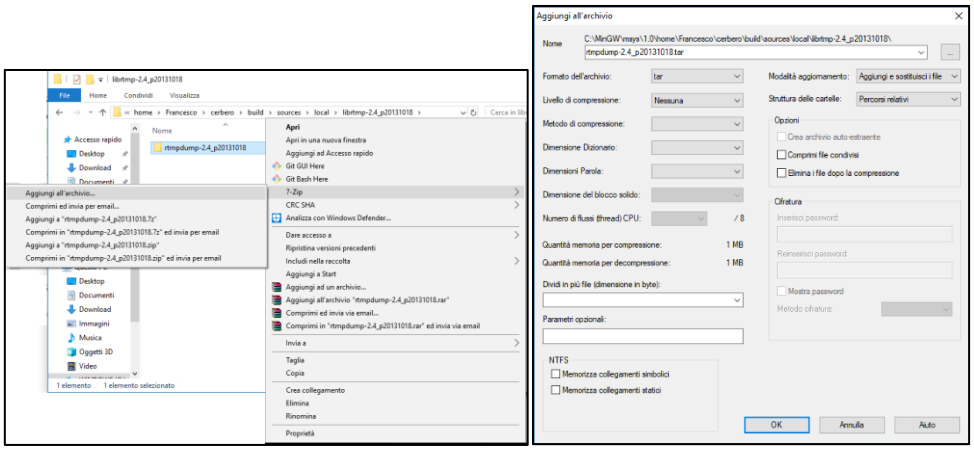

Figura 31- Si comprime la cartella in un archivio tar

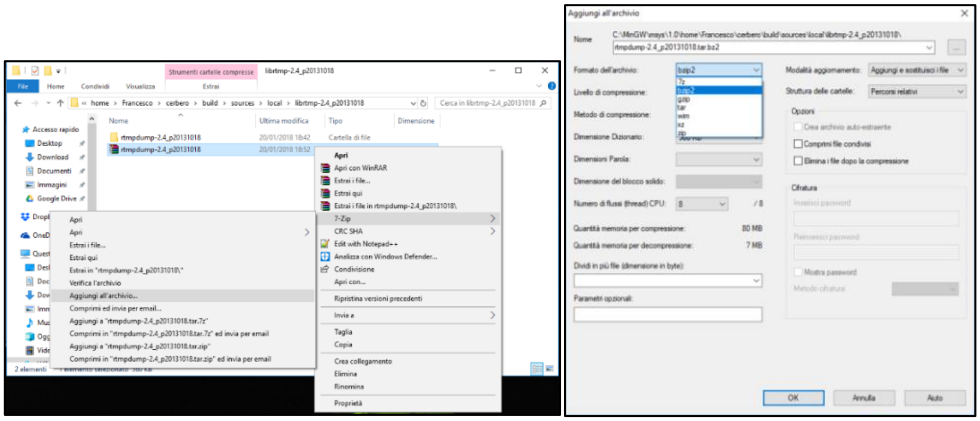

Figura 32 - Il risultato della compressione precedente, viene compresso a sua volta in un archivio bz2

5. Assicurarsi che il nuovo archivio tar.bz2 ottenuto (rtmpdump-2.4_p20131018) si trovi nella cartella C:\MinGW\msys\1.0\home\NomeUtente\cerbero\build\sources\local\librtmp-2.4_p20131018
6. Collocarsi nella cartella C:\MinGW\msys\1.0\home\NomeUtente\cerbero\build\sources\ windows_x86_64 e cancellare la cartella librtmp-2.4_p20131018
7. Aggiungere uno spazio vuoto (per aggiornarne la data di modifica) al file C:\MinGW\msys\1.0\home\NomeUtente\cerbero\recipes\librtmp\librtmp.recipes
8. Dare il seguente comando nella shell:
   ```
   $ cerbero build librtmp
   ```
9. Aggiungere uno spazio vuoto a C:\MinGW\msys\1.0\home\NomeUtente\cerbero\recipes\gst-plugins-bad-1.0.recipe
10. Dare il seguente comando nella shell:
    ```
    $ cerbero build gst-plugins-bad-1.0
    ```

5.3.2 Modifica di gsttaglist.h e gstflvmux.c

Una volta modificati i sorgenti gsttaglist.h e gstflvmux.c così come descritto nella sezione 3.3.2, si seguono i seguenti passaggi.

Modifica di gstaglist.h

Questo sorgente è contenuto nella cartella cerbero/build/sources/windows_x86_64/gstreamer-1.0-1.9/gst.

1. Posizionarsi nella cartella opportune tramite il comando
   ```
   $ cd cerbero/build/sources/windows_x86_64/gstreamer-1.0-1.9/gst
   ```
2. Inviare il comando e annotare
   ```
   $ make
   ```
3. Occorre commentare le righe dei file gstsystemclock.c e gstutils.c (che si trova nella stessa cartella di gsttaglist.h) che hanno generato errore nel comando “make”.
4. Dopodiché si inviano i comandi
   ```
   $ make
   $ make install
   ```

gstflvmux.c

gstflvmux.c, si trova nella cartella C:\MinGW\msys\1.0\home\NomeUtente\cerbero\build\sources\windows_x86_64\gst-plugins-good-1.0-1.9\gst\flv

Una volta modificato si apre la shell e si digitano i seguenti comandi:

```
$ cd cerbero/build/sources/windows_x86_64/gst-plugins-good-1.0-
1.9/gst/flv
$ make
$ make install
```

5.3.3 Modifica del Makefile e di gsttaginject.c

Collocarsi nella cartella:
C:\MinGW\msys\1.0\home\NomeUtente\cerbero\build\sources\windows_x86_64\gst-plugins-good-1.0-1.9\gst\debugutils

Dapprima occorre apportare delle modifiche alle seguenti righe del Makefile per includere la libreria “gio-2.0” nella compilazione.

```
GLIB_LIBS = -LC:/MinGW/msys/1.0/home/<NomeUtente>/cerbero/build/dist/windows_x86_64/lib -
lgobject-2.0 -lgmodule-2.0 -lglib-2.0 -lintl -lgio-2.0
```

```
GST_BASE_LIBS = -LC:/MinGW/msys/1.0/home/<NomeUtente>/cerbero/build/dist/windows_x86_64/lib
-lgstbase-1.0 -lgstreamer-1.0 -lgobject-2.0 -lglib-2.0 -lintl -lgio-2.0
```

```
GST_PLUGINS_BASE_LIBS = -
LC:/MinGW/msys/1.0/home/<NomeUtente>/cerbero/build/dist/windows_x86_64/lib -lgstreamer-1.0
-lgobject-2.0 -lglib-2.0 -lintl -lgio-2.0
```

Dopodiché si modifica gsttaginject.c, e nella stessa cartella, si inviano i comandi:

```
$ make
$ make install
```

7 Conclusioni

In conclusione di questo lavoro di tesi, alla luce dei risultati ottenuti nei test effettuati, possiamo asserire di essere riusciti a pervenire ad una architettura di streaming in cui l'encoding e l'immissione dei tag in tempo reale nel flusso audio, avvengono attraverso una componente sviluppata estendendo del software open source. Abbiamo visto in particolare come sia possibile modificare le librerie open source di GStreamer e di RTMP, raggiungendo l'obiettivo della compatibilità del flusso multimediale in uscita dall'encoder con i moduli del server di streaming, i quali vengono lasciati sostanzialmente intatti.

In questo percorso ci si è scontrati con diverse problematiche. Inizialmente non si riusciva a venire al capo del fatto che la stringa @setDataFrame dei pacchetti AMF generati dalla versione originale di librtmp su cui si poggia il plug-in rtmpsink di Gstreamer, non riuscisse a *triggerare* la funzione del modulo di Wowza che è stato aggiunto all'applicazione "live" per intercettare i metadati e formattarli come tag ID3. Lo studio di RTMP, dei sorgenti della relativa libreria e la loro successiva modifica ha portato alla soluzione di questo problema, ed è stata la chiave di volta che ha permesso tutto lo sviluppo successivo.

Quindi avendo imparato a sfruttare alcuni strumenti utili come il framework GLib e la libreria GIO, con i rispettivi supporti per la programmazione ad oggetti e i socket in C (quest'ultimi utili a implementare il server in ascolto dei metadati sulla porta TCP), a quel punto è stato facile concentrarsi nella modifica e nell'utilizzo della componente taginject di GStreamer, che può essere vista, in fin dei conti, come una black box, con più ingressi e una sola uscita. Nel nostro caso gli ingressi sono i due canali per l'introduzione dei metadati (file e porta TCP) e uno per la ricezione del flusso audio, e l'uscita è un canale da cui esce il flusso audio con gli eventuali metadati immessi.

Abbiamo visto inoltre come attivare e come funziona il meccanismo di autenticazione della sorgente RTMP lato-server.

Infine si è riusciti ad implementare su Windows quanto fatto su Linux. A tal proposito è risultato fondamentale l'apporto del software MinGW e del sistema Cerbero, che ci hanno consentito rispettivamente di effettuare il porting di GCC e la compilazione cross-platform di GStreamer su Windows, dove le procedure di bootstrap e build avvengono grazie a degli script particolari chiamati "recipes", che contengono gli url delle librerie e dei plug-in e alcune opzioni che è stato necessario aggiornare e modificare per far sì che la compilazione andasse a buon fine.

Bibliografia

[1] GStreamer, «faac,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad/html/gst-plugins-bad-plugins-faac.html.

[2] GStreamer, «What is GStreamer,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html.

[3] GStreamer, «Tutorials,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/tutorials/index.html.

[4] GStreamer, «Elements,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/application-development/basics/elements.html.

[5] GStreamer, «gst-launch-1.0,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/tools/gst-launch.html.

[6] GStreamer, «audiotestsrc,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-audiotestsrc.html.

[7] GStreamer, «aacparse,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good/html/gst-plugins-good-plugins-aacparse.html.

[8] GStreamer, «faad,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad/html/gst-plugins-bad-plugins-plugin-faad.html.

[9] GStreamer, «autoaudiosink,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good/html/gst-plugins-good-plugins-autoaudiosink.html.

[10] GStreamer, «taginject,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good/html/gst-plugins-good-plugins-taginject.html.

[11] GStreamer, «flvmux,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good/html/gst-plugins-good-plugins-flvmux.html.

[12] GStreamer, «rtmpsink,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad/html/gst-plugins-bad-plugins-rtmpsink.html.

[13] Adobe Systems, «Adobe's Real Time Messaging Protocol,» [Online]. Available: http://wwwimages.adobe.com/www.adobe.com/content/dam/acom/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf.

[14] Adobe Systems, «AMF Specification,» [Online]. Available: https://wwwimages2.adobe.com/content/dam/acom/en/devnet/pdf/amf0-file-format-specification.pdf.

[15] Wikipedia, «Wowza Streaming Engine,» [Online]. Available: https://en.wikipedia.org/wiki/Wowza_Streaming_Engine.

[16] Wowza, «Video Streaming Server,» [Online]. Available: https://www.wowza.com/products/streaming-engine.

[17] Wowza, «User's Guide,» [Online]. Available: http://www.wowza.com/resources/WowzaStreamingEngine_UsersGuide-4.0.6.pdf.

[18] Wowza, «How to convert onTextData events in live, VOD, or nDVR streams to timed events (ID3 tags) in Apple HLS streams,» [Online]. Available: https://www.wowza.com/docs/how-to-convert-ontextdata-events-in-a-live-or-vod-stream-to-timed-events-id3-tags-in-an-apple-hls-stream.

[19] The GNOME Project, «GSocket: GIO Reference Manual,» [Online]. Available: https://developer.gnome.org/gio/stable/GSocket.html.

[20] GStreamer, «Transform elements,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/design/element-transform.html.

[21] GStreamer, «GstBaseTransform,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-libs/html/GstBaseTransform.html.

[22] The GNOME Project, «String Utility Functions,» [Online]. Available: https://developer.gnome.org/glib/stable/glib-String-Utility-Functions.html.

[23] GStreamer, «GstTagList,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstTagList.html.

[24] GStreamer, «tcpserversrc,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-tcpserversrc.html.

[25] GStreamer, «Adding properties,» [Online]. Available: https://gstreamer.freedesktop.org/documentation/plugin-development/basics/args.html.

[26] Wikipedia, «MinGW,» [Online]. Available: https://en.wikipedia.org/wiki/MinGW.

[27] «Cerbero,» [Online]. Available: https://github.com/GStreamer/cerbero.

[28] Wikipedia, «Advanced Audio Coding,» [Online]. Available: https://it.wikipedia.org/wiki/Advanced_Audio_Coding.

[29] Wikipedia, «ActionScript,» [Online]. Available: https://en.wikipedia.org/wiki/ActionScript.

[30] Wikipedia, «SHOUTcast,» [Online]. Available: https://it.wikipedia.org/wiki/SHOUTcast.

[31] The GNOME Project, «GIO Overview - Introduction,» [Online]. Available: https://developer.gnome.org/gio/stable/ch01.html.

[32] Gstreamer, «Gstreamer - Open Source Multimedia Framework,» [Online]. Available: https://gstreamer.freedesktop.org/.

[33] GStreamer, «GStreamer Editing Services Overview,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-editing-services/html/ch01.html.

[34] GStreamer, «alsasrc,» [Online]. Available: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-alsasrc.html.

[35] Wikipedia, «Advanced Linux Sound Architecture,» [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Linux_Sound_Architecture.

[36] Wikipedia, «ECMAScript,» [Online]. Available: https://it.wikipedia.org/wiki/ECMAScript.