Università di Pisa

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

Implementazione di un modulo Network Coding per sistemi di comunicazione multipath in real-time

Candidato:
Claudio Fadda

Tutore Accademico:
Prof. Stefano Chessa

Tutori Aziendali:
Dott. Felice Manlio Bacco
Dott. Alberto Gotta

Anno Accademico 2015-2016

Indice

Introduzione

L'obiettivo di questo lavoro è stato la creazione di un plugin per il framework di sviluppo di applicazioni multimediali Gstreamer [1]; il framework è open-source. Questo lavoro contribuisce all'implementazione di funzionalità definite nell'IETF draft in [2], ma ad oggi non ancora disponibili, nell'ambito di trasmissioni basate su tecniche *multipath*, che vedono l'utilizzo del protocollo Multipath Real-time Transport Protocol (MP-RTP) [2],

Nello specifico, lo scenario considerato vede la trasmissione di un flusso multimediale in real-time da una camera montata su un Unmanned Aerial Vehicle (UAV) a una postazione fissa di riproduzione a terra. La trasmissione avviene mediante l'utilizzo di più canali fisici di comunicazione in contemporanea (multipath).

Il plugin, o modulo di comunicazione, implementato in questo lavoro e descritto nel seguito, aggiunge funzionalità di *error correction* e di *security* all'implementazione esistente di MP-RTP [3]. I due obiettivi sono raggiunti con l'integrazione software di tecniche di Network Coding (NC) [4], nello specifico con l'utilizzo del Random Linear Network Coding (RLNC) [5], che offre protezione dagli errori e intrinseca crittografia dei dati. Una semplice ma efficace schematizzazione del lavoro svolto è visibile in tabella 4.2.

Acronimi utilizzati

ARQ	Automatic Repeat reQuest
FEC	Forward Error Correction
MP-RTCP	Multipath Real-time Transport Control Protocol
MP-RTP	Multipath Real-time Transport Protocol
NC	Network Coding
PLR	Packet Loss Rate
QoE	Quality of Experience
QoS	Quality of Service
RLNC	Random Linear Network Coding
RR	Receive Report
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
RTT	Round Trip Time
SR	Send Report

Capitolo 1

Scenario di riferimento

1.1 Obiettivi

Il presente lavoro di tesi è sviluppato su uno specifico scenario di riferimento; tuttavia, come discusso in seguito, questo lavoro si presta a molteplici scenari di utilizzo. In dettaglio, lo scenario qui considerato è relativo alla trasmissione di flussi video in real-time, da un UAV (Unmanned Aerial Vehicle) in volo ad una postazione fissa a terra. In questo contesto, molteplici sono i requisiti perché sia possibile una trasmissione dati multimediale in real-time. Di seguito i principali:

- *banda larga*, necessaria a garantire il trasporto di un flusso video ad alta definizione;
- *affidabilità* del canale, per garantire una connessione stabile e uno streaming multimediale privo di interruzioni;
- *costi contenuti*, per garantire l'accessibilità ad un bacino di utenza il più ampio possibile.

Questo lavoro ha visto lo sviluppo software di un modulo per il framework multimediale open-source *Gstreamer* [1], col fine di fornire ulteriore affidabilità alle trasmissioni dati multimediali con sistemi multipath.

Nel seguito, utilizzeremo la definizione di *canale fisico* per indicare il mezzo (Ethernet, Wi-Fi, Bluetooth, ecc.) che collega un trasmettitore e un ricevitore per la trasmissione fisica a distanza dell'informazione; come *canale logico* intendiamo invece un canale di comunicazione realizzato mediante l'utilizzo (aggregazione) di più canali fisici.

L'idea è quella di avere a disposizione un *canale logico* con le caratteristiche già descritte, ottenuto sfruttando tecniche di aggregazione di più *canali fisici*,come rappresentato, ad esempio, in figura 1.1. A tale fine, è stato utilizzato il protocollo di comunicazione Multipath Real-time Transport Protocol (MP-RTP), già a disposizione in Gstreamer. Il modulo sviluppato all'interno di questo lavoro di tesi permette l'utilizzo di tecniche *Network Coding*, descritte in Sezione 3, col fine di aumentare l'affidabilità del canale logico utilizzato.

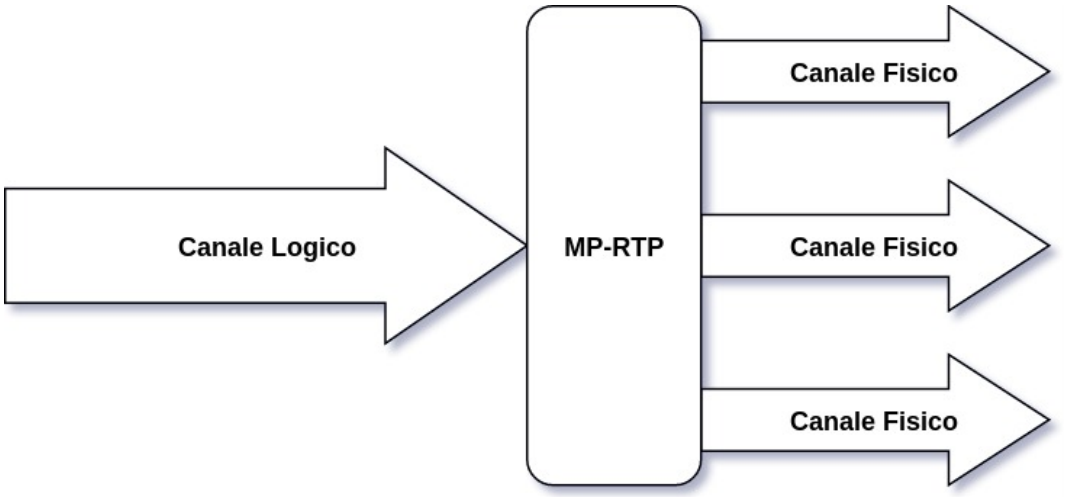

Figura 1.1: Realizzazione di un canale logico con l'utilizzo di più canali fisici

1.2 Real-time Transport Protocol

Il protocollo RTP, descritto in [6], è un protocollo di livello applicativo utilizzato per la comunicazione in tempo reale su reti IP, in particolare in sistemi di comunicazione di streaming multimediale, telefonia, applicazioni di videoconferenza e servizi televisivi. Si basa sull'utilizzo del protocollo UDP a livello di trasporto. RTP prevede, opzionalmente, l'utilizzo di meccanismi a feedback, basati sull'utilizzo del protocollo Real-time Transfer Control Protocol (RTCP), descritto in [7]. Quest'ultimo fornisce statistiche sulla qualità del servizio (QoS), inviando periodicamente dei pacchetti di feedback. Tale feedback include parametri come: Packet Loss Rate (PLR), variazione del ritardo di ricezione dei pacchetti (Jitter) e round trip time (RTT).

1.2.1 Affidabilità

RTP non prevede nessun sistema di garanzia della consegna dei pacchetti e ogni pacchetto ha un numero di sequenza (SN, Sequence Number) che viene

utilizzato in ricezione per ricostruire la sequenza originale dei dati trasmessi. Ad ogni sorgente di uno stream dati è associato un numero intero a 32 bit, detto SSRC (Synchronization Source Identifier), che funge da identificatore univoco.

1.3 MultiPath Real-time Transport Protocol

Come descritto in [2, 8], MP-RTP è un protocollo di livello applicativo, come visibile in figura 1.2, sviluppato per permettere la trasmissione di un flusso multimediale su più canali fisici. MP-RTP permette di sfruttare un canale logico suddividendo un flusso dati in ingresso in più sottoflussi assegnati ad altrettanti canali fisici in uscita. MP-RTP estende il protocollo RTP, quindi ne eredita le funzionalità principali, quali trasmissione end-to-end di pacchetti multimediali, sincronizzazione intra ed inter-stream, monitoraggio end-to-end e sessione di controllo. A livello di pacchetto, il protocollo MP-RTP estende l'header dei pacchetti RTP andando ad aggiungere in coda le informazioni a lui necessarie, ma lasciando inalterata la parte riguardante il protocollo RTP. Questa è una condizione fondamentale per garantire la retrocompatibilità del protocollo MP-RTP.
MP-RTP prevede la presenza di uno scheduler lato *sender* capace di suddividere un flusso RTP in più sottoflussi, instradati nella rete in maniera indipendente. I dati vengono ricombinati a destinazione da uno scheduler lato *receiver*, che riordina i pacchetti basandosi sul Sequence Number. L'applicazione che utilizza MP-RTP vede quindi un unico canale logico, e lo scheduling sottostante è *trasparente* all'applicazione, come visibile in figura 1.3.

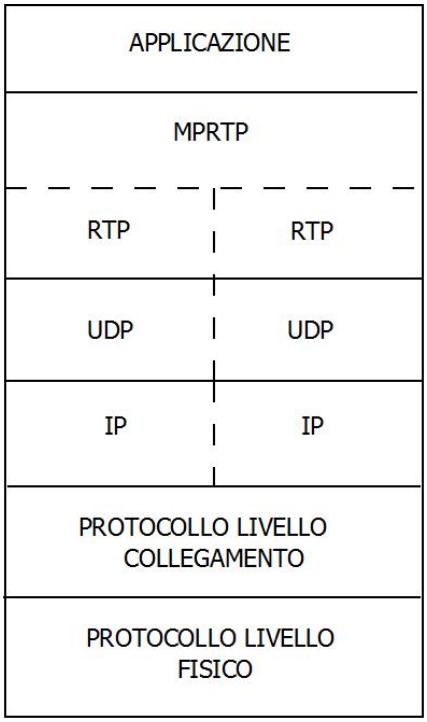

Figura 1.2: Stack protocollare con MP-RTP

Il protocollo MP-RTP offre quindi tre funzionalità principali:

- schedulazione dei pacchetti: il flusso principale, proveniente dall'applicazione, viene diviso in più sottoflussi e inviato su interfacce radio multiple, una per sottoflusso;
- ricombinazione dei sotto-flussi: a destinazione, il flusso originale è ottenuto ricombinando i dati;
- gestione dei path di rete: MP-RTP conosce i diversi path verso ogni singolo host di destinazione e stabilisce in che modo instradare i pacchetti per ogni destinazione.

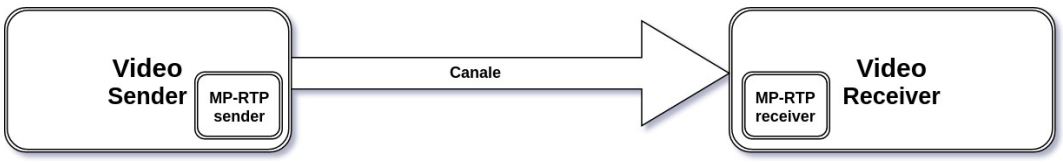

Figura 1.3: Integrazione di MP-RTP nel modulo di trasmissione

1.3.1 Vantaggi offerti da MP-RTP

I principali vantaggi offerti dal protocollo MP-RTP sono i seguenti:

- incremento del throughput: l'aggregazione di più canali fisici in un unico canale logico mette a disposizione più banda, quindi un bit-rate sorgente più alto è possibile, incrementando la qualità del flusso video;

- bilanciamento del carico: la trasmissione di un flusso RTP su percorsi multipli può ridurre il consumo di banda su un unico percorso; il protocollo MP-RTP prevede la ridistribuzione del carico sui percorsi disponibili al variare delle loro caratteristiche;

- affidabilità: il protocollo MP-RTP prevede l'utilizzo di meccanismi di ridondanza (Forward Error Correction (FEC), ritrasmissione, ecc.), facendo sì che interruzioni su un percorso abbiano meno impatto sulla qualità dello stream percepita dall'utente.

1.3.2 Instradamento dei dati

Per assicurare una maggiore affidabilità nella trasmissione dei dati, una stazione MP-RTP deve poter scegliere in maniera flessibile come instradare i pacchetti attraverso i canali di comunicazione disponibili. L'instradamento dei dati è deciso in base alle finalità della trasmissione e, conseguentemente, in base ai parametri più rilevanti in tale contesto: ad esempio, nel caso di uno streaming video, in cui la qualità finale della trasmissione è molto sensibile alla perdita di dati, il tasso di perdita di pacchetti può essere preso come fattore maggiormente rilevante nella scelta dei percorsi in modo da direzionare il carico maggiore sui canali più affidabili da questo punto di vista. I canali con un tasso di perdita troppo elevato per la trasmissione video possono comunque essere utilizzati per la trasmissione di altri dati, meno soggetti ai danni relativi alla perdita di pacchetti, come ad esempio il monitoraggio del traffico.

Per assicurare la massima efficienza, il protocollo MP-RTP deve quindi monitorare costantemente lo stato dei percorsi attivi servendosi del protocollo MP-RTCP, descritto in Sezione 1.4.

1.3.3 Capacità di adattarsi alle variazioni di banda

Una stazione che utilizza il protocollo MP-RTP dovrebbe essere in grado di ridistribuire il carico di traffico su altri percorsi quando uno o più dei percorsi correnti vengono congestionati o presentano un alto tasso di perdita. In questi casi si opera spostando tutto il traffico verso percorsi meno congestionati, ma MP-RTP invia comunque una piccola porzione di dati su questi percorsi

congestionati in modo da riuscire a monitorare continuamente le caratteristiche del percorso.

1.3.4 Scheduling dei pacchetti

Nella fase di avvio, l'algoritmo di scheduling dei pacchetti, fig. 1.4, prevede che sia assegnata ad ogni canale una percentuale uguale del fusso principale. Dopo l'invio dei primi pacchetti, si iniziano a ricevere i report di controllo del protocollo MP-RTCP che forniscono le informazioni sullo stato dei canali e che indicano la storia a breve termine del canale stesso. A questo punto l'algoritmo di scheduling calcola e assegna ad ogni canale una diversa percentuale del flusso principale basando tali scelte sulle statistiche ricevute.

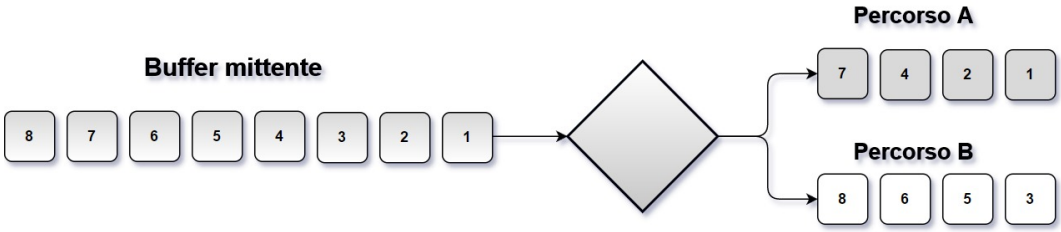

Figura 1.4: Distribuzione dei pacchetti tramite algoritmo di scheduling.

Il lato receiver, fig. 1.5, si preoccupa di ricostruire il flusso originale a partire dai sottoflussi ricevuti. I pacchetti provenienti dai vari sottoflussi vengono ricombinati e passati ad un buffer, descritto nel dettaglio in Sezione 1.3.6, utilizzato per compensare la variazione del tempo di interarrivo dei pacchetti e per il loro riordinamento. Il ricevitore segnala regolarmente le informazioni sul trasporto (RTT, PLR, ecc.) al mittente, che le userà nelle politiche di scheduling.

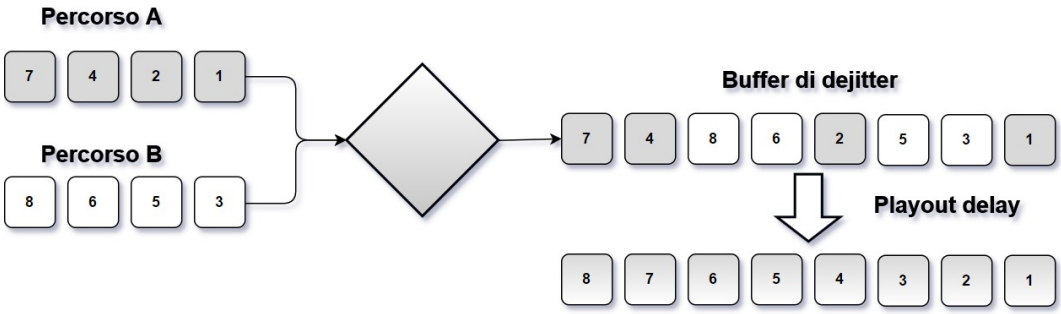

Figura 1.5: Ricostruzione del flusso principale con buffer di dejitter e playout delay

1.3.5 Intervalli di scheduling

Le operazioni di scheduling e di riassegnazione dei carichi ai vari percorsi possono essere computazionalmente impegnative e possono perciò introdurre latenza. Per ovviare a questo problema è possibile impostare la frequenza di esecuzione di tali calcoli.

1.3.6 Buffer di dejitter e ritardo di playout

In uno scenario multipath, i pacchetti vengono schedulati in percorsi con latenze ampiamente variabili e pacchetti inviati su percorsi diversi avranno ritardi diversi. Questo implica il fatto che i vari pacchetti arrivino a destinazione fuori ordine.
Per ovviare a questo problema sul lato receiver è utilizzato un buffer di dejitter. Il buffer di dejitter serve a compensare la variazione del tempo di interarrivo dei pacchetti e per fare ciò deve necessariamente essere sufficientemente lungo. L'utilizzo di un buffer di dejitter può non bastare per via del fatto che l'RTT può andare oltre al valore compensabile con il la dimensione del buffer calcolata. Per questo motivo oltre al buffer di dejitter viene introdotto un ritardo di playout adattivo per ogni percorso opportunamente calcolato da un algoritmo in modo da avere il tempo necessario per assicurare che la riproduzione del video sia fluida e priva di interruzioni, al costo dell'introduzione di un tempo di latenza in fase di riproduzione. In uno scenario multipath i pacchetti vengono inviati attraverso percorsi con latenze ampiamente variabili ed i metodi utilizzati potrebbero non essere comunque sufficienti.

1.3.7 Retrocompatibilità

Il protocollo MP-RTP offre retrocompatibilità trasparente tra un host MP-RTP e uno RTP. Le estensioni multipath, presenti nelle intestazioni MP-RTP, sono ignorati da un host RTP.

1.4 Multipath Real-time Transport Control Protocol

Il protocollo MP-RTP prevede un sistema di monitoraggio dello stato dei percorsi verso l'host di destinazione. Per fare questo si avvale del protocollo

MP-RTCP, estensione di RTCP. RTCP utilizza uno scambio di pacchetti di report suddivisi in due categorie:

- Sender Report (SR): inviati dalla stazione sorgente alla stazione di destinazione, contengono le statistiche riguardanti la trasmissione e la ricezione dei dati;
- Receiver Report (RR): inviati dalla stazione di destinazione alla stazione sorgente, contengono solamente le statistiche riguardanti i dati ricevuti.

Con l'utilizzo di questo meccanismo, un sender MP-RTP ha gli elementi necessari per poter decidere come instradare i pacchetti sui vari percorsi disponibili, come discusso nella sezione 1.3.3.

1.4.1 Intervalli di report MP-RTCP

Gli intervalli tra l'invio di due report consecutivi sono generalmente inclusi tra i *5 ms* e i *2,5 ms*, [8], come accade per RTP. É comunque possibile inviare alcuni report in maniera più frequente nel caso in cui si verifichino perdite o scarti di pacchetti di tali pacchetti

1.5 Casi d'uso più comuni

I protocolli MP-RTP e MP-RTCP vengono utilizzati principalmente in ambito di trasmissioni multimediali in real-time di diverso tipo:

- streaming multimediale ad alto bitrate su dispositivi multihomed
- comunicazione Voice Of Ip (VOIP) in dispositivi a interfacce multiple (es. 3G e wifi)
- collegameto multiplo in tecnologia 3G: in alcuni contesti, come ad esempio sui treni, vengono utilizzati diversi collegamenti 3G per formarne uno virtuale.

Capitolo 2

Piattaforma software utilizzata

In base alle caratteristiche del materiale già presente e alle necessità per lo sviluppo del progetto abbiamo scelto il *C* come linguaggio di programmazione e il framework Gstreamer come piattaforma per la realizzazione del modulo di comunicazione. Come descritto al paragrafo 2.1, Gstreamer è un software particolarmente indicato per lo sviluppo di applicazioni orientate alla trasmissione di dati multimediali in real-time. Per quanto riguarda la fase di test è stato utilizzato il software *Dummynet*, descritto al paragrafo 4.3.3, per la simulazione dei canali di trasmissione e per modificarne le caratteristiche, e il programma Speedometer, descritto al paragrafo 4.3.4, per osservare il carico dei dati trasmessi su ogni canale.

2.1 Gstreamer

Gstreamer [1, 9] è un framework per la creazione di applicazioni orientate alla gestione di flussi multimediali. Il concetto fondamentale di Gstreamer è quello di leggere un file in un formato, processarlo ed esportarlo in un altro formato. Gstreamer è una piattaforma che si basa sul concetto di pipeline, attraverso cui fluiscono i dati multimediali. Nello specifico, il framework mette a disposizione:

- un insieme di procedure per la creazione di applicazioni multimediali;
- un insieme di plug-in;
- un'architettura a pipeline;
- un meccanismo per la gestione/negoziazione del tipo di dispositivo utilizzato;

- un meccanismo per la sincronizzazione.

Le pipeline Gstreamer vengono realizzate collegando in serie diversi elementi, detti *plug-in*, aventi delle porte di connessione, dette *pad*, di ingresso e uscita, indicati rispettivamente come *source pad* e *sink pad*, per la connessione con gli elementi adiacenti.
I plug-in possono essere suddivisi in tre categorie generali, visibili in fig. 2.1:

- *source*: si occupano di prelevare il file o il flusso multimediale sorgente e hanno una uscita *source pad*;
- *filter*: eseguono operazioni di elaborazione del flusso come codifica, decodifica, compressione, decompressione ecc. Hanno sempre almeno un *source pad* e un *sink pad*;
- *sink*: elementi di interfaccia di ricezione del flusso. Fondamentalmente si occupano della ricezione/riproduzione. Hanno una uscita sink pad.

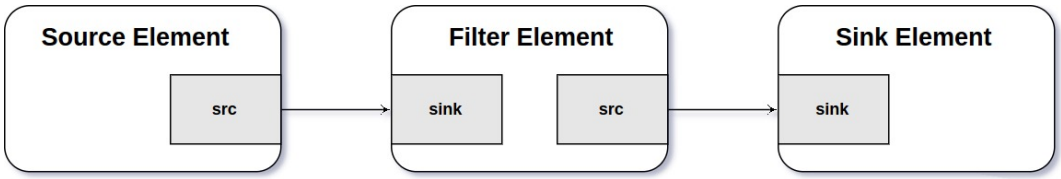

Figura 2.1: Schema di una pipeline Gstreamer minimale.

2.1.1 Architettura modulare di Gstreamer

Gstreamer utilizza una architettura a plug-in ed essi sono l'elemento principale nella costruzione delle applicazioni. Ogni plug-in ha una funzione specifica, come ad esempio la lettura o scrittura dei dati su file, la decodifica o l'invio dei dati su una periferica di output. I plug-in sono visti come delle black box attraverso le quali i dati fluiscono e vengono processati. Concatenando diversi plug-in, è possibile realizzare una pipeline capace di svolgere un compito specifico come la riproduzione, la cattura o la trasmissione di un flusso dati. Gstreamer possiede una vasta gamma di plug-in pronti all'uso, ma è anche possibile lo sviluppo di nuovi plug-in secondo le proprie esigenze. Il cuore del nostro progetto è stato proprio la realizzazione di un plug-in che permette l'utilizzo del RLNC su Gstreamer, a partire dalla implementazione preesistente Kodo [10].

2.1.2 Installazione di Gstreamer

Per poter utilizzare Gstreamer è necessario, per prima cosa, installare sulle macchine utilizzate il pacchetto principale "gstreamer", ed i suoi principali plug-in aggiornati alla stessa versione del pacchetto principale.

La versione di Gstreamer utilizzata in questo contesto è la 1.7.1 e sono stati installati i vari plug-in ad essa associati. I plug-in installati, descritti in [11], sono:

- gst_plugin_base
- gst_plugin_good
- gst_plugin_ugly
- gst_plugin_bad
- gst_libav

2.2 Descrizione del plug-in MP-RTP

Oltre ai plug-in forniti da Gstreamer è stato installato il plug-in "gst_-mprtp_master" [3] che implementa un set di strumenti per la realizzazione di applicazioni per la trasmissione multipath utilizzando il protocollo MP-RTP [2].
Il pugin è ancora in via di sviluppo ed è di tipo *open source*.

2.2.1 Struttura del plug-in MP-RTP

Il plug-in MP-RTP è composto da diversi elementi da utilizzare, alcuni al lato sender e altri al lato receiver, per creare una comunicazione multipath. Di seguito gli elementi principali per il lato sender:

- Receiver: ancora in via di sviluppo, si occupa della ricezione e della gestione dei pacchetti MP-RTCP di tipo RR provenienti dai collegamenti dedicati, uno per ognuno degli N sottoflussi. Questo componente è direttamente collegato al componente *Scheduler* e ad esso invia le statistiche da utilizzare per il calcolo della distribuzione del carico di dati sugli N sottoflussi.
- Scheduler: questo componente svolge principalmente i compiti di gestione della creazione e eliminazione dei sottoflussi, del settaggio dei loro

parametri e della creazione dei pacchetti SR contenenti le informazioni di controllo per il protocollo MP-RTCP. I parametri di settaggio dei vari sottoflussi vengono utilizzati dal componente *Sender*, collocato dopo lo *Scheduler* nella pipeline, per prendere le decisioni in merito alla distribuzione dei pacchetti.

- Sender: anch'esso in via di sviluppo, questo componente gestisce i pacchetti MP-RTP e MP-RTCP che riceve per mezzo di due porte distinte. I pacchetti MP-RTCP vengono inviati sulla porta a loro dedicata, mentre i pacchetti MP-RTP vengono spartiti sulle N porte, una per ogni sottoflusso, dedicate alla trasmissione MP-RTP, secondo i parametri settati dal componente *Scheduler*. Nel caso in cui non sia ancora stato ricevuto alcun parametro dallo *Scheduler*, i parametri dei sottoflussi vengono settati con valori di default.

Dal lato receiver vengono invece utilizzati i seguenti componenti:

- Receiver: diversamente dall'omonimo lato sender, il *Receiver* viene usato per ricevere i pacchetti provenienti dagli N sottoflussi. Dopo averne controllato l'integrità e il tipo, invia i dati al componente successivo.

- Playouter: riceve, attraverso una unica porta, i dati dal componente *Receiver*, li invia al componente successivo e ricava le informazioni necessarie a creare i pacchetti RR MP-RTCP che verranno inviati al mittente che li utilizzerà per creare le statistiche dei canali.

- Sender: riceve i pacchetti RR del protocollo MP-RTCP dal componente *Playouter* e, dopo averne verificato il tipo, alla porta di uscita ad essi dedicata.

2.2.2 Limitazioni del plugin MP-RTP

Benché implementi il protocollo MP-RTP, il plug-in MP-RTP [3] utilizzato ancora non rispetta alcune delle specifiche richieste:

- **adattamento ai cambiamenti di banda:** il plug-in assegna in maniera statica le percentuali di carico ad ogni canale e non prevede nessun ricalcolo della distribuzione al variare delle caratteristiche qualitative del collegamento, come ad esempio in caso di congestionamento.

- **affidabilità:** non è previsto nessun meccanismo di ridondanza, pertanto la comunicazione è soggetta a dei visibili cali di qualità, come la presenza

di artefatti o l'interruzione dello streaming, in caso di perdita di informazione durante la trasmissione. In particolare non è presente nessun codice a correzione di errore (FEC)[12].

2.3 Estensione del plug-in MP-RTP

Il lavoro in [13] propone una soluzione al primo problema citato nel paragrafo 2.2.2 con la reimplementazione di alcune funzioni del plug-in MP-RTP. Di seguito la descrizione delle funzionalità disponibili in [13].
Ad ogni canale di invio è stato associato un canale di ricezione feedback, fig. 2.2, utilizzando il protocollo MP-RTCP, attraverso cui vengono trasmessi tre parametri, con un messaggio chiamato *report*.
Attraverso l'uso dei *report* MP-RTCP è stato introdotto un sistema di monitoraggio che permette di conoscere in tempo reale lo stato dei canali di comunicazione dando una stima della qualità di ognuno di essi.
Grazie ai parametri ricevuti con i *report*, indicati come RTT, PLR e Jitter, e una funzione matematica opportunamente creata è possibile stimare la qualità di un canale ad intervalli di tempo regolari:

- RTT: Round Trip Time, indica il tempo trascorso tra l'invio di un pacchetto e la ricezione del suo *report*.
- PLR: Packet Loss Rate, indica il tasso di perdita di pacchetti su un canale.
- Jitter: indica la variazione del ritardo di trasmissione dei pacchetti.

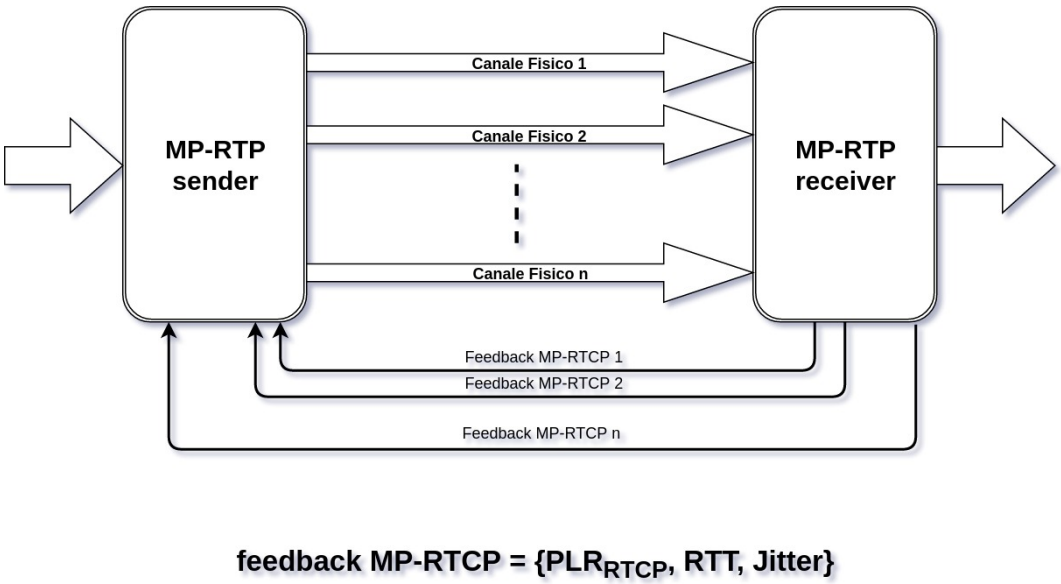

Figura 2.2: Posizionamento del plug-in MP-RTP

Lo scheduler MP-RTP e' stato modificato, perché fosse capace di ricalcolare e ridistribuire le percentuali di carico sui diversi sottoflussi in base ai parametri di feedback ricevuti sui canali di ricezione. Per la ridistribuzione dei carichi sui sottoflussi è stato introdotto un meccanismo a penalità: dato un valore massimo di trasmissione, espresso in KB/s, questo viene ridotto in maniera proporzionale al valore della penalità calcolata. La penalità del sottoflusso i-esimo è definita come *i*-esimo come $f_i = f_i(t, \cdot)$, cioè come una funzione che dipende dal tempo e dal valore dei parametri RTT, di PLR o di Jitter, contenuti nei Report di feedback ricevuti. Per ciascuno dei parametri l'ultimo campione di misura ricevuto tramite il *Report*, è indicato con R_i^t, mentre il valore medio di tutte le misure disponibili, a meno dell'ultima, è indicato con $\bar{R}_i$ nella formula seguente:

$$f_i = \alpha\, R_i^t + (1 - \alpha)\, \bar{R}_i \tag{2.1}$$

dove α viene letto durante la fase di inizializzazione dal file di configurazione e indica come distribuire il peso storico dei dati (valor medio) e dell'ultima misura disponibile nel calcolo del valore della penalità. La penalità calcolata per ogni canale viene quindi usata come parametro per calcolare come spartire il flusso principale. In questo modo è possibile adattare la distribuzione dei carichi sui flussi disponibili in base alle variazioni di banda.

2.3.1 Limitazioni del plug-in MP-RTP esteso

Nella sezione precedente è stato descritto come ad ognuno dei tre parametri di feedback ricevuti viene assegnato un peso differente nel calcolo della penalità. In questo caso il peso maggiore è stato attribuito al parametro PLR in quanto ad una anche minima perdita di dati corrisponde una notevole perdita di qualità del flusso video riprodotto. Dare maggiore importanza al parametro PLR fa si che ad un canale soggetto alla perdita di pacchetti venga assegnata una percentuale di carico prossima allo zero. Questo comportamento fa si che un canale con caratteristiche non ottime tenda ad essere completamente inutilizzato. Inoltre i dati inviati non hanno nessuna forma di codifica, pertanto vengono trasmessi in chiaro esponendo così la trasmissione a possibili intercettazioni malevole.

Capitolo 3

Network Coding

Il Network Coding (NC) [14] è una tecnica che può essere utilizzata per migliorare il throughput di una rete, l'efficienza e la scalabilità, nonché la resistenza agli attacchi e alle intercettazioni. Anziché limitarsi a trasmettere i pacchetti di informazioni ricevuti, i nodi di una rete prendono diversi pacchetti e li combinano insieme prima di trasmetterli.

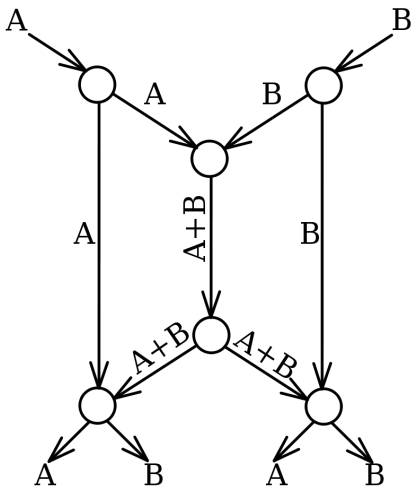

Figura 3.1: Rete a farfalla

Per comprendere meglio il funzionamento del NC, e come questo possa aumentare le performance di una rete, prendiamo l'esempio della rete a farfalla in figura 3.1. Supponiamo di avere due nodi di origine (in alto nell'immagine) che devono trasmettere i pacchetti A e B ai due nodi di destinazione (in basso). Supponiamo inoltre che ogni collegamento possa trasportare un solo pacchetto per unita' di tempo. In una rete con routing classico, il collegamento centrale sarebbe in grado di trasportare solo A o solo B, ma non entrambi allo stesso tempo. Il routing è quindi insufficiente perché nessuno schema di

instradamento può trasmettere sia A che B contemporaneamente ad entrambe le destinazioni.
Supponiamo di inviare A attraverso il collegamento: centrale la destinazione a destra riceverebbe correttamente entrambi i valori, la destinazione a sinistra invece riceverebbe A due volte, ma non riceverebbe B. L'invio di B sul collegamento centrale pone un problema simile per la destinazione a destra.

Utilizzando un semplice codice, come illustrato in 3.1, A e B possono essere trasmessi ad entrambe le destinazioni contemporaneamente inviando la somma dei valori attraverso il collegamento centrale; in altre parole, dobbiamo codificare A e B con la formula $A \oplus B$. La destinazione a sinistra riceve A e $A \oplus B$, e può calcolare B sottraendo i due valori. Analogamente, la destinazione a destra riceverà B e $A \oplus B$, e sarà in grado di determinare sia A e B.

3.1 Random Linear Network Coding (RLNC)

Il Random Linear Network Coding (RLNC) [5] è un sistema di codifica semplice ma potente, che permette di avvicinarsi ad un throughput ottimale utilizzando un algoritmo decentralizzato. In una rete che utilizza il RLNC, un gruppo di nodi P è coinvolto nell'invio di dati da S nodi origine a K nodi destinazione. Ogni nodo suddivide i pacchetti in ingresso in blocchi di k pacchetti e genera n nuovi pacchetti, con $n \geq k$, ottenuti come combinazioni dei pacchetti ricevuti, moltiplicati per dei coefficienti scelti in maniera casuale da un campo finito (Galois field) di dimensione $q = 2^x$. I nodi destinazione ricevono i messaggi RLNC, e li salvano in una matrice. I messaggi originali vengono quindi recuperati eseguendo l'*eliminazione di Gauss* su tale matrice. I coefficienti usati per la codifica alla sorgente sono aggiunti all'header dei pacchetti (header RLNC), in modo che il ricevente possa decodificare i blocchi di dati. Se la dimensione del campo è sufficientemente grande, la probabilità che il ricevitore ottenga k combinazioni linearmente indipendenti, dati k pacchetti ricevuti, è prossima a 1. Almeno k combinazioni lineari indipendenti devono essere ricevute perché la decodifica sia possibile.

3.2 Kodo (libreria software RLNC)

Come descritto in [10], Kodo è una libreria di correzione di errore ad alte prestazioni focalizzata su algoritmi Network Coding. Kodo è scritta in C++ ed è stato progettata per garantire alte prestazioni, flessibilità ed estensibi-

lità. Le librerie Kodo dispongono di una semplice Application Programming Interface (API), che permette ai programmatori di integrare il NC nelle proprie applicazioni. Kodo supporta vari codec NC, come ad esempio RLNC, sistematico e non [1], o sparse RLNC. Ogni algoritmo offre vantaggi e svantaggi diversi, nonché un diverso insieme di parametri che possono essere impostati in tempo reale. Ad esempio, Kodo può essere impiegato per utilizzare diversi canali cellulari contemporaneamente, per impostare la quantità di ridondanza in una rete di storage distribuito, o per adattare i nodi in caso di guasti nelle reti magliate.

3.2.1 kodo-c

Come riportato in [15], la libreria *kodo-c* fornisce interfacce di alto livello in *C* per accedere alle funzionalità base di Kodo, come la codifica e la decodifica di dati con vari codec. La libreria kodo-c dispone di una semplice API in *C* che permette al programmatore di utilizzare Kodo senza alcuna preoccupazione per i sottostanti dettagli di implementazione in *C++*. In base alle sua caratteristiche di performance e di praticità di utilizzo, questa libreria è stata scelta per la realizzazione del nostro plug-in RLNC.

[1] La codifica sistematica prevede l'invio dei pacchetti originali all'inizio di ogni blocco trasmesso mentre i pacchetti codificati vengono inviati come parte finale del blocco. In questo modo si riduce il costo di decodifica in caso di perdita nulla. Con la codifica non sistematica tutti i pacchetti vengono codificati prima di essere inviati.

Capitolo 4

Sviluppo del Progetto

Lo sviluppo del progetto è stato diviso in differenti fasi:

- Studio del framework Gstreamer e del plug-in MP-RTP: è stato studiato l'ambiente Gstreamer, sono stati analizzati nello specifico tutti i componenti del modulo MP-RTP preso come base di partenza, ne è stato studiato il funzionamento e in particolare è stato studiato come e dove inserire, all'interno della pipeline, le funzionalità RLNC.

- Implementazione del plug-in RLNC: fase centrale del lavoro. É stato studiato il sistema di sviluppo di plug-in offerto da Gstreamer ed è stato realizzato il modulo RLNC.

- Modifiche ai programmi Client e Server: sono state eseguite alcune modifiche sui programmi Client e Server in modo da poter integrare e testare il modulo RLNC sviluppato.

- Setup del sistema e test: in questa fase è stato testato il funzionamento dell'intero modulo di comunicazione con l'utilizzo dei programmi di test Client e Server, eseguiti su una rete virtuale appositamente creata e settata mediante l'utilizzo di appositi script *bash* e del programma di emulazione *Dummynet*.

4.1 Studio del framework Gstreamer e del modulo MP-RTP

La prima fase dello svolgimento del progetto è consistita in un lavoro di studio per capire appieno il funzionamento dell'ambiente di sviluppo Gstreamer e del plugin MP-RTP. Come prima cosa è stato studiato nei dettagli il framework Gstreamer analizzando a fondo il sistema di sviluppo di applicazioni descritto in [9], in cui sono illustrati i meccanismi per la costruzione delle pipeline di comunicazione a partire dai singoli elementi, dopodiché è stata eseguita un'analisi dettagliata del plug-in MP-RTP. In particolare è stata analizzata la costruzione della pipeline, utilizzata per la trasmissione multipath, per cercare di capire come integrare e configurare le funzionalità di RLNC. La prima ipotesi fatta è stata quella di eseguire una ulteriore modifica al plug-in MP-RTP in modo da integrare al suo interno le funzionalità di RLNC: dopo una attenta analisi, a questa soluzione abbiamo preferito quella della realizzazione di un plug-in indipendente in modo da sfruttare a pieno le caratteristiche modulari di Gstreamer. Un plug-in RLNC indipendente offre il vantaggio di poter essere inserito o meno nella pipeline di trasmissione senza nessuna particolare modifica. Un'altro vantaggio di questa soluzione è il fatto di non legare l'utilizzo del plug-in RLNC in modo esclusivo a questo contesto: il plug-in sviluppato può infatti essere utilizzato in una qualsiasi pipeline di trasmissione che utilizzi il protocollo RTP per integrarne le funzionalità di RLNC. Oltre ad una maggiore flessibilità, la soluzione proposta offre una maggiore leggibilità e una più semplice ed intuitiva strutturazione del codice.

4.1.1 Posizionamento del plug-in RLNC nella pipeline di comunicazione

Un aspetto chiave nella realizzazione del progetto è stato lo studio del punto in cui inserire il plug-in RLNC all'interno della pipeline *Gstreamer*: le soluzioni analizzate sono state principalmente due e per ognuna di esse sono stati valutati attentamente vantaggi e svantaggi.

4.1.2 RLNC prima dello scheduling dei flussi

La prima soluzione analizzata prevede l'inserimento del plug-in RLNC sul flusso dati principale, prima della procedura di scheduling e della divisione in più sottoflussi, fig. 4.1: in questo caso, il flusso dati in formato RTP passa

attraverso un *encoder* RLNC che genera i dati codificati, dopodiché il flusso codificato viene trasmesso al *sender* del plug-in MP-RTP che provvede alla sua ripartizione attraverso i canali di comunicazione. L'idea sostanziale alla base di tale soluzione è quella di poter comunque utilizzare un canale affetto da perdita di pacchetti facendo fluire al suo interno esclusivamente dei dati ridondanti. In questo modo, gli effetti ottenuti si tradurrebbero in una maggiore robustezza della trasmissione; i dati ridondanti trasmessi potrebbero compensare la perdita di pacchetti su altri canali. Mantenere un canale attivo permetterebbe inoltre di monitorare lo stato dello stesso attraverso i l'utilizzo del protocollo di controllo MP-RTCP e di conseguenza consentirebbe di scoprire se, a partire da un certo istante, la qualità del canale *migliora* al tal punto da poter considerare il canale nuovamente affidabile e quindi utilizzabile.

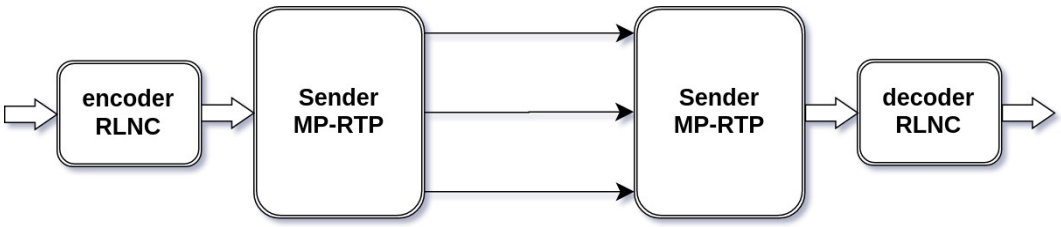

Figura 4.1: Schema di posizionamento del modulo RLNC sul flusso dati prima che questo venga diviso dallo scheduler

Un altro vantaggio di tale soluzione è il costo computazionale; in questo modo si andrebbe a creare una sola istanza di *encoder* e *decoder* beneficiando così di vantaggi come l'allocazione di un minor numero di strutture dati e della riduzione del numero di calcoli da eseguire. Questa soluzione presenta però alcuni svantaggi: facendo lavorare in questa maniera il modulo RLNC non è possibile generare una quantità di dati ridondanti mirata a compensare la perdita di pacchetti di un canale specifico, ma si dovrà generare una quantità di dati di protezione sulla base della media (PLR) dei pacchetti persi rilevata sul canale logico.

4.1.3 Posizionamento del plug-in RLNC dopo lo scheduling dei flussi

La seconda opzione analizzata prevede l'utilizzo di un modulo RLNC per ogni sottoflusso generato dal plug-in MP-RTP, fig.4.2. In questa configurazione si va a creare un'istanza della coppia *encoder* e *decoder* per ogni canale di comunicazione. Questa soluzione permette una azione mirata alla compensazione

della perdita di pacchetti su ogni singolo canale di trasmissione. Conoscendo la percentuale di dati persi su ogni canale è così possibile compensare le perdite generando una quantità di dati ridondanti sufficiente ma non eccessiva.
Gli svantaggi di questa soluzione un maggiore costo computazionale, una maggiore allocazione di strutture dati necessarie alla codifica e la necessità di un canale di feedback aggiuntivo per ogni modulo di codifica-decodifica.

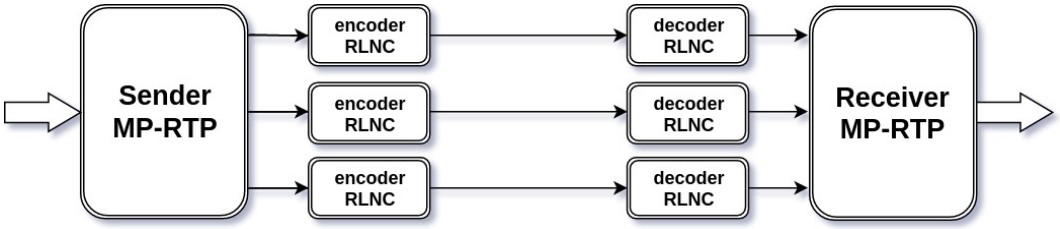

Figura 4.2: Schema di posizionamento del plug-in RLNC su ogni sottoflusso

4.2 Implementazione del plug-in RLNC

L'obiettivo principale del progetto è la realizzazione di un plug-in RLNC da integrare nella pipeline di trasmissione.

4.2.1 Scelta implementativa

Dopo l'analisi e la valutazione delle due opzioni architetturali descritte in 4.1.1 abbiamo optato per la scelta di realizzare un plug-in RLNC da posizionare dopo lo scheduling dei flussi, come in sezione 4.1.3. Le motivazioni di tale scelta sono:

- il matching perfetto tra PLR del canale e quantità di ridondanza generata;
- l'utilizzo di tecniche NC offre vantaggi rispetto all'utilizzo di tecniche FEC [16] e di tecniche Automatic Repeat reQuest (ARQ) [17].

Il plug-in è stato realizzato utilizzando la libreria per la codifica RLNC Kodo, descritta in sezione 3.2, ed è composto da due componenti principali:

- Encoder: svolge la funzione di codifica dei dati di generazione dei pacchetti di ridondanza
- Decoder: riceve e decodifica il flusso codificato ripristinando i dati originali

Per lo studio degli strumenti di sviluppo di un plug-in abbiamo utilizzato la guida fornita da Gstreamer [18]. Come prima cosa, sia per il componente *Encoder* che per il *Decoder* è stato generato un template avente le funzionalità base per la gestione di un flusso dati, dopodiché sono state implementate e aggiunte le funzioni di RLNC.

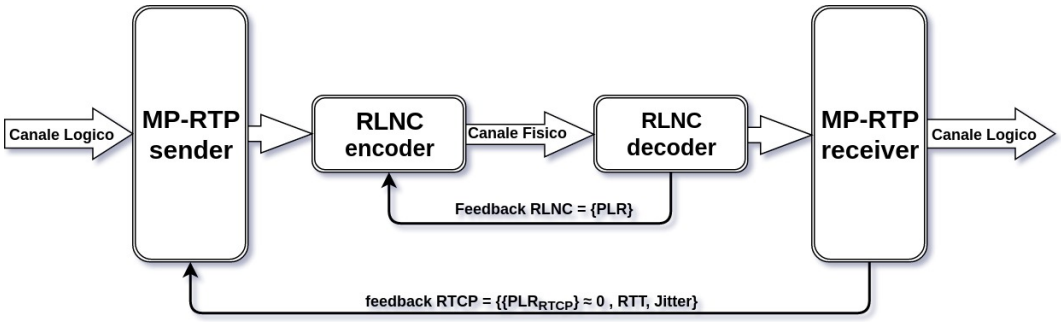

Figura 4.3: Schema del funzionamento del plug-in RLNC per ogni canale fisico

Come descritto nella figura 4.3, il plug-in aggiunge la funzionalità di RLNC su ogni canale fisico con lo scopo di compensare la PLR di ognuno di essi. Per fare questo l'*Encoder* suddivide i dati in ingresso in blocchi di k pacchetti, che vengono memorizzati in un buffer di dimensione k; dopo aver riempito il buffer i pacchetti, con l'utilizzo del padding, vengono portati tutti alla massima dimensione contenibile in un payload RTP: il blocco così ottenuto viene passato alla funzione *encoder* che genera n pacchetti di uscita, con $k \leq n$. Il *Decoder* ha invece il compito di decodificare i dati ricevuti sul canale di comunicazione mediante una apposita funzione: perché la decodifica di un blocco vada a buon fine sarà necessario che il *Decoder* riceva almeno k degli n pacchetti inviati dall'*Encoder* e che i pacchetti ricevuti siano tutti diversi tra loro. Per assicurare la ricezione di un numero sufficiente di pacchetti da parte del *Decoder* viene dunque calcolato il numero n di pacchetti in uscita dall'*Encoder* in maniera appropriata alla qualità del canale di comunicazione rilevata con un meccanismo di feedback.

Il plug-in realizzato crea un proprio anello di feedback attraverso cui vengono trasmessi, dal *Decoder* verso l'*Encoder* i dati relativi alla PLR, rilevata a livello RLNC. Dato il plug-in MP-RTP, i parametri per la gestione della ripartizione del carico sui diversi sottoflussi, trasmessi con il protocollo MP-RTCP, sono i seguenti:

$$feedback\ RTCP := \{PLR_{RTCP}, RTT, Jitter\} \tag{4.1}$$

Inserendo il plug-in RLNC, che compensa le perdite con la produzione di un numero appropriato di pacchetti ridondanti, l'effetto ottenuto è quello di azzerare, con buona approssimazione, la PLR rilevata dal protocollo RTCP. La 4.1 è quindi trasformata nella seguente:

$$feedback\ RTCP := \{\approx 0, RTT, Jitter\} \tag{4.2}$$

4.2.2 Encoder

Il componente *Encoder* si occupa di codificare i dati in ingresso, di gestire la generazione dei dati ridondanti e dell'invio dei dati elaborati al componente successivo. Questo elemento è stato progettato per avere due interfacce *sinkpad*, una per l'ingresso del flusso video e una per la ricezione dei feedback PLR, e una interfaccia *srcpad* utilizzata per l'inoltro dei dati in uscita.

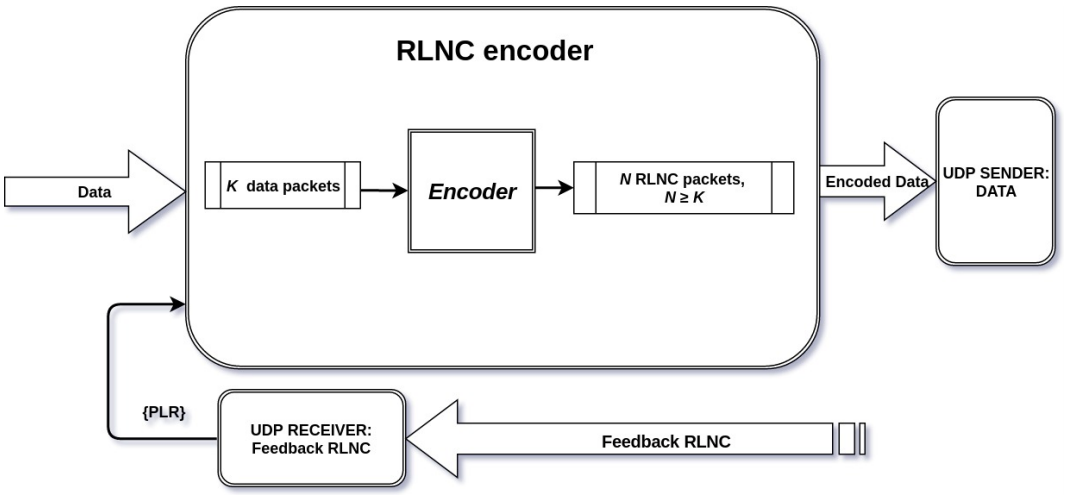

Figura 4.4: Schema del funzionamento del modulo encoder RLNC

Come è possibile osservare nella figura 4.4, al componente *Encoder* è collegato un plug-in $UDP\,RECEIVER$ attraverso una della due interfacce di ingresso, e ha il compito di ricevere i messaggi di feedback RLNC inviati dal *Decoder* attraverso un canale UDP appositamente creato. La principale funzione implementata in questo elemento è la *gst_my_kodo_enc_chain*: come prima cosa per ogni pacchetto ricevuto la funzione ne controlla l'integrità e il tipo. Definiamo con z il numero di valori di feedback di cui tenere uno storico. Se l'*Encoder* riceve da una delle due interfacce di ingresso un pacchetto di tipo diverso dall'RTP[1], questo sarà necessariamente di un pacchetto di feedback: in

[1]l*Encoder* non può stabilire da quale delle due interfacce sia stato ricevuto il pacchetto, ma può solo stabilirne il tipo

questo caso il parametro indicante la percentuale di pacchetti persi nel blocco appena trasmesso viene memorizzato in un array circolare di dimensione z. Definiamo PLR_i come l'ultimo valore di feedback ricevuto e $\bar{M}$ come la media di tutti gli ultimi z valori di PLR_i salvati nell'array circolare, escluso l'ultimo ricevuto. I due parametri vengono utilizzati per il calcolo del numero di pacchetti di ridondanza da generare per compensare la percentuale di pacchetti persi. Definiamo PLR_t un valore che dipende dal tempo e dall'ultimo PLR_i ricevuto; viene calcolato mediante l'utilizzo di un filtro di Kalman descritto in [19]:

$$PLR_t = (1 - \alpha)\,\bar{M} + \alpha\,PLR_i \tag{4.3}$$

dove $0 \leq PLR_t < 1$ e α, detto parametro di ottimizzazione, indica la distribuzione del peso attribuito alle variabili $\bar{M}$ e PLR_i nel calcolo del valore finale e viene letto dal file di configurazione $config\_file1.txt$ all'avvio del programma. Dopo aver ottenuto il parametro PLR_t, il numero n di pacchetti generati per compensare le perdite viene calcolato come di seguito:

$$n \geq \frac{k}{(1 - PLR_t)} \tag{4.4}$$

Se l'*Encoder* riceve in ingresso, un pacchetto dati di tipo RTP, questo è memorizzato in un buffer e si attende il pacchetto successivo. Al raggiungimento di k pacchetti collezionati nel buffer si avvia la procedura per la codifica del blocco: dato che la libreria di codifica Kodo necessita di ricevere in ingresso blocchi di pacchetti di dimensione omogenea, tutti i pacchetti memorizzati vengono ridimensionati e portati alla stessa dimensione, la massima contenibile in un payload RTP, utilizzando il meccanismo di *padding* descritto in [20]. L'array di elementi ottenuto viene a questo punto utilizzato dalla funzione di codifica Kodo che si occupa di codificare i dati e di generare n pacchetti in uscita, dove $k \leq n$ come descritto nella formula (4.4); uno dei punti di forza della libreria Kodo è la possibilità di poter decidere in qualsiasi istante il numero n di pacchetti di uscita generati dall'*encoder*. Dopo aver codificato il blocco, i pacchetti che lo costituiscono vengono inoltrati attraverso l'interfaccia di uscita dopo aver aggiunto in testa ad ognuno di essi un numero di sequenza relativo al blocco di appartenenza. Il procedimento di ricezione codifica e invio si ripete di continuo per tutta la durata della trasmissione. I valori delle variabili k, n, α e z vengono letti in fase di avvio dal file di configurazione $config\_file1.txt$.

4.2.3 Decoder

Il componente *Decoder*, fig. 4.5, realizzato ha il compito di ricevere, decodificare e inoltrare al componente successivo, i dati ricevuti su una interfaccia di ingresso. La struttura del *Decoder* prevede infatti una interfaccia *sinkpad* di ingresso e due interfacce *srcpad* di uscita: una viene utilizzata per l'invio dei dati decodificati al componente successivo mentre l'altra viene utilizzata per inviare dei messaggi di feedback al componente *Encoder*.

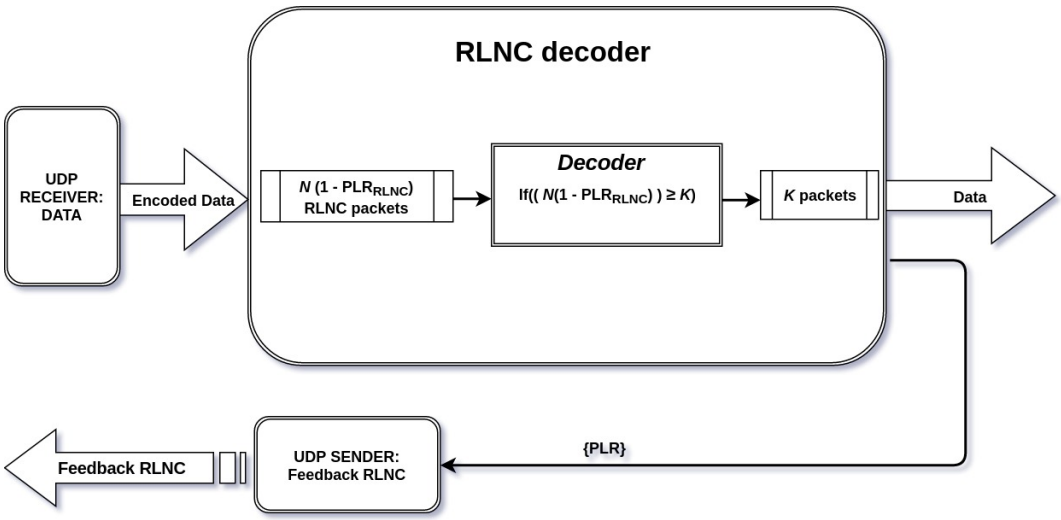

Figura 4.5: Schema del funzionamento modulo decoder RLNC

A questo elemento è collegato, attraverso una delle due interfacce di uscita, un plug-in $UDP\,SENDER$ incaricato di inviare i dati di feedback RLNC al componente *Encoder* utilizzando il canale UDP dedicato. Per ogni pacchetto ricevuto, come prima cosa il *Decoder* verifica il numero di sequenza che ne identifica il relativo blocco di appartenenza: come descritto in Sezione 4.2.2 i pacchetti vengono raggruppati in blocchi di dimensione k e numerati in base al blocco di cui fanno parte:

- se il numero di sequenza del pacchetto identifica un blocco la cui processazione è già stata terminata, il pacchetto viene ignorato;
- se il numero di sequenza del pacchetto identifica il blocco attualmente in fase di processazione, detto attuale, il pacchetto viene inoltrato alla funzione di decodifica: avendo la possibilità di decodificare parzialmente un blocco, la funzione controlla, per ogni pacchetto ricevuto, se è possibile la decodifica di qualche pacchetto di origine ancora non ottenuto e in caso affermativo, dopo aver marcato il pacchetto decodificato come tale, il

Decoder si preoccupa di eliminare i dati di *padding* inseriti dall'*Encoder*, riportando il pacchetto alla sua dimensione originale, e di inviare i dati ottenuti al componente successivo.

- nel caso in cui il numero di sequenza del pacchetto ricevuto identifichi un nuovo blocco la ricezione dei pacchetti del blocco attuale viene considerata conclusa, quindi il *Decoder* verifica se il blocco è stato decodificato completamente, invia il messaggio di feedback al componente *Encoder* e incrementa di uno il contatore del blocco attuale. Il messaggio di feedback inviato all'*Encoder* è incapsulato in un frame UDP e indica quanti pacchetti dell'ultimo blocco sono stati ricevuti. Il messaggio viene inoltrato attraverso una delle due interfacce *srcpad* verso un componente $UDP\,SENDER$ che invia il pacchetto attraverso il canale appositamente creato.

Anche in questo caso il procedimento si ripete di continuo per tutta la durata della trasmissione dati.

4.3 Setup del sistema e test

Per testare i plug-in MP-RTP e RLNC, sono stati realizzati i programmi *Client* e *Server*, una libreria *opt.h*,A.7 , e due script di setup della rete virtuale chiamati *setup_env.sh* e *setup_pipe.sh*, il cui codice è riportato in A.8 A.9. Mentre i programmi *Client* e *Server* creano una pipeline di trasmissione, collegando assieme diversi elementi, tra cui il plug-in MP-RTP e il plug-in RLNC, la libreria *opt.h* permette la scelta, da parte dell'utente, delle modalità con cui eseguire il programma di test definendo il modo in cui viene realizzata e configurata la pipeline di trasmissione. Lo script *setup_env.sh* permette di creare tre collegamenti virtuali ad ognuno dei quali sono associati un indirizzo lato client ed un indirizzo lato server. Lo script *setup_pipe.sh* invece setta le caratteristiche dei canali andando a modificare i parametri per l'emulazione della rete con l'utilizzo del software *Dummynet*, descritto in 4.3.3.

4.3.1 Server

Il programma *Server* si occupa del prelievo dei dati multimediali, da webcam oppure da file video in formato *.yuv*, e della loro trasmissione. Al suo avvio il programma legge dal file *opt.h*, in base al profilo scelto con le modalità

indicate nel file stesso, le informazioni necessarie alla configurazione e all'inizializzazione della pipeline di invio, dopodiché crea la pipeline. La struttura della pipeline creata sarà infatti diversa a seconda della modalità scelta dall'utente. Le specifiche su come calcolare il profilo di esecuzione del programma possono essere visualizzate chiamando lo stesso con l'opzione *–info*.

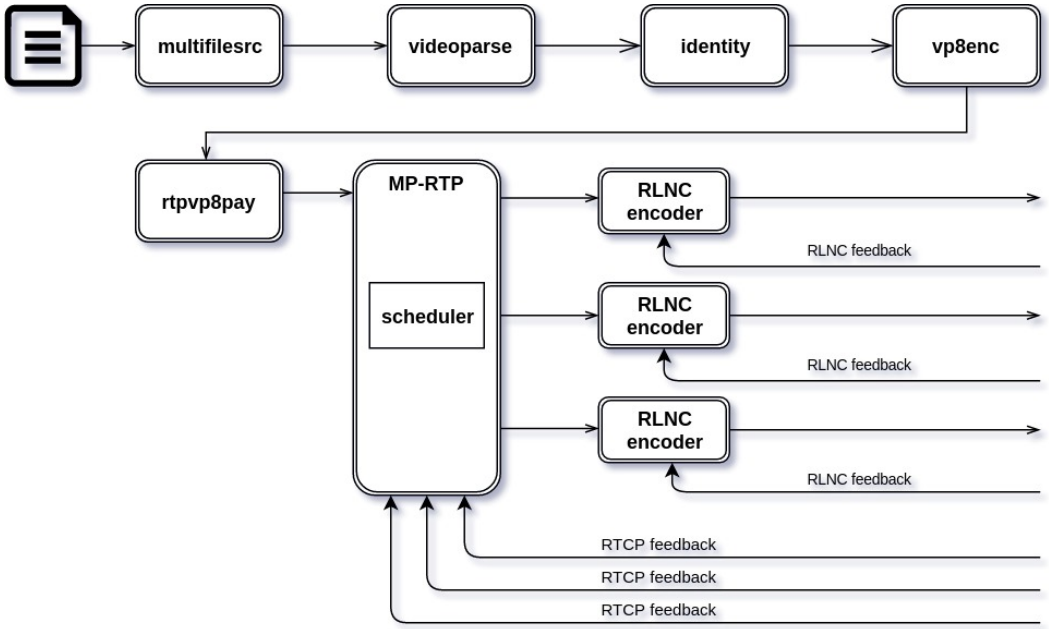

Figura 4.6: Esempio di costruzione della pipeline lato server con lettura da file e modulo codifica RLNC

Nella figura 4.6 è rappresentata una pipeline in cui è stata scelta l'acquisizione da file video e la codifica con RLNC. Gli elementi di cui è composta sono:

- multifilesrc: permette di leggere un file video passato come argomento del campo *location*;
- videoparse: converte il il flusso di dati in frame video;
- identity: passa i dati ricevuti al componente successivo senza apportare modifiche su di essi; viene usato per la sincronizzazione di frame video;
- vp8enc: codifica i frame video in formato *vp*8;
- rtpvp8enc: a partire dal flusso video codificato in formato *vp*8 incapsula i dati in pacchetti RTP;
- MP-RTP: riceve i pacchetti RTP e provvede al loro instradamento attraverso i canali disponibili come descritto in Sezione 2.3 ;

- RLNC encoder: esegue le operazioni di RLNC come descritto nella Sezione 4.2.2.

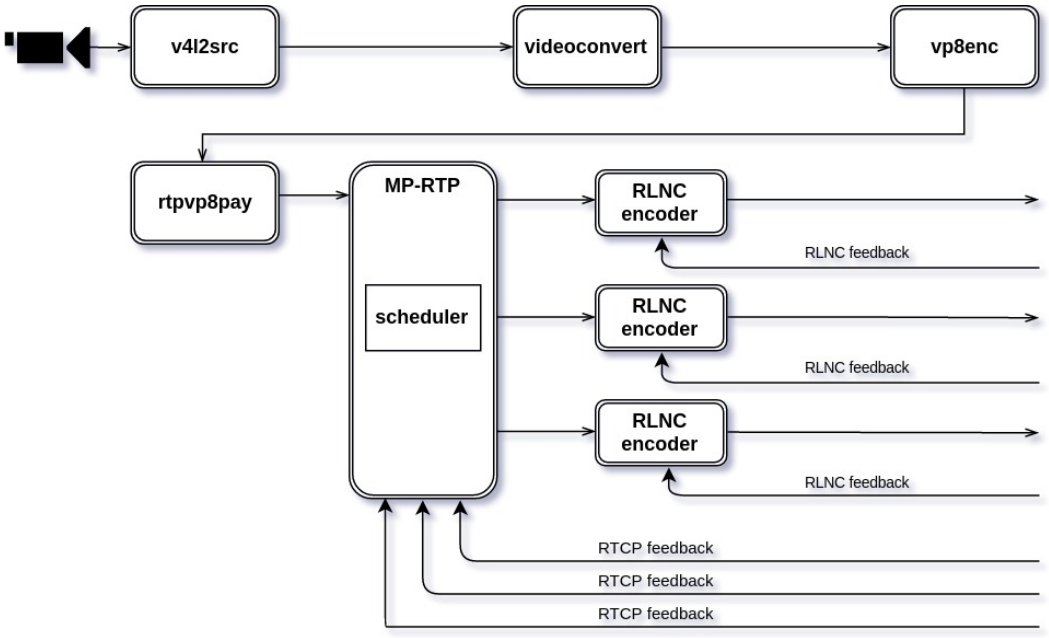

Figura 4.7: Esempio di costruzione della pipeline lato server con acquisizione da webcam e modulo di codifica RLNC

Nel caso in cui venga selezionata l'opzione di acquisizione video da webcam anziché da file video, la pipeline creata, mostrata nella figura sopra 4.7, presenta alcune differenze rispetto alla pipeline creata in caso di riproduzione da file: i componenti $multifilesrc$, $videoparse$ e $identity$ vengono sostituiti con $v4l2src$ e $videoconvert$, qui descritti:

- v4l2src: permette l'acquisizione dalla periferica webcam;
- videoconvert: converte i frame video acquisiti dalla webcam in frame video adatti alla trasmissione.

4.3.2 Client

Il programma *Client* è incaricato di ricevere, decodificare e ricomporre il flusso video originale per poi riprodurlo su un monitor o salvarlo su un file. Il suo avvio è simile a quello del programma *Server* e prevede anch'esso un profilo di esecuzione il cui calcolo può essere determinato a partire dalla tabella visualizzata chiamando il programma *Client* con l'opzione *–info*. Il programma crea una pipeline, in questo caso di ricezione, di elementi Gstreamer e la inizializza secondo le specifiche del profilo selezionato.

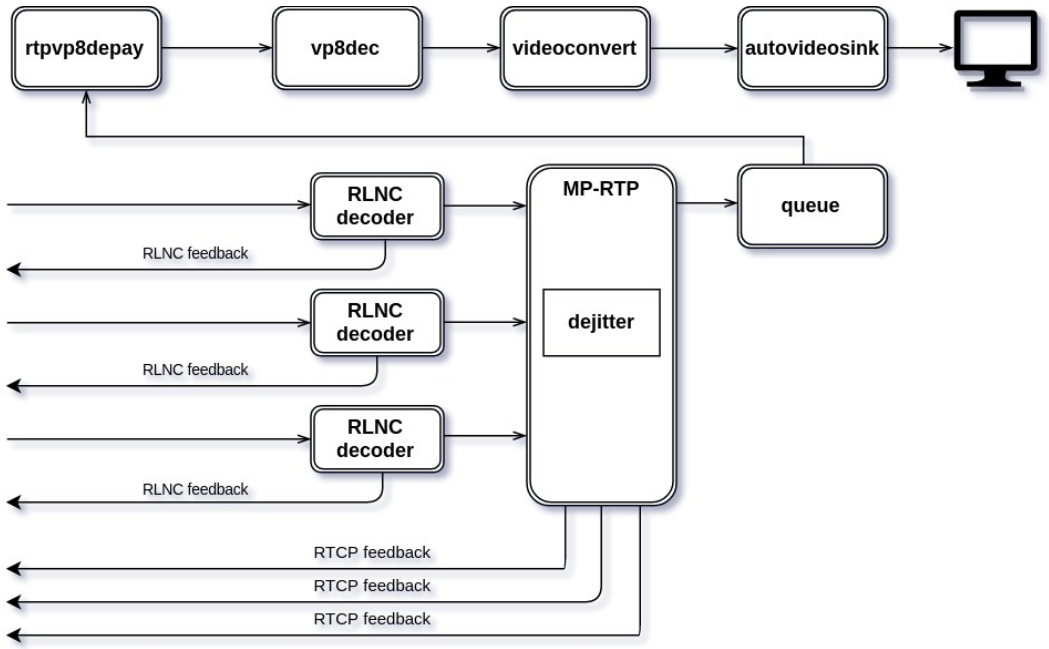

Figura 4.8: Esempio di costruzione della pipeline lato client con riproduzione su schermo e modulo di decodifica RLNC

La figura 4.8 mostra la composizione logica della pipeline creata dal *Client* in cui è stata scelta l'opzione di visualizzazione su schermo dello streaming video. Gli elementi utilizzati sono:

- RLNC decoder: esegue la decodifica dei blocchi RLNC;
- MP-RTP: si occupa della ricostruzione del flusso dati principale bufferizzando e riordinando i dati ricevuti dai sottoflussi attivi;
- queue: permette di creare una coda di ingresso per la ricezione dei dati;
- rtpvp8dapay: estrae dai pacchetti RTP il payload contenente i dati in formato vp8 e li passa al componente successivo;
- vp8dec: questo elemento decodifica i dati da formato vp8 a formato raw video;
- videoconvert: converte il flusso video da formato raw ad un formato video riproducibile;
- autovideosink: a partire dal flusso video ricevuto dal componente precedente crea una finestra e riproduce lo stream video su schermo.

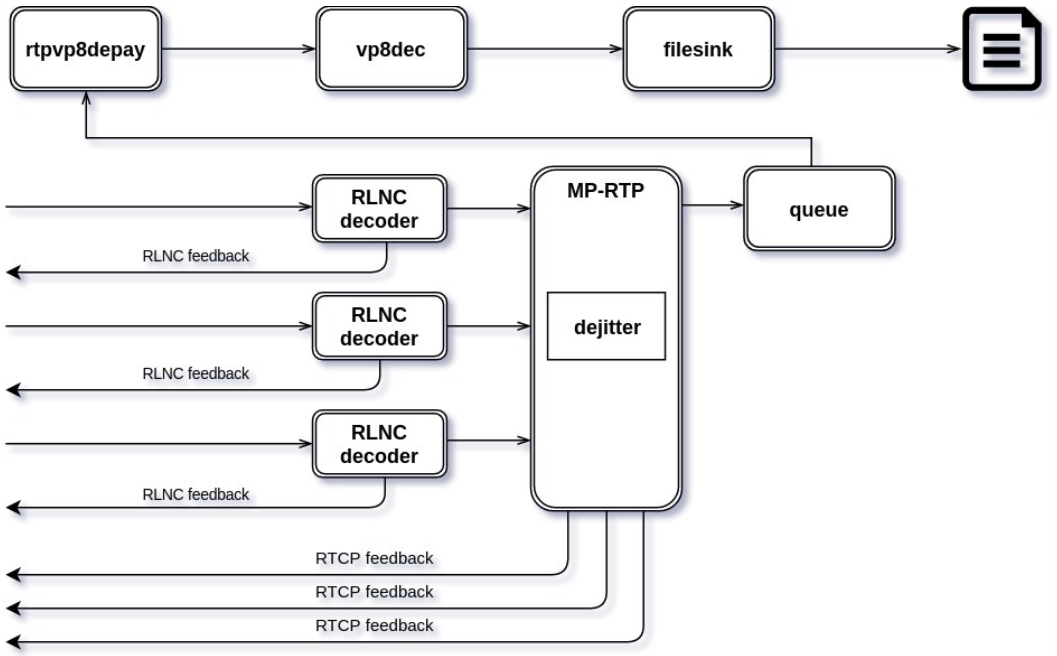

Figura 4.9: Esempio di costruzione della pipeline lato client con salvataggio su file e modulo di decodifica RLNC

Nel caso in cui venga selezionata l'opzione di salvataggio su file, come riportato in figura 4.9, i componenti *videoconvert* e *autovideosink* vengono sostituiti dal componente $filesink$, che permette di memorizzare il flusso video ricevuto in un file situato nella posizione indicata con il parametro *location*.

4.3.3 Dummynet

Dummynet [21, 22] è un uno strumento di emulazione di una rete, sviluppato in origine per testare i protocolli di rete e successivamente utilizzato per diverse applicazioni come la gestione della banda e di altre caratteristiche di rete. Questo pratico strumento permette di simulare e forzare dei parametri quali limitazioni di banda, ritardi e perdite di pacchetti. *Dummynet* intercetta i traffico selezionato attraverso lo stack di rete e lo ridireziona verso degli oggetti chiamati pipe, identificati con un nome univoco, che implementano un set di code, uno scheduler e un collegamento, il tutto con parametri configurabili (bandwidth, delay, loss rate, queue size, scheduling policy...) , settati dall'utente attraverso il firewall $ipfw$ [23]. Il nostro utilizzo di *Dummynet* è stato quello di creare dei canali virtuali su cui testare la nostra applicazione andando a simulare diverse condizioni di qualità del canale agendo in particolare sui parametri Packet Loss Rate e Round Trip Time.

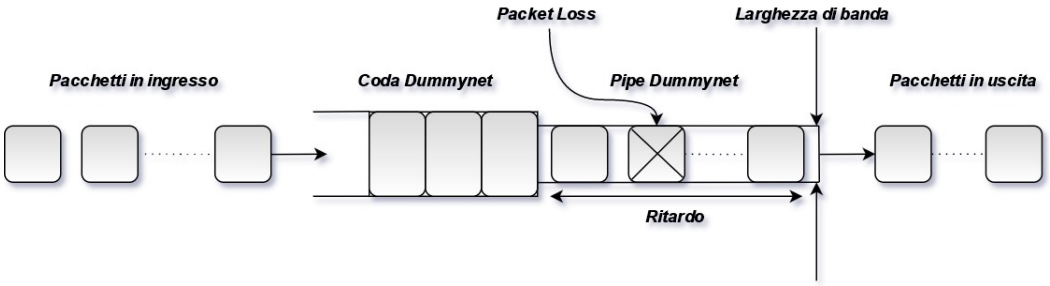

Figura 4.10: Una generica pipe creata con Dummynet

4.3.4 Speedometer

Come riportato in [24], Speedometer è un semplice programma che permette di misurare e visualizzare la velocità con cui fluiscono i dati attraverso un collegamento di rete. Nel nostro caso è stato utilizzato per verificare il variare della quantità di dati trasmessi attraverso i tre canali simulati, al variare dei parametri di codifica del plug-in RLNC.

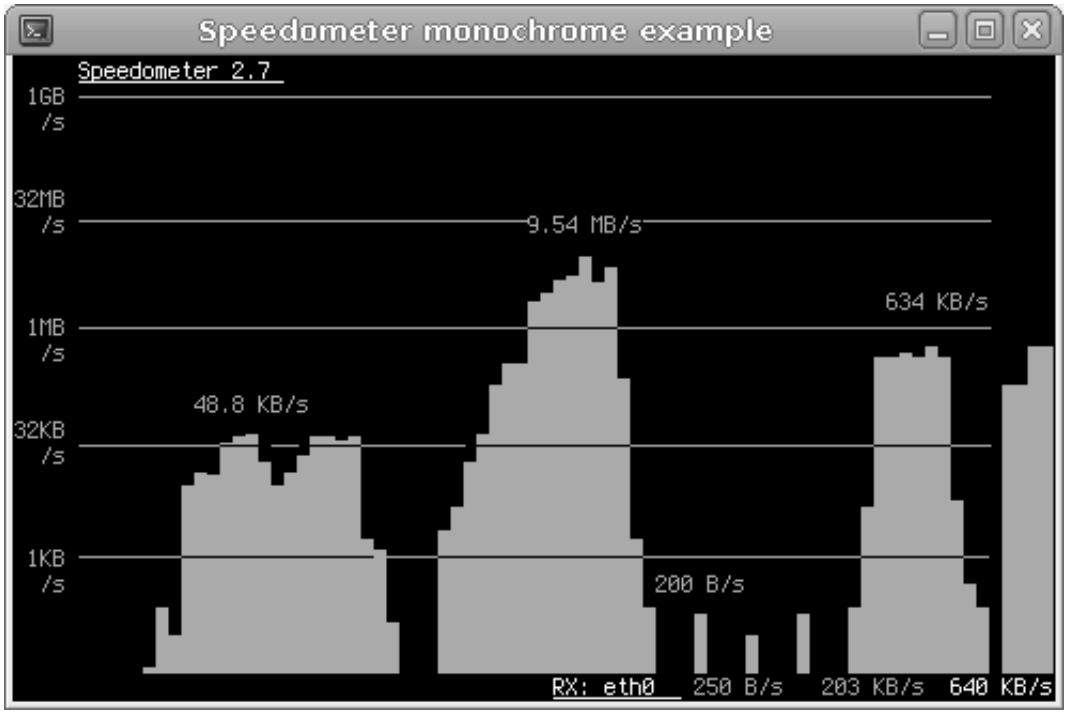

Figura 4.11: Schermata di Speedometer in fase di monitoraggio su un interfaccia di rete.

4.3.5 Creazione della rete virtuale

Per poter permettere l'esecuzione dei programmi di test viene creata una rete virtuale grazie ai due script *setup_env.sh* e *setup_pipe.sh*. Il programma

setup_env.sh si occupa di creare due *NetworkNamespace*, uno per il lato Server e uno per il lato Cient. I *NetworkNamespace*, descritti in [25], delle copie dello stack della rete principale, con tutti i suo percorsi, le regole del firewall e i dispositivi di rete, sui quali è possibile aggiungere delle diverse interfacce virtuali. Dopo la creazione dei due *NetworkNamespace* il programma crea e aggiunge ai relativi ad ognuno di essi diverse interfacce, una per ogni canale fisico simulato: per ogni interfaccia creata sul lato server ne viene creata una corrispondente sul lato client. Infine creata una pipe *Dummynet* per ogni coppia di interfacce in modo da rendere possibile il settaggio delle caratteristiche di trasmissione su ogni canale. Dopo l'esecuzione di *setup_env.sh* viene eseguito il programma *setup_pipe.sh* che permette, grazie all'utilizzo del software *Dummynet*, la configurazione dei parametri dei canali di comunicazione precedentemente realizzati. In questo modo vengono ricreate diverse condizioni con lo scopo di testare il modulo di comunicazione in diverse situazioni.

4.3.6 Esecuzione dei programmi di test Client/Server

Dopo aver creato e settato la rete virtuale, è possibile eseguire i programmi di test *Client* e *Server* per verificare il funzionamento del modulo di comunicazione. In questo caso è stato realizzato lo script *run_test.sh* che si occupa del lancio dei due programmi e della loro della creazione di una pipeline di comunicazione, riportata in fig. 4.12, in base alle specifiche scelte dall'utente, comunicate al programme mediante il numero di profilo. I valori iniziali del plug-in RLNC sono letti dal file $config\_file1.txt$ e sono riportati nella tabella 4.1.

L'esecuzione dello script avviene tramite l'istruzione

#./run_test.sh -p NUM_PROFILO -d DURATA_TEST

dove NUM_PROFILO è il numero del profilo scelto dall'utente in base alle specifiche riportate nel file *opt.h* e DURATA_TEST è la durata del test espressa in secondi.

Lo script manda in esecuzione il *Server* con il profilo prescelto sull'apposito *NetworkNamespace* precedentemente creato, attende due secondi e manda in esecuzione il *Client*, con il medesimo profilo del Server, nell'apposito *NetworkNamespace*. I programmi così lanciati andranno a creare uno streaming video utilizzando la rete simulata dando la possibilità di verificare visivamente il funzionamento del modulo di comunicazione. Nel caso in cui

Nome	Valore
k	10
n	$\geq k$ - come in eq. (4.4)
α	0.4
z	20

Tabella 4.1: Valori attribuiti alle variabili nella fase di test.

venga selezionata la riproduzione su monitor, viene aperta una finestra di riproduzione in cui viene visualizzato il video ricevuto sul lato client: in questo modo è possibile verificare in tempo reale l'eventuale presenza di artefatti, interruzioni e fluidità della riproduzione.

4.3.7 Test

Il test del sistema si basa principalmente sul controllo visivo della qualità del flusso video ricevuto. Oltre al controllo della qualità della trasmissione abbiamo monitorato il variare dell'ammontare dei dati trasmessi sulle diverse interfacce utilizzando Speedometer: in questo modo è stato possibile monitorare i canali di comunicazione in maniera tale da poter verificare quanto e come un canale viene effettivamente utilizzato.
Con l'utilizzo del software *Dummynet* abbiamo simulato diverse condizioni della rete, agendo principalmente sul parametro PLR. In questo modo è stato possibile verificare il corretto funzionamento del plug-in RLNC e di verificare le sue prestazioni: con la configurazione attuale il plug-in può compensare le perdite, su un canale con un tasso di perdita di circa il 20%. Tra i lavori futuri è prevista l'ottimizzazione della funzione utilizzata per il calcolo della ridondanza generata, in modo da innalzare il massimo tasso di perdita compensabile, migliorare i tempi di reazione, migliorare la precisione e soprattutto ridurre il più possibile la latenza dovuta ai tempi di bufferizzazione e codifica. Un'altro sviluppo futuro prevede di un buffer di codifica con dimensione che varia in base alla percentuale di flusso assegnata ad ogni canale: in questo modo sarà possibile sfruttare al meglio il plug-in RLNC riducendo i tempi di bufferizzazione.

	Throughput	**High Quality**	**Congestion Control**	**Reliability**	**Packet Skew**	**Security**
IETF draft in [2]	√	√	√	√	√	?
Plug-in MP-RTP in [3]	√	*X*	√	*X*	*X*	*X*
estensione in [13]	√	√	√	◯	◯	*X*
estensione corrente	√	√	√	√	◯	√

Tabella 4.2: Funzionalità implementate nelle diverse versioni

Legenda:
√ : implementato
X : non implementato
◯ : parzialmente implementato
? : non ancora definito nell'IETF draft [2]

Nella tabella 4.2 vengono riportate le funzionalità implementate nelle diverse versioni del modulo di comunicazione multipath:

- **Throughput:** incremento del throughput;
- **High Quality:** strategie per garantire una elevata QoS/QoE del flusso video in ricezione;
- **Congestion Control:** riduzione del carico che insiste su ciascun canale fisico, al fine di evitare fenomeni di congestione;
- **Reliability:** tecniche a rilevazione e correzione d'errore per la gestione delle perdite;
- **Packet Skew:** gestione della variazione del tempo di interarrivo dei pacchetti;
- **Security:** codifica dei dati e resistenza alle intercettazioni.

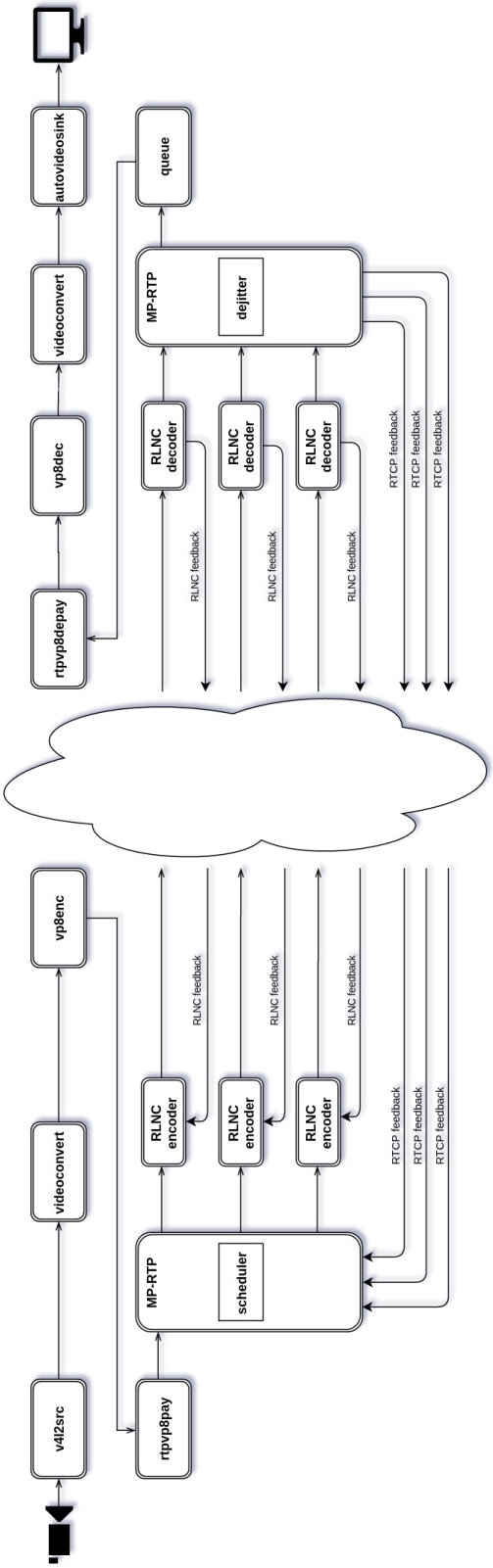

Figura 4.12: Struttura della pipeline di trasmissione e principali componenti. Da questa immagine è possibile notare in che posizione sono stati inseriti i moduli *Encoder* e *Decoder* RLNC e i relativi canali di $feedback$

Conclusioni

L'obiettivo di questo lavoro è stato l'implementazione di un modulo di comunicazione multipath orientato alla trasmissione di flussi multimediali real-time, tramite l'integrazione di tecniche RLNC. Per il raggiungimento dell'obiettivo, è stato sviluppato un plug-in Gstreamer che permette l'utilizzo di tecniche RLNC sfruttando l'implementazione offerta dalle librerie Kodo [10]. Per ogni canale fisico utilizzato, il plug-in permette di generare una quantità di dati ridondanti proporzionata al tasso di perdita rilevato sul canale stesso, allo scopo di garantire, dei dati trasmessi, la protezione da errori nel canale radio. Ho implementato due componenti principali, un *encoder* e un *decoder*, inseriti rispettivamente al lato sender e al lato receiver di ogni canale di comunicazione. I due componenti sviluppati si avvalgono di un *anello di feedback*, realizzato con una coppia sender-receiver UDP. Attraverso tale anello,il decoder invia all'encoder i dati relativi al tasso di perdita di ciascun canale fisico in uso. In questo modo, l'encoder RLNC calcola l'opportuna quantità di ridondanza e offre un encrypting nativo dei dati. Sono qui sottolineati due punti di forza del plug-in RLNC sviluppato: *modularità* e possibilità di *riuso*. Grazie a queste caratteristiche, il plug-in è utilizzabile in una qualsiasi pipeline RTP Gstreamer, senza porre come precondizione l'utilizzo di tecniche multipath, come nello scenario considerato in questo lavoro.

L'implementazione fornita riduce l'attuale gap fra l'implementazione esistente di MP-RTP [3] e l'IETF draft di riferimento [2].

Appendice A

Appendice

A.1 gstmykodoenc.h

```
/*
 * GStreamer
 * Copyright (C) 2005 Thomas Vander Stichele <thomas@apestaart.org>
 * Copyright (C) 2005 Ronald S. Bultje <rbultje@ronald.bitfreak.net>
 * Copyright (C) 2016 claudio <<user@hostname.org>>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

```
19   * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
       EVENT SHALL THE
20   * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
       DAMAGES OR OTHER
21   * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
       OTHERWISE, ARISING
22   * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER
23   * DEALINGS IN THE SOFTWARE.
24   *
25   * Alternatively, the contents of this file may be used under
       the
26   * GNU Lesser General Public License Version 2.1 (the "LGPL"),
       in
27   * which case the following provisions apply instead of the
       ones
28   * mentioned above:
29   *
30   * This library is free software; you can redistribute it and/
       or
31   * modify it under the terms of the GNU Library General Public
32   * License as published by the Free Software Foundation; either
33   * version 2 of the License, or (at your option) any later
       version.
34   *
35   * This library is distributed in the hope that it will be
       useful,
36   * but WITHOUT ANY WARRANTY; without even the implied warranty
       of
37   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
       the GNU
38   * Library General Public License for more details.
39   *
40   * You should have received a copy of the GNU Library General
       Public
41   * License along with this library; if not, write to the
42   * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
43   * Boston, MA 02111-1307, USA.
44   */
45  /*
46   * Author: Claudio Fadda <fadda.cla@gmail.com>
47   */
48  #ifndef __GST_MYKODOENC_H__
49  #define __GST_MYKODOENC_H__
50
```

```
51 #include <gst/gst.h>
52 #include "kodoc/kodoc.h"
53
54 G_BEGIN_DECLS
55
56 /* #defines don't like whitespacey bits */
57 #define GST_TYPE_MYKODOENC \
58         (gst_my_kodo_enc_get_type())
59 #define GST_MYKODOENC(obj) \
60         (G_TYPE_CHECK_INSTANCE_CAST((obj),GST_TYPE_MYKODOENC,
            GstMyKodoEnc))
61 #define GST_MYKODOENC_CLASS(klass) \
62         (G_TYPE_CHECK_CLASS_CAST((klass),GST_TYPE_MYKODOENC,
            GstMyKodoEncClass))
63 #define GST_IS_MYKODOENC(obj) \
64         (G_TYPE_CHECK_INSTANCE_TYPE((obj),GST_TYPE_MYKODOENC))
65 #define GST_IS_MYKODOENC_CLASS(klass) \
66         (G_TYPE_CHECK_CLASS_TYPE((klass),GST_TYPE_MYKODOENC))
67
68 typedef struct _GstMyKodoEnc      GstMyKodoEnc;
69 typedef struct _GstMyKodoEncClass GstMyKodoEncClass;
70
71 struct _GstMyKodoEnc
72 {
73     GstElement element;
74
75     GstPad *sinkpad,*sinkpad_2, *srcpad;
76
77     gboolean silent;
78
79     /* For KODO */
80     int seq_num;
81     int flag;
82     void **src;
83     gint16 buffer_count;
84     GstBuffer *buffertemp[20];
85
86     /* packet pointers */
87
88
89     // Variables needed for the coding
90     uint32_t max_symbols;
91     uint32_t max_symbol_size;
92
93     uint32_t symbols;
```

```
94      uint32_t packets;
95
96      int32_t codec;
97      int32_t finite_field;
98
99      kodoc_factory_t encoder_factory;
100     kodoc_coder_t encoder;
101
102     // The buffer sent to the receiver
103     uint32_t payload_size;
104     uint8_t* payload;
105     uint32_t bytes_used;
106
107     // The data to be encoded
108     uint32_t block_size;
109     uint8_t* data_in;
110
111     guint dim_array_PLR;
112     guint array_count;
113     gdouble *array_PLR;
114     gboolean array_is_full;
115     gdouble alpha;
116 };
117
118 struct _GstMyKodoEncClass
119 {
120     GstElementClass parent_class;
121 };
122
123 GType gst_my_kodo_enc_get_type (void);
124
125 G_END_DECLS
126
127 #endif
```

A.2 gstmykodoenc.c

```
1  /*
2   * GStreamer
3   * Copyright (C) 2005 Thomas Vander Stichele <thomas@apestaart.
       org>
4   * Copyright (C) 2005 Ronald S. Bultje <rbultje@ronald.bitfreak
       .net>
```

```
5   * Copyright (C) 2016 claudio <<user@hostname.org>>
6   *
7   * Permission is hereby granted, free of charge, to any person
        obtaining a
8   * copy of this software and associated documentation files (
        the "Software"),
9   * to deal in the Software without restriction, including
        without limitation
10  * the rights to use, copy, modify, merge, publish, distribute,
         sublicense,
11  * and/or sell copies of the Software, and to permit persons to
         whom the
12  * Software is furnished to do so, subject to the following
        conditions:
13  *
14  * The above copyright notice and this permission notice shall
        be included in
15  * all copies or substantial portions of the Software.
16  *
17  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR
18  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
        MERCHANTABILITY,
19  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
        EVENT SHALL THE
20  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
        DAMAGES OR OTHER
21  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
        OTHERWISE, ARISING
22  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
        OR OTHER
23  * DEALINGS IN THE SOFTWARE.
24  *
25  * Alternatively, the contents of this file may be used under
        the
26  * GNU Lesser General Public License Version 2.1 (the "LGPL"),
        in
27  * which case the following provisions apply instead of the
        ones
28  * mentioned above:
29  *
30  * This library is free software; you can redistribute it and/
        or
31  * modify it under the terms of the GNU Library General Public
32  * License as published by the Free Software Foundation; either
```

```
33   * version 2 of the License, or (at your option) any later
         version.
34   *
35   * This library is distributed in the hope that it will be
         useful,
36   * but WITHOUT ANY WARRANTY; without even the implied warranty
         of
37   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
         the GNU
38   * Library General Public License for more details.
39   *
40   * You should have received a copy of the GNU Library General
         Public
41   * License along with this library; if not, write to the
42   * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
43   * Boston, MA 02111-1307, USA.
44   */
45
46  /*
47   * Author: Claudio Fadda <fadda.cla@gmail.com>
48   */
49
50  #ifdef HAVE_CONFIG_H
51  #  include <config.h>
52  #endif
53
54  #include <stdlib.h>
55  #include <string.h>
56  #include <stdio.h>
57
58
59  #include "gstmykodoenc.h"
60
61  GST_DEBUG_CATEGORY_STATIC (gst_my_kodo_enc_debug);
62  #define GST_CAT_DEFAULT gst_my_kodo_enc_debug
63  #define PACKET_IS_RTP_OR_RTCP(b) (b > 0x7f && b < 0xc0)
64  #define PACKET_IS_RTCP(b) (b > 192 && b < 223)
65
66  /* coder signals and args */
67  enum
68  {
69      /* FILL ME */
70      LAST_SIGNAL
71  };
72
```

```
73 enum
74 {
75     PROP_0,
76     PROP_SILENT
77 };
78
79 /* the capabilities of the inputs and outputs.
80  *
81  * describe the real formats here.
82  */
83 static GstStaticPadTemplate sink_factory =
      GST_STATIC_PAD_TEMPLATE ("sink",
84         GST_PAD_SINK,
85         GST_PAD_ALWAYS,
86         GST_STATIC_CAPS ("ANY")
87 );
88
89 static GstStaticPadTemplate src_factory =
      GST_STATIC_PAD_TEMPLATE ("src",
90         GST_PAD_SRC,
91         GST_PAD_ALWAYS,
92         GST_STATIC_CAPS ("ANY")
93 );
94
95 #define gst_my_kodo_enc_parent_class parent_class
96 G_DEFINE_TYPE (GstMyKodoEnc, gst_my_kodo_enc, GST_TYPE_ELEMENT)
      ;
97
98 static void gst_my_kodo_enc_finalize (GObject * object);
99 static void gst_my_kodo_enc_set_property (GObject * object,
      guint prop_id,
100         const GValue * value, GParamSpec * pspec);
101 static void gst_my_kodo_enc_get_property (GObject * object,
      guint prop_id,
102         GValue * value, GParamSpec * pspec);
103
104 static gboolean gst_my_kodo_enc_sink_event (GstPad * pad,
      GstObject * parent, GstEvent * event);
105 static void Read_ConfigFile(const gchar* config,GstMyKodoEnc *
      coder);
106 static void Set_Variabile(gchar *nome, gdouble val,
      GstMyKodoEnc *coder);
107 static gdouble media_PLR (gdouble array[],GstMyKodoEnc *coder);
108 static GstFlowReturn gst_my_kodo_enc_chain (GstPad * pad,
      GstObject * parent, GstBuffer * buf);
```

```
109
110 //Caratteristiche file di configurazione
111 static guint lung_descr = 200;
112 static guint lung_nome = 50;
113 int margine =0;
114
115 /* GObject vmethod implementations   */
116
117 /* initialize the mykodoenc's class */
118 static void
119 gst_my_kodo_enc_class_init (GstMyKodoEncClass * klass)
120 {
121     GObjectClass *gobject_class;
122     GstElementClass *gstelement_class;
123
124     gobject_class = (GObjectClass *) klass;
125     gstelement_class = (GstElementClass *) klass;
126     gobject_class->finalize = gst_my_kodo_enc_finalize;
127
128     gobject_class->set_property = gst_my_kodo_enc_set_property;
129     gobject_class->get_property = gst_my_kodo_enc_get_property;
130
131     g_object_class_install_property (gobject_class, PROP_SILENT
        ,
132             g_param_spec_boolean ("silent", "Silent", "Produce
                verbose output ?",
133                  FALSE, G_PARAM_READWRITE));
134
135     gst_element_class_set_details_simple(gstelement_class,
136             "MyKodoEnc",
137             "Kodo coder",
138             "The plugin implements Kodo Linear Network Coding
                encoder"
139             "that auto set the numbrer of protection packets
                based on PLR",
140             "Claudio Fadda <<fadda.cla@gmail.com>>");
141
142     gst_element_class_add_pad_template (gstelement_class,
143             gst_static_pad_template_get (&src_factory));
144     gst_element_class_add_pad_template (gstelement_class,
145             gst_static_pad_template_get (&sink_factory));
146 }
147
148 /* initialize the new element
149  * instantiate pads and add them to element
```

```
150  * set pad calback functions
151  * initialize instance structure
152  */
153 static void
154 gst_my_kodo_enc_init (GstMyKodoEnc * coder)
155 {
156     coder->sinkpad = gst_pad_new_from_static_template (&
        sink_factory, "sink");
157     gst_pad_set_event_function (coder->sinkpad,
158             GST_DEBUG_FUNCPTR(gst_my_kodo_enc_sink_event));
159     gst_pad_set_chain_function (coder->sinkpad,
160             GST_DEBUG_FUNCPTR(gst_my_kodo_enc_chain));
161     GST_PAD_SET_PROXY_CAPS (coder->sinkpad);
162     gst_element_add_pad (GST_ELEMENT (coder), coder->sinkpad);
163
164     coder->sinkpad_2 = gst_pad_new_from_static_template (&
        sink_factory, "sink_2");
165     gst_pad_set_event_function (coder->sinkpad_2,
166             GST_DEBUG_FUNCPTR(gst_my_kodo_enc_sink_event));
167     gst_pad_set_chain_function (coder->sinkpad_2,
168             GST_DEBUG_FUNCPTR(gst_my_kodo_enc_chain));
169     GST_PAD_SET_PROXY_CAPS (coder->sinkpad_2);
170     gst_element_add_pad (GST_ELEMENT (coder), coder->sinkpad_2)
        ;
171
172     coder->srcpad = gst_pad_new_from_static_template (&
        src_factory, "src");
173     GST_PAD_SET_PROXY_CAPS (coder->srcpad);
174     gst_element_add_pad (GST_ELEMENT (coder), coder->srcpad);
175
176     coder->silent = FALSE;
177     /* inizializza variabili kodo*/
178
179     //Lettura dati dal file di configurazione
180     Read_ConfigFile("config_file1.txt",coder);
181
182     // Variables needed for the coding
183     coder->max_symbols = 32;
184     coder->max_symbol_size = 1420;
185
186     coder->symbols = 10;
187     coder->packets = 10;
188
189     coder->codec = kodoc_on_the_fly;
190     coder->finite_field = kodoc_binary8;
```

```
191
192     coder->encoder = 0;
193
194     // The buffer sent to the receiver
195     coder->payload_size = 0;
196     coder->payload = 0;
197     coder->bytes_used = 0;
198
199     // The data to be encoded
200     coder->block_size = 0;
201     coder->data_in = 0;
202
203
204     coder->encoder_factory = kodoc_new_encoder_factory(coder->
            codec, coder->finite_field,
205                 coder->max_symbols, coder->max_symbol_size);
206
207     kodoc_factory_set_symbols(coder->encoder_factory, coder->
            symbols);
208
209     coder->encoder = kodoc_factory_build_coder(coder->
            encoder_factory);
210     kodoc_set_systematic_on(coder->encoder);
211     coder->payload_size = kodoc_payload_size(coder->encoder);
212     coder->payload = (uint8_t*)g_malloc(coder->payload_size);
213     // Create some data to encode
214     coder->block_size = kodoc_block_size(coder->encoder);
215
216     coder->src = malloc(coder->symbols * sizeof(void *));
217
218     coder->buffer_count=0;
219     coder->flag=0;
220     coder->seq_num = 0;
221     coder->dim_array_PLR = 50;
222     coder->array_count = 0;
223     coder->array_PLR = (gdouble *) malloc(coder->dim_array_PLR*
            sizeof(gdouble));
224     coder->array_is_full = FALSE;
225
226 }
227
228 static void
229 gst_my_kodo_enc_finalize (GObject * object)
230 {
231     GstMyKodoEnc * coder = GST_MYKODOENC (object);
```

```
232     g_free(coder->src);
233     kodoc_delete_coder(coder->encoder);
234     kodoc_delete_factory(coder->encoder_factory);
235 }
236 
237 
238 static void
239 gst_my_kodo_enc_set_property (GObject * object, guint prop_id,
240         const GValue * value, GParamSpec * pspec)
241 {
242     GstMyKodoEnc *coder = GST_MYKODOENC (object);
243 
244     switch (prop_id) {
245     case PROP_SILENT:
246         coder->silent = g_value_get_boolean (value);
247         break;
248     default:
249         G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
               pspec);
250         break;
251     }
252 }
253 
254 static void
255 gst_my_kodo_enc_get_property (GObject * object, guint prop_id,
256         GValue * value, GParamSpec * pspec)
257 {
258     GstMyKodoEnc *coder = GST_MYKODOENC (object);
259 
260     switch (prop_id) {
261     case PROP_SILENT:
262         g_value_set_boolean (value, coder->silent);
263         break;
264     default:
265         G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
               pspec);
266         break;
267     }
268 }
269 
270 /* GstElement vmethod implementations */
271 
272 /* this function handles sink events */
273 static gboolean
```

```
274 gst_my_kodo_enc_sink_event (GstPad * pad, GstObject * parent,
      GstEvent * event)
275 {
276     GstMyKodoEnc *coder;
277     gboolean ret;
278 
279     coder = GST_MYKODOENC (parent);
280 
281     GST_LOG_OBJECT (coder, "Received %s event: %"
         GST_PTR_FORMAT,
282          GST_EVENT_TYPE_NAME (event), event);
283 
284     switch (GST_EVENT_TYPE (event)) {
285     case GST_EVENT_CAPS:
286     {
287         GstCaps * caps;
288 
289         gst_event_parse_caps (event, &caps);
290 
291         ret = gst_pad_event_default (pad, parent, event);
292         break;
293     }
294     default:
295         ret = gst_pad_event_default (pad, parent, event);
296         break;
297     }
298     return ret;
299 }
300 
301 
302 static void Read_ConfigFile(const gchar* config,GstMyKodoEnc *
      coder){
303     FILE *fd;
304     gchar *stringa,*nome,segno;
305     gdouble val;
306 
307     //Apertura file di configurazione
308     fd=fopen(config, "r");
309     if(fd==NULL)
310     {
311         perror("Errore in apertura del file");
312         exit(1);
313     }
314 
315     //Allocazione memoria per le stringhe
```

```
316     stringa = (gchar *) malloc(lung_descr*sizeof(gchar));
317     if(stringa == NULL){
318         fclose(fd);
319         perror("Errore in allocazione memoria");
320         exit(1);
321     }
322     nome = (gchar *) malloc(lung_nome*sizeof(gchar));
323     if(nome == NULL){
324         free(stringa);
325         fclose(fd);
326         perror("Errore in allocazione memoria");
327         exit(1);
328     }
329 
330     //Elimino la prima riga(descrizione) e gestisco la seconda
331     while(fgets(stringa,lung_descr,fd)!=NULL){
332         if(fgets(stringa,lung_nome,fd)!=NULL){
333             sscanf(stringa,"%s %c %lf",nome,&segno,&val);
334             //Vado a inserire il valore nella variabile giusta
335             Set_Variabile(nome,val,coder);
336         }
337     }
338     free(stringa);
339     free(nome);
340     fclose(fd);
341 }
342 
343 
344 
345 static void Set_Variabile(gchar *nome, gdouble val,GstMyKodoEnc
        *coder){
346     if(strcmp(nome,"alpha")==0){
347         coder->alpha = val;
348         return;
349     }
350     if(strcmp(nome,"kpackets")==0){
351         coder->symbols = val;
352         g_print("\npackettss kkk");
353         return;
354     }
355     if(strcmp(nome,"npackets")==0){
356         coder->packets = val;
357         g_print("\npakkeeess nnn");
358         return;
359     }
```

```
360     if(strcmp(nome,"dim_array_PLR")==0){
361         coder->dim_array_PLR = val;
362         return;
363     }
364 }
365
366
367
368 static gdouble media_PLR (gdouble array[],GstMyKodoEnc *coder){
369     gdouble somma = 0;
370
371     for(int i=0;i<coder->dim_array_PLR;i++){
372         somma = somma+array[i];
373     }
374     return somma/(coder->dim_array_PLR);
375 }
376
377
378 /* chain function
379  * this function does the actual processing
380  */
381 static GstFlowReturn
382 gst_my_kodo_enc_chain (GstPad * pad, GstObject * parent,
       GstBuffer * buf)
383 {
384     GstMyKodoEnc *coder;
385
386     coder = GST_MYKODOENC (parent);
387
388     GstFlowReturn result=GST_FLOW_OK;
389     GstBuffer *tempbuf1,*tempbuf2;
390     GstMapInfo map1;
391     gsize diff=0;
392     guint8 first_byte;
393     gdouble rcvdpkts;
394     gdouble lostpkts;
395     gdouble media;
396     gdouble plr;
397
398     if (gst_buffer_extract (buf, 0, &first_byte, 1) != 1 ){
399         g_print("\nBytes not extracted");
400     }
401     /* Se il pacchetto ricevuto non è rtp sarà un report (kodo)
            sulla PLR */
402     if (!PACKET_IS_RTP_OR_RTCP (first_byte)) {
```

```
403         gst_buffer_map(buf,&map1,GST_MAP_READWRITE);
404         rcvdpkts= GST_READ_UINT16_BE(map1.data);
405         gst_buffer_unmap(buf,&map1);
406 
407         rcvdpkts = (coder->packets-rcvdpkts);
408         lostpkts = (rcvdpkts/coder->packets);
409         if(!coder->array_is_full){
410             coder->array_PLR[coder->array_count] = lostpkts;
411             coder->array_count++;
412             if(coder->array_count == coder->dim_array_PLR)
                    coder->array_is_full = TRUE;
413         }else{
414             media = media_PLR (coder->array_PLR,coder);
415             plr = ((1-coder->alpha)*media) + (coder->alpha*
                    lostpkts);
416             coder->packets = ((coder->symbols / (1-plr)));
417             coder->array_PLR[coder->array_count] = lostpkts;
418             coder->array_count = (coder->array_count+1)%coder->
                    dim_array_PLR;
419         }
420     }
421     else{
422 
423         /* first packet*/
424         if (coder->flag++==0) {
425             return gst_pad_push (coder->srcpad,buf);
426         }
427 
428         /*
                ----------------------------------------------------------------
                */
429         coder->buffertemp[coder->buffer_count] =
                gst_buffer_copy(buf);
430         coder->buffer_count++;
431 
432         /* dopo aver collezionato k pacchetti parte la procdura
                di codifica*/
433         if(coder->buffer_count >= coder->symbols){
434             /*Ciclo per la creazione di un'array di pacchetti
                    della stessa dimensione*/
435             for(guint i=0; i< coder->symbols;i++){
436                 /*alloca   buffer temporaneo*/
437                 tempbuf2 = gst_buffer_new_allocate (NULL,2,NULL
                        );
438 
```

```
439                 /*scrive su tempbuf2 la lunghezza del buffer
                        originario*/
440                 gst_buffer_map (tempbuf2,&map1, GST_MAP_WRITE);
441                 GST_WRITE_UINT16_BE (map1.data,
                        gst_buffer_get_size(coder->buffertemp[i]));
442                 gst_buffer_unmap (tempbuf2, &map1);
443
444                 /*aggiunge in testa a bufftemp[i] la sua
                        lunghezza*/
445                 coder->buffertemp[i] = gst_buffer_append (
                        tempbuf2,coder->buffertemp[i]);
446
447                 /*Calcola la lunghezza del padding e lo
                        aggiunge in coda */
448                 diff = coder->max_symbol_size-
                        gst_buffer_get_size(coder->buffertemp[i]);
449                 tempbuf2 = gst_buffer_new_allocate (NULL,diff,
                        NULL);
450                 gst_buffer_memset(tempbuf2,0,0,diff);/*
                        inserisce padding 0*/
451                 coder->buffertemp[i] = gst_buffer_append (coder
                        ->buffertemp[i],tempbuf2);
452
453                 /* Ora ho un'array di buffers della stessa
                        dimensione*/
454             }
455             /*
456              * A questo punto sono stati collezionati symbols(k
                    ) buffers e sono stati portati tutti a
                    dimensione fissa di 1420bytes
457              *
458              * */
459             /*fa puntare gli elementi di src[] ai campi data di
                    buffertemp[]*/
460             for(int i=0;i<coder->symbols;i++){
461                 gst_buffer_map(coder->buffertemp[i],&map1,
                        GST_MAP_READWRITE);
462                 coder->src[i]=map1.data;
463                 gst_buffer_unmap(coder->buffertemp[i],&map1);
464             }
465             // KODO Send data
466             for (int i = 0; i < coder->packets/*packets13*/; ++
                    i)
467             {
468                 /*aggiunge i k simboli all'encoder */
```

```
469                 if (kodoc_rank(coder->encoder) < kodoc_symbols(
                        coder->encoder)/**/)
470                 {
471                     // The rank of an encoder indicates how many
                            symbols have
472                     // been added, i.e.  how many symbols are available
                            for encoding
473                     uint32_t rank = kodoc_rank(coder->encoder);
474                     // Calculate the offset to the next symbol to
                            insert
475 
476                     kodoc_set_const_symbol(coder->encoder, rank
                            , coder->src[rank]/*symbol*/,
                            kodoc_symbol_size(coder->encoder));
477                 }
478                 /*crea e scrive un pacchetto codificato in
                        payload*/
479                 coder->bytes_used = kodoc_write_payload(coder->
                        encoder, coder->payload);
480                 /*allocazione e riempimento buffer contenente
                        payload */
481                 tempbuf1 = gst_buffer_new_allocate (NULL,coder
                        ->bytes_used,NULL);
482                 gst_buffer_fill (tempbuf1,0,coder->payload,
                        coder->bytes_used);
483 
484                 /*Inserimento del numero di sequenza in un
                        nuovo buffer */
485                 tempbuf2 = gst_buffer_new_allocate (NULL,2,NULL
                        );
486                 gst_buffer_map (tempbuf2,&map1, GST_MAP_WRITE);
487                 GST_WRITE_UINT16_BE (map1.data, coder->seq_num)
                        ;
488                 gst_buffer_unmap (tempbuf2, &map1);
489                 tempbuf1 = gst_buffer_append (tempbuf2,tempbuf1
                        );
490                 result=gst_pad_push (coder->srcpad, tempbuf1);
491 
492             }
493             kodoc_delete_coder(coder->encoder);
494             coder->encoder = kodoc_factory_build_coder(coder->
                    encoder_factory);
495             kodoc_set_systematic_on(coder->encoder);
496             coder->payload_size = kodoc_payload_size(coder->
                    encoder);
```

```
497             coder->block_size = kodoc_block_size(coder->encoder
                    );
498             coder->buffer_count=0;
499             coder->seq_num=(coder->seq_num+1)%10;
500         }
501
502     }
503     return result;
504 }
505
506
507 /* entry point to initialize the plug-in
508  * initialize the plug-in itself
509  * register the element factories and other features
510  */
511 static gboolean
512 mykodoenc_init (GstPlugin * mykodoenc)
513 {
514     GST_DEBUG_CATEGORY_INIT (gst_my_kodo_enc_debug, "mykodoenc"
            ,
515             0, "mykodoenc");
516
517     return gst_element_register (mykodoenc, "mykodoenc",
            GST_RANK_NONE,
518             GST_TYPE_MYKODOENC);
519 }
520
521 /* PACKAGE: this is usually set by autotools depending on some
        _INIT macro
522  * in configure.ac and then written into and defined in config.
        h, but we can
523  * just set it ourselves here in case someone doesn't use
        autotools to
524  * compile this code. GST_PLUGIN_DEFINE needs PACKAGE to be
        defined.
525  */
526 #ifndef PACKAGE
527 #define PACKAGE "myfirstmykodoenc"
528 #endif
529
530 /* gstreamer looks for this structure to register mykodoencs
531  *
532  */
533 GST_PLUGIN_DEFINE (
534         GST_VERSION_MAJOR,
```

```
535         GST_VERSION_MINOR,
536         mykodoenc,
537         "mykodoenc",
538         mykodoenc_init,
539         VERSION,
540         "LGPL",
541         "GStreamer",
542         "http://gstreamer.net/"
543 )
```

A.3 gstmykododec.h

```
1  /*
2   * GStreamer
3   * Copyright (C) 2005 Thomas Vander Stichele <thomas@apestaart.
       org>
4   * Copyright (C) 2005 Ronald S. Bultje <rbultje@ronald.bitfreak
       .net>
5   * Copyright (C) 2016 claudio <<user@hostname.org>>
6   *
7   * Permission is hereby granted, free of charge, to any person
       obtaining a
8   * copy of this software and associated documentation files (
       the "Software"),
9   * to deal in the Software without restriction, including
       without limitation
10  * the rights to use, copy, modify, merge, publish, distribute,
        sublicense,
11  * and/or sell copies of the Software, and to permit persons to
        whom the
12  * Software is furnished to do so, subject to the following
       conditions:
13  *
14  * The above copyright notice and this permission notice shall
       be included in
15  * all copies or substantial portions of the Software.
16  *
17  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
       KIND, EXPRESS OR
18  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
       MERCHANTABILITY,
19  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
       EVENT SHALL THE
```

```
20  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
       DAMAGES OR OTHER
21  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
       OTHERWISE, ARISING
22  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER
23  * DEALINGS IN THE SOFTWARE.
24  *
25  * Alternatively, the contents of this file may be used under
       the
26  * GNU Lesser General Public License Version 2.1 (the "LGPL"),
       in
27  * which case the following provisions apply instead of the
       ones
28  * mentioned above:
29  *
30  * This library is free software; you can redistribute it and/
       or
31  * modify it under the terms of the GNU Library General Public
32  * License as published by the Free Software Foundation; either
33  * version 2 of the License, or (at your option) any later
       version.
34  *
35  * This library is distributed in the hope that it will be
       useful,
36  * but WITHOUT ANY WARRANTY; without even the implied warranty
       of
37  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
       the GNU
38  * Library General Public License for more details.
39  *
40  * You should have received a copy of the GNU Library General
       Public
41  * License along with this library; if not, write to the
42  * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
43  * Boston, MA 02111-1307, USA.
44  */
45 /*
46  * Author: Claudio Fadda <fadda.cla@gmail.com>
47  */
48
49 #ifndef __GST_MYKODODEC_H__
50 #define __GST_MYKODODEC_H__
51
52 #include <gst/gst.h>
```

```
53 #include "kodoc/kodoc.h"
54
55
56 G_BEGIN_DECLS
57
58 /* #defines don't like whitespacey bits */
59 #define GST_TYPE_MYKODODEC \
60         (gst_my_kodo_dec_get_type())
61 #define GST_MYKODODEC(obj) \
62         (G_TYPE_CHECK_INSTANCE_CAST((obj),GST_TYPE_MYKODODEC,
            GstMyKodoDec))
63 #define GST_MYKODODEC_CLASS(klass) \
64         (G_TYPE_CHECK_CLASS_CAST((klass),GST_TYPE_MYKODODEC,
            GstMyKodoDecClass))
65 #define GST_IS_MYKODODEC(obj) \
66         (G_TYPE_CHECK_INSTANCE_TYPE((obj),GST_TYPE_MYKODODEC))
67 #define GST_IS_MYKODODEC_CLASS(klass) \
68         (G_TYPE_CHECK_CLASS_TYPE((klass),GST_TYPE_MYKODODEC))
69
70 typedef struct _GstMyKodoDec      GstMyKodoDec;
71 typedef struct _GstMyKodoDecClass GstMyKodoDecClass;
72
73 struct _GstMyKodoDec
74 {
75     GstElement element;
76
77     GstPad *sinkpad, *srcpad,*srcpad_2;
78
79     gboolean silent;
80     gsize seq_num;
81     // Variables needed for the coding
82     uint32_t max_symbols;
83     uint32_t max_symbol_size;
84
85     uint32_t symbols;
86
87     int32_t codec;
88     int32_t finite_field;
89
90     kodoc_factory_t decoder_factory;
91     kodoc_coder_t decoder;
92
93     // The buffer used to receive incoming packets
94     uint32_t payload_size;
95     uint8_t* payload;
```

```
96
97      // Keeps track of which symbols have been decoded
98      uint8_t* decoded;
99
100     uint32_t block_size;
101
102     uint8_t* data_out;
103     guint flag;
104     guint packetcount;
105     guint blockcount;
106 };
107
108 struct _GstMyKodoDecClass
109 {
110     GstElementClass parent_class;
111 };
112
113 GType gst_my_kodo_dec_get_type (void);
114
115 G_END_DECLS
116
117 #endif
```

A.4 gstmykododec.c

```
1  /*
2   * GStreamer
3   * Copyright (C) 2005 Thomas Vander Stichele <thomas@apestaart.
       org>
4   * Copyright (C) 2005 Ronald S. Bultje <rbultje@ronald.bitfreak
       .net>
5   * Copyright (C) 2016 claudio <<user@hostname.org>>
6   *
7   * Permission is hereby granted, free of charge, to any person
       obtaining a
8   * copy of this software and associated documentation files (
       the "Software"),
9   * to deal in the Software without restriction, including
       without limitation
10  * the rights to use, copy, modify, merge, publish, distribute,
        sublicense,
11  * and/or sell copies of the Software, and to permit persons to
       whom the
```

```
12  * Software is furnished to do so, subject to the following
        conditions:
13  *
14  * The above copyright notice and this permission notice shall
        be included in
15  * all copies or substantial portions of the Software.
16  *
17  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR
18  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
        MERCHANTABILITY,
19  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
        EVENT SHALL THE
20  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
        DAMAGES OR OTHER
21  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
        OTHERWISE, ARISING
22  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
        OR OTHER
23  * DEALINGS IN THE SOFTWARE.
24  *
25  * Alternatively, the contents of this file may be used under
        the
26  * GNU Lesser General Public License Version 2.1 (the "LGPL"),
        in
27  * which case the following provisions apply instead of the
        ones
28  * mentioned above:
29  *
30  * This library is free software; you can redistribute it and/
        or
31  * modify it under the terms of the GNU Library General Public
32  * License as published by the Free Software Foundation; either
33  * version 2 of the License, or (at your option) any later
        version.
34  *
35  * This library is distributed in the hope that it will be
        useful,
36  * but WITHOUT ANY WARRANTY; without even the implied warranty
        of
37  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
        the GNU
38  * Library General Public License for more details.
39  *
```

```
40  * You should have received a copy of the GNU Library General
       Public
41  * License along with this library; if not, write to the
42  * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
43  * Boston, MA 02111-1307, USA.
44  */
45
46 /**
47  * Author: Claudio Fadda <fadda.cla@gmail.com>
48  */
49
50 #ifdef HAVE_CONFIG_H
51 #  include <config.h>
52 #endif
53
54 #include <string.h>
55 #include <stdio.h>
56
57 #include "gstmykododec.h"
58
59 GST_DEBUG_CATEGORY_STATIC (gst_my_kodo_dec_debug);
60 #define GST_CAT_DEFAULT gst_my_kodo_dec_debug
61 #define PACKET_IS_RTP_OR_RTCP(b) (b > 0x7f && b < 0xc0)
62
63 /* Filter signals and args */
64 enum
65 {
66     /* FILL ME */
67     LAST_SIGNAL
68 };
69
70 enum
71 {
72     PROP_0,
73     PROP_SILENT
74 };
75
76 /* the capabilities of the inputs and outputs.
77  *
78  * describe the real formats here.
79  */
80 static GstStaticPadTemplate sink_factory =
       GST_STATIC_PAD_TEMPLATE ("sink",
81         GST_PAD_SINK,
82         GST_PAD_ALWAYS,
```

```
83          GST_STATIC_CAPS ("ANY")
84  );
85
86  static GstStaticPadTemplate src_factory =
        GST_STATIC_PAD_TEMPLATE ("src",
87          GST_PAD_SRC,
88          GST_PAD_ALWAYS,
89          GST_STATIC_CAPS ("ANY")
90  );
91
92  #define gst_my_kodo_dec_parent_class parent_class
93  G_DEFINE_TYPE (GstMyKodoDec, gst_my_kodo_dec, GST_TYPE_ELEMENT)
        ;
94
95  static void gst_my_kodo_dec_finalize (GObject * object);
96  static void gst_my_kodo_dec_set_property (GObject * object,
        guint prop_id,
97          const GValue * value, GParamSpec * pspec);
98  static void gst_my_kodo_dec_get_property (GObject * object,
        guint prop_id,
99          GValue * value, GParamSpec * pspec);
100
101 static gboolean gst_my_kodo_dec_sink_event (GstPad * pad,
        GstObject * parent, GstEvent * event);
102 static GstFlowReturn gst_my_kodo_dec_chain (GstPad * pad,
        GstObject * parent, GstBuffer * buf);
103
104 /* GObject vmethod implementations */
105
106 /* initialize the mykododec's class */
107 static void
108 gst_my_kodo_dec_class_init (GstMyKodoDecClass * klass)
109 {
110     GObjectClass *gobject_class;
111     GstElementClass *gstelement_class;
112
113     gobject_class = (GObjectClass *) klass;
114     gstelement_class = (GstElementClass *) klass;
115     gobject_class->finalize = gst_my_kodo_dec_finalize;
116
117     gobject_class->set_property = gst_my_kodo_dec_set_property;
118     gobject_class->get_property = gst_my_kodo_dec_get_property;
119
120     g_object_class_install_property (gobject_class, PROP_SILENT
            ,
```

```
121             g_param_spec_boolean ("silent", "Silent", "Produce
                    verbose output ?",
122                     FALSE, G_PARAM_READWRITE));
123
124     gst_element_class_set_details_simple(gstelement_class,
125             "MyKodoDec",
126             "Kodo Decoder",
127             "The plugin implements Kodo Linear Network Coding
                    decoder",
128             "Claudio Fadda <<fadda.cla@gmail.com>>");
129
130     gst_element_class_add_pad_template (gstelement_class,
131             gst_static_pad_template_get (&src_factory));
132     gst_element_class_add_pad_template (gstelement_class,
133             gst_static_pad_template_get (&sink_factory));
134 }
135
136 /* initialize the new element
137  * instantiate pads and add them to element
138  * set pad calback functions
139  * initialize instance structure
140  */
141 static void
142 gst_my_kodo_dec_init (GstMyKodoDec * filter)
143 {
144     filter->sinkpad = gst_pad_new_from_static_template (&
            sink_factory, "sink");
145     gst_pad_set_event_function (filter->sinkpad,
146             GST_DEBUG_FUNCPTR(gst_my_kodo_dec_sink_event));
147     gst_pad_set_chain_function (filter->sinkpad,
148             GST_DEBUG_FUNCPTR(gst_my_kodo_dec_chain));
149     GST_PAD_SET_PROXY_CAPS (filter->sinkpad);
150     gst_element_add_pad (GST_ELEMENT (filter), filter->sinkpad)
            ;
151
152     filter->srcpad = gst_pad_new_from_static_template (&
            src_factory, "src");
153     GST_PAD_SET_PROXY_CAPS (filter->srcpad);
154     gst_element_add_pad (GST_ELEMENT (filter), filter->srcpad);
155
156     filter->srcpad_2 = gst_pad_new_from_static_template (&
            src_factory, "src_2");
157     GST_PAD_SET_PROXY_CAPS (filter->srcpad_2);
158     gst_element_add_pad (GST_ELEMENT (filter), filter->srcpad_2
            );
```

```
159
160     filter->silent = FALSE;
161
162     // Variables needed for the coding
163     filter->max_symbols = 10;
164     filter->max_symbol_size = 1420;
165
166     filter->symbols = 10;
167
168     filter->codec = kodoc_on_the_fly;
169     filter->finite_field = kodoc_binary8;
170
171     filter->decoder_factory = kodoc_new_decoder_factory(filter
            ->codec, filter->finite_field,
172                 filter->max_symbols, filter->max_symbol_size);
173     filter->decoder = kodoc_factory_build_coder(filter->
            decoder_factory);
174
175     // The buffer used to receive incoming packets
176     filter->payload_size = kodoc_payload_size(filter->decoder);
177     filter->payload = (uint8_t*) g_malloc(filter->payload_size)
            ;
178
179     // Keeps track of which symbols have been decoded
180     filter->decoded = (uint8_t*) g_malloc(sizeof(uint8_t) *
            filter->max_symbols);
181
182     filter->block_size = kodoc_block_size(filter->decoder);
183
184     filter->data_out = (uint8_t*) g_malloc(filter->block_size);
185
186     // Zero initialize the decoded array */
187     memset(filter->decoded, '\0', sizeof(uint8_t) * filter->
            max_symbols);
188     filter->flag=0;
189     filter->seq_num=0;
190
191     filter->packetcount=0;
192     filter->blockcount=0;
193
194     kodoc_set_mutable_symbols(filter->decoder, filter->data_out
            , filter->block_size);
195
196 }
197
```

```
198 static void gst_my_kodo_dec_finalize (GObject * object){
199
200     GstMyKodoDec *filter = GST_MYKODODEC (object);
201
202     kodoc_delete_coder(filter->decoder);
203     kodoc_delete_factory(filter->decoder_factory);
204     g_free(filter->payload);
205 }
206
207 static void
208 gst_my_kodo_dec_set_property (GObject * object, guint prop_id,
209         const GValue * value, GParamSpec * pspec)
210 {
211     GstMyKodoDec *filter = GST_MYKODODEC (object);
212
213     switch (prop_id) {
214     case PROP_SILENT:
215         filter->silent = g_value_get_boolean (value);
216         break;
217     default:
218         G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
               pspec);
219         break;
220     }
221 }
222
223 static void
224 gst_my_kodo_dec_get_property (GObject * object, guint prop_id,
225         GValue * value, GParamSpec * pspec)
226 {
227     GstMyKodoDec *filter = GST_MYKODODEC (object);
228
229     switch (prop_id) {
230     case PROP_SILENT:
231         g_value_set_boolean (value, filter->silent);
232         break;
233     default:
234         G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id,
               pspec);
235         break;
236     }
237 }
238
239 /* GstElement vmethod implementations */
240
```

```
241 /* this function handles sink events */
242 static gboolean
243 gst_my_kodo_dec_sink_event (GstPad * pad, GstObject * parent,
       GstEvent * event)
244 {
245     GstMyKodoDec *filter;
246     gboolean ret;
247
248     filter = GST_MYKODODEC (parent);
249
250     GST_LOG_OBJECT (filter, "Received %s event: %"
           GST_PTR_FORMAT,
251          GST_EVENT_TYPE_NAME (event), event);
252
253     switch (GST_EVENT_TYPE (event)) {
254     case GST_EVENT_CAPS:
255     {
256         GstCaps * caps;
257
258         gst_event_parse_caps (event, &caps);
259         /* do something with the caps */
260
261         /* and forward */
262         ret = gst_pad_event_default (pad, parent, event);
263         break;
264     }
265     default:
266         ret = gst_pad_event_default (pad, parent, event);
267         break;
268     }
269     return ret;
270 }
271
272 /* chain function
273  * this function does the actual processing
274  */
275 static GstFlowReturn
276 gst_my_kodo_dec_chain (GstPad * pad, GstObject * parent,
       GstBuffer * buf)
277 {
278     GstMyKodoDec *filter;
279     guint seq_rcv;
280     guint8 *payload;
281     GstBuffer *tempbuf1=NULL;
282     GstMapInfo map;
```

```
283     GstFlowReturn result=GST_FLOW_OK;
284     guint16 newsize;
285
286     filter = GST_MYKODODEC (parent);
287
288     if(filter->flag++==0){
289         result = gst_pad_push (filter->srcpad, buf);
290     }
291     else
292     {
293         //copia in seqrcv il numero di seq, relativo al blocco, del
                datagramma ricevuto
294         payload = g_malloc(sizeof(guint8 *));
295         gst_buffer_map(buf,&map,GST_MAP_READWRITE);
296         seq_rcv= GST_READ_UINT16_BE(map.data);
297         gst_buffer_unmap(buf,&map);
298         gst_buffer_resize(buf,2,-1);
299
300         gst_buffer_map(buf,&map,GST_MAP_READWRITE);
301         payload=map.data;
302         gst_buffer_unmap(buf,&map);
303
304         //se il numero di seq del blocco attuale è uguale al numeroseq
                del simbolo ricevuto
305         //effettua la decodifica, altrimenti salta al prossimo
306         if(filter->seq_num==seq_rcv)
307         {
308             /*Aggiunge pacchetto ricevuto al decoder,
                    successivamente controlla e memorizza l'esito
                    dell'operazione*/
309             kodoc_read_payload(filter->decoder, payload);
310             /* qui si tiene solo traccia dei simboli
                    decodificati */
311             if (kodoc_has_partial_decoding_interface(filter->
                    decoder) && kodoc_is_partially_complete(filter->
                    decoder))
312             {
313                 for (int i=0; i < kodoc_symbols(filter->decoder
                        ); ++i)
314                 {
315                     if (!kodoc_is_symbol_uncoded(filter->
                            decoder, i))
316                         continue;
317                     if (!filter->decoded[i])
318                     {
319                         filter->decoded[i] = 1;
```

```
320                         tempbuf1 = gst_buffer_new_allocate (
                                NULL,filter->max_symbol_size,NULL);
321                         int ind = (i*kodoc_symbol_size(filter->
                                decoder));
322                         gst_buffer_fill (tempbuf1,0,filter->
                                data_out+ind,filter->max_symbol_size
                                );
323                         gst_buffer_map(tempbuf1,&map,
                                GST_MAP_READ);
324                         newsize = GST_READ_UINT16_BE(map.data);
325                         gst_buffer_unmap(tempbuf1,&map);
326                         gst_buffer_resize(tempbuf1,2,newsize);
327
328                         result = gst_pad_push (filter->srcpad,
                                tempbuf1);
329                     }
330                 }
331             }
332         }
333
334         /* se il blocco è stato decodificato o se è arrivato un
                simbolo appartenente al blocco successivo */
335
336         if(kodoc_is_complete(filter->decoder) || seq_rcv==(
                filter->seq_num+1)%10){
337             /*legge la dimensione originale del buffer e
                    spedisce */
338
339             kodoc_delete_coder(filter->decoder);
340             filter->decoder = kodoc_factory_build_coder(filter
                    ->decoder_factory);
341
342             // Create the buffer needed for the payload
343             filter->payload_size = kodoc_payload_size(filter->
                    decoder);
344
345             filter->block_size = kodoc_block_size(filter->
                    decoder);
346             kodoc_set_mutable_symbols(filter->decoder, filter->
                    data_out, filter->block_size);
347
348             // Zero initialize the decoded array */
349             memset(filter->data_out, '\0', filter->block_size);
350             memset(filter->decoded, '\0', sizeof(uint8_t) *
                    filter->max_symbols);
```

```
351
352             /*se l'ultimo pacchetto ricevuto appartiene al
                    blocco successivo invio il report e azzero il
                    contatore */
353
354             filter->seq_num=(filter->seq_num+1)%10;
355         }
356         /* se il pacchetto ricevuto appartiene al blocco
                attuale incrementa contatore pacchetti corretti */
357         if(seq_rcv == filter->blockcount) filter->packetcount
                ++;
358         /* altrimenti spedisci repport riportante il numero
                dipacchetti corretti ricevuti in questo blocco */
359         else{
360             tempbuf1 = gst_buffer_new_allocate (NULL,2,NULL);
361             gst_buffer_map (tempbuf1,&map, GST_MAP_WRITE);
362             GST_WRITE_UINT16_BE (map.data, filter->packetcount)
                    ;
363             gst_buffer_unmap (tempbuf1, &map);
364             result = gst_pad_push (filter->srcpad_2, tempbuf1);
365             filter->packetcount=1;
366             filter->blockcount=(filter->blockcount+1)%10;
367         }
368     }
369     return result;
370 }
371 /* entry point to initialize the plug-in
372  * initialize the plug-in itself
373  * register the element factories and other features
374  */
375 static gboolean
376 mykododec_init (GstPlugin * mykododec)
377 {
378     /* debug category for fltering log messages
379      *
380      */
381     GST_DEBUG_CATEGORY_INIT (gst_my_kodo_dec_debug, "mykododec"
            ,
382             0, "mykododec");
383
384     return gst_element_register (mykododec, "mykododec",
            GST_RANK_NONE,
385             GST_TYPE_MYKODODEC);
386 }
387
```

```
388 #ifndef PACKAGE
389 #define PACKAGE "myfirstmykododec"
390 #endif
391
392 /* gstreamer looks for this structure to register mykododecs
393  *
394  */
395 GST_PLUGIN_DEFINE (
396         GST_VERSION_MAJOR,
397         GST_VERSION_MINOR,
398         mykododec,
399         "mykododec",
400         mykododec_init,
401         VERSION,
402         "LGPL",
403         "GStreamer",
404         "http://gstreamer.net/"
405 )
```

A.5 servermod.c

```
1  #include <gst/gst.h>
2  #include <gst/rtp/rtp.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <glib.h>
7  #include "opt.h"
8
9  static GstElement *encoder;
10 int contatore = 0;
11 int secondo = 0;
12 FILE *fd;
13 //Indica il numero di sottoflussi attivi-
14 guint num_subflows = 0;
15
16 //Struttura contenente i dati che indicano un rapporto di utilizzo di un
       subflow
17 typedef struct _SubflowUtilization{
18     gboolean   controlled;
19     struct _SubflowUtilizationReport{
20         gint32   lost_bytes;
21         gint32   discarded_bytes;
22         gint32   target_rate;
```

```
23          gint32   sending_rate;
24          guint64  owd;
25          gdouble  rtt;
26          gint     state;
27      }report;
28      struct _SubflowUtilizationControl{
29          gint32   max_rate;
30          gint32   min_rate;
31      }control;
32  }SubflowUtilization;
33
34  //Struttura contenente il rapporto dell'utilizzo
35  typedef struct _MPRTPPluginUtilization{
36      struct{
37          gint32 target_rate;
38      }report;
39      struct{
40          gint32  max_rate,min_rate;
41          gdouble max_mtakover,max_stakover;
42      }control;
43      SubflowUtilization subflows[32];
44  }MPRTPPluginUtilization;
45
46  //Struttura dei dati usati per la modalità Join/Detach
47  typedef struct _JoinDetachData{
48      gboolean active_s1;
49      gboolean active_s2;
50      gboolean active_s3;
51      gboolean join_s1;
52      gboolean join_s2;
53      gboolean join_s3;
54      gboolean stop;
55      GstElement *mprtpsch;
56  }JoinDetachData;
57
58  //Struttura contenente i dati che caratterizzano una sessione
59  typedef struct _SessionData
60  {
61      int ref;
62      guint sessionNum;
63      GstElement *input;
64  } SessionData;
65
66
67
```

```
68
69
70
71  static JoinDetachData join_detach_data;
72
73  //Incremento parametro ref della sessione per riferimento
74  static SessionData *session_ref (SessionData * data)
75  {
76      g_atomic_int_inc (&data->ref);
77      return data;
78  }
79
80  //Libera memoria dalle strutture allocate per una sessione
81  static void session_unref (gpointer data)
82  {
83      SessionData *session = (SessionData *) data;
84      if (g_atomic_int_dec_and_test (&session->ref)) {
85          g_free (session);
86      }
87  }
88
89  //Creazione di una nuova sessione
90  static SessionData *session_new (guint sessionNum)
91  {
92      SessionData *ret = g_new0 (SessionData, 1);
93      ret->sessionNum = sessionNum;
94      return session_ref (ret);
95  }
96
97
98  //Genera messaggi informativi durante il funzionamento della pipeline
99  static void cb_state (GstBus * bus, GstMessage * message,
        gpointer data)
100 {
101     GstObject *pipe = GST_OBJECT (data);
102     GstState old, new, pending;
103     gst_message_parse_state_changed (message, &old, &new, &
            pending);
104     if (message->src == pipe) {
105         g_print ("Pipeline %s cambia stato da %s a %s\n",
106                 GST_OBJECT_NAME (message->src),
                        gst_element_state_get_name (old),
                        gst_element_state_get_name (new));
107     }
108 }
```

```
109
110 /* Crea un GstGhostPad chiamato "src" nel bin passato come
    argomento, collegato al pad "src" passato anch'esso come
    argomento*/
111 static void setup_ghost (GstElement * src, GstBin * bin)
112 {
113     GstPad *srcPad = gst_element_get_static_pad (src, "src");
114     if(srcPad == NULL)
115         g_error("Pad src non esistente");
116     GstPad *binPad = gst_ghost_pad_new ("src", srcPad);
117     gst_element_add_pad (GST_ELEMENT (bin), binPad);
118 }
119
120
121 //Creazione sessione v4l2src
122 static SessionData *make_video_vl2src_session (guint sessionNum
    )
123 {
124     GstBin *videoBin = GST_BIN (gst_bin_new (NULL));
125     //GstCaps *videoCaps;
126     SessionData *session;
127     gboolean res;
128     GstElement *videoSrc, *videoConv, *payloader;
129
130     //Creazione di un bin che conterrà tutti gli elementi
131     videoBin = GST_BIN (gst_bin_new (NULL));
132
133
134
135     //Creazione elemento v4l2src
136     videoSrc = gst_element_factory_make ("v4l2src", NULL);
137     g_assert(videoSrc);
138
139     //Creazione convertitore video
140     videoConv = gst_element_factory_make("videoconvert", NULL);
141     g_assert(videoConv);
142
143     //Creazione elemento per aggiungere payload RTP per formato VP8
144     payloader = gst_element_factory_make ("rtpvp8pay", NULL);
145     g_assert(videoConv);
146
147     //Creazione encoder per formato video VP8
148     encoder = gst_element_factory_make ("vp8enc", NULL);
149     g_assert(encoder);
150
```

```
151
152     //Setto proprietà encoder
153     //Bitrate mandato in uscita da encoder --> se siamo in auto rate
            questo verrà modificaton a seconda dei feedback ricevuti
154     g_object_set (encoder, "target-bitrate", 128000 *
            test_parameters_.subflow_num, NULL);
155     //Massima distanza tra i keyframe (in numero di frame)
156     g_object_set (encoder, "keyframe-max-dist", 20, NULL);
157     /*Modalità di controllo rate
158      *0 modalità Variable Bit Rate
159      *1 modalità Constant Bit Rate ->utile per streaming su
             canali di capacità limitata
160      *2 modalità Constant Quality Mode*/
161     g_object_set (encoder, "end-usage", 1, NULL);
162     //Scadenza di un frame
163     g_object_set (encoder, "deadline", 20000, NULL);
164     //Datarate undershoot (minimo)
165     g_object_set (encoder, "undershoot", 100, NULL);
166     g_object_set (encoder, "cpu-used", 2, NULL);
167     //keyframe-mode se messo a 0 non sceglie automaticamente dove mettere
            i keyframe
168
169     //Aggiungo gli elementi al bin creato
170     gst_bin_add_many (videoBin, videoConv, videoSrc, encoder,
            payloader, NULL);
171
172
173
174     //Collego i vari elementi tra di loro
175
176     res=gst_element_link (videoSrc, videoConv);
177     if(!res)
178         g_error("Impossibile collegare videoSrc a VideoConv");
179     res=gst_element_link (videoConv, encoder);
180     if(!res)
181         g_error("Impossibile collegare VideoConv ad encoder");
182     res=gst_element_link (encoder, payloader);
183     if(!res)
184         g_error("Impossibile collegare encoder a payloader");
185
186     /*
187     res=gst_element_link (payloader, myencoder);
188     if(!res)
189         g_error("Impossibile collegare payloader a myencoder");
190     //  g_object_set(videoSrc, "filter-caps", videoCaps, NULL);
191      */
```

```
192     //creazione di un pad fantasma sul bin creato usando quello del
            payloader
193     setup_ghost (payloader, videoBin);
194
195     //Creazione della sessione
196     session = session_new (sessionNum);
197     session->input = GST_ELEMENT (videoBin);
198     return session;
199 }
200
201
202 //Creazione sessione video con lettura da file
203 static SessionData *make_video_filevideo_session (guint
        sessionNum)
204 {
205     GstBin *videoBin;
206     GstElement *videoSrc,*identity,*videoParse,*videoConv,*
            payloader,*myfilter;
207     SessionData *session;
208
209     videoBin = GST_BIN (gst_bin_new (NULL));
210
211     //Creazione elemento per lettura da file
212     videoSrc = gst_element_factory_make ("multifilesrc", NULL);
213     g_assert(videoSrc);
214
215     //Passaggio dei dati senza modifiche
216     identity = gst_element_factory_make ("identity", NULL);
217     g_assert(identity);
218
219     //Converte lo stream in frame video
220     videoParse = gst_element_factory_make ("videoparse", NULL);
221     g_assert(videoParse);
222
223     //Mette video VP8 nel pacchetto RTP
224     payloader = gst_element_factory_make ("rtpvp8pay", NULL);
225     g_assert(payloader);
226
227     /*myfilter
228     myfilter = gst_element_factory_make ("myfilter", NULL);
229     g_assert(myfilter);
230      */
231
232     //Indico al multifilesrc dove deve prendere il file e che deve
            riprodurlo a ripetizione
233
```

```
234     g_object_set (videoSrc,
235             "location",  "video.yuv",
236             NULL);
237
238     //Codifica video VP8
239     encoder = gst_element_factory_make ("vp8enc", NULL);
240     g_assert(encoder);
241
242     //Setto proprietà per codifica del video
243     g_object_set (encoder, "target-bitrate", 128000 *
           test_parameters_.subflow_num, NULL);
244     g_object_set (encoder, "end-usage", 1, NULL);
245     g_object_set (encoder, "deadline", 20000, NULL);
246     g_object_set (encoder, "undershoot", 100, NULL);
247     g_object_set (encoder, "cpu-used", 0, NULL);
248
249     //Aggiungo elementi al bin
250     gst_bin_add_many (videoBin, videoSrc, identity, videoParse,
            encoder, payloader,  NULL);
251
252     //Caratteristiche video
253     g_object_set (videoParse,
254             "width", 640,
255             "height", 360,
256             "framerate", 25, 1,
257             "format", 2,
258             NULL);
259     /*original = TRUE, modified to FALSE for kodo */
260     g_object_set (identity, "sync", TRUE, NULL);
261
262     //COllego gli elementi tra di loro
263     gst_element_link (videoSrc, videoParse);
264     gst_element_link (videoParse, identity);
265     gst_element_link (identity, encoder);
266     gst_element_link (encoder, payloader);
267
268     setup_ghost (payloader, videoBin);
269     session = session_new (sessionNum);
270     session->input = GST_ELEMENT (videoBin);
271
272     return session;
273 }
274
275
276 //Creazione di rtprtxsend come elemento "aux" di rtpbin
```

```
277 static GstElement *request_aux_sender (GstElement * rtpbin,
        guint sessid, SessionData * session)
278 {
279     GstElement *rtx, *bin;
280     GstPad *pad;
281     gchar *name;
282 
283     GST_INFO ("Creazione AUX sender");
284     bin = gst_bin_new (NULL);
285     //Creazione elemento rtprtxsend
286     rtx = gst_element_factory_make ("rtprtxsend", NULL);
287     g_assert(rtx);
288 
289 
290 
291     //Aggiungo rtprtxsend al bin creato
292     gst_bin_add (GST_BIN (bin), rtx);
293 
294     //Creazione pad fantasma di tipo src per il bin
295     pad = gst_element_get_static_pad (rtx, "src");
296     if(pad == NULL)
297         g_error("Pad src non esistente");
298     name = g_strdup_printf ("src_%u", sessid);
299     if(name == NULL)
300         g_error("name: stringa vuota");
301     gst_element_add_pad (bin, gst_ghost_pad_new (name, pad));
302     g_free (name);
303     gst_object_unref (pad);
304 
305     //Creazione pad fantasma di tipo sink per il bin
306     pad = gst_element_get_static_pad (rtx, "sink");
307     if(pad == NULL)
308         g_error("Pad sink non esistente");
309     name = g_strdup_printf ("sink_%u", sessid);
310     if(name == NULL)
311         g_error("name: stringa vuota");
312     gst_element_add_pad (bin, gst_ghost_pad_new (name, pad));
313     g_free (name);
314     gst_object_unref (pad);
315 
316     return bin;
317 }
318 
319 
320 //Gestione rate per addattarsi alla rete
```

```
321 static void changed_event (GstElement * mprtp_sch, gpointer ptr
    )
322 {
323     MPRTPPluginUtilization *ur = ptr;
324     gint new_bitrate;
325     gint get_bitrate;
326     gint i;
327     g_object_get (encoder, "target-bitrate", &get_bitrate, NULL
        );
328     {
329 
330         new_bitrate = ur->report.target_rate;
331         for(i=0; i<32; ++i){
332             if(!ur->subflows[i].controlled) continue;
333             if(test_parameters_.video_session == FILE_VIDEO){
334                 ur->subflows[i].control.min_rate = 100000 /
                        test_parameters_.subflow_num;
335                 ur->subflows[i].control.max_rate = 300000;
336             }
337         }
338     }
339     secondo++;
340     fprintf(fd, "--------------------Controllo al cambiamento
        numero %d--------------------\n", secondo);
341     //Stampa dei dati
342     for(i=1; i<4; i++){
343         if(!ur->subflows[i].controlled) continue;
344         fprintf(fd,"Sottoflusso %d\n",i);
345         fprintf(fd,"Stato : ");
346         switch(ur->subflows[i].report.state){
347         case -1:
348             fprintf(fd,"Sovrautilizzato\n");
349             break;
350         case 0:
351             fprintf(fd,"Stabile\n");
352             break;
353         case 1:
354             fprintf(fd,"Sottoutilizzato\n");
355             break;
356         default:
357             break;
358         }
359         fprintf(fd,"Send Rate : %-10d  Byte persi : %-10d  Byte
                scartati: %-10d  OWD : %-10lu  RTT: %f\n",
```

```
360                 ur->subflows[i].report.sending_rate, ur->
                        subflows[i].report.lost_bytes,
361                 ur->subflows[i].report.discarded_bytes,ur->
                        subflows[i].report.owd,ur->subflows[i].
                        report.rtt);
362     }
363     g_print("get_bitrate: %d new_bitrate: %d\n", get_bitrate,
            new_bitrate);
364     g_object_set (encoder, "target-bitrate", new_bitrate, NULL)
            ;
365     return;
366 }
367
368
369
370 int called = 0;
371 //Setta bitrate dei flussi in modo casuale
372 static gboolean _random_rate_controller (gpointer data)
373 {
374     GstElement *mprtpsch;
375     gint subflow1_target, subflow2_target, subflow3_target;
376     gint cap = 1000000;
377
378     mprtpsch = data;
379     subflow1_target = g_random_int_range(100000, 500000);
380     subflow2_target = g_random_int_range(100000, 500000);
381     subflow3_target = cap - subflow1_target - subflow2_target;
382
383     if(called % 60 == 0){
384         //Setta il rate dei diversi sottoflussi
385         /*setup-sending-target vuole in ingresso 32 bit dove i
                primi 8 identificano il sottoflusso mentre gli altri
                 24
386         identificano il target*/
387         g_object_set (mprtpsch,
388                 "setup-sending-target", (1 << 24) |
                        subflow1_target,
389                 "setup-sending-target", (2 << 24) |
                        subflow2_target,
390                 "setup-sending-target", (3 << 24) |
                        subflow3_target,
391                 NULL);
392     }
393     if(++called < 600) goto go_on;
```

```
394     //macro usata per uscire dalla funzione eliminando la
            GSource(rappresenta una fonte di evento) nel loop principale
395     return G_SOURCE_REMOVE;
396     go_on:
397     //macro usata per uscire dalla funzione lasciando la
            GSource(rappresenta una fonte di evento) nel loop principale
398     return G_SOURCE_CONTINUE;
399 }
400
401
402 static gboolean _join_detach_test(gpointer ptr)
403 {
404     JoinDetachData *data = ptr;
405     GstElement *mprtpsch;
406     mprtpsch = data->mprtpsch;
407     switch(contatore)
408     {
409     //Attivo Flusso 1 e Disattivo Flusso 2/3
410     case 0:{
411         //Se è disattivato e fa parte del gruppo di quelli che fanno
                parte della sessione
412         if(!data->active_s1 && data->join_s1)
413         {
414             g_print("Activate Subflow 1 ...\n\n");
415             g_object_set (mprtpsch, "join-subflow", 1, NULL);
416             data->active_s1=TRUE;
417             num_subflows++;
418         }
419         //Se è attivo ma non è l'unico
420         if(data->active_s2 && num_subflows>1)
421         {
422             g_print("Disabling Subflow 2 ...\n\n");
423             g_object_set (mprtpsch, "detach-subflow", 2, NULL);
424             data->active_s2=FALSE;
425             num_subflows--;
426         }
427         if(data->active_s3 && num_subflows>1)
428         {
429             g_print("Disabling Subflow 3 ...\n\n");
430             g_object_set (mprtpsch, "detach-subflow", 3, NULL);
431             data->active_s3=FALSE;
432             num_subflows--;
433         }
434         break;
435     }
436     //Attivo Flusso 2 e Disattivo Flusso 1/3
```

```
437     case 1:{
438         if(data->active_s1 && num_subflows>1)
439         {
440             g_print("Disabling Subflow 1 ...\n\n");
441             g_object_set (mprtpsch, "detach-subflow", 1, NULL);
442             data->active_s1=FALSE;
443             num_subflows--;
444         }
445         if(!data->active_s2 && data->join_s2)
446         {
447             g_print("Activate Subflow 2 ...\n\n");
448             g_object_set (mprtpsch, "join-subflow", 2, NULL);
449             data->active_s2=TRUE;
450             num_subflows++;
451         }
452         if(data->active_s3 && num_subflows>1)
453         {
454             g_print("Disabling Subflow 3 ...\n\n");
455             g_object_set (mprtpsch, "detach-subflow", 3, NULL);
456             data->active_s3=FALSE;
457             num_subflows--;
458         }
459         break;
460     }
461     //Attivo Flusso 3 e Disattivo Flusso 1/2
462     case 2:{
463         if(data->active_s1 && num_subflows>1)
464         {
465             g_print("Disabling Subflow 1 ...\n\n");
466             g_object_set (mprtpsch, "detach-subflow", 1, NULL);
467             data->active_s1=FALSE;
468             num_subflows--;
469         }
470         if(data->active_s2 && num_subflows>1)
471         {
472             g_print("Disabling Subflow 2 ...\n\n");
473             g_object_set (mprtpsch, "detach-subflow", 2, NULL);
474             data->active_s2=FALSE;
475             num_subflows--;
476         }
477         if(!data->active_s3 && data->join_s3)
478         {
479             g_print("Activate Subflow 3 ...\n\n");
480             g_object_set (mprtpsch, "join-subflow", 3, NULL);
481             data->active_s3=TRUE;
```

```
482             num_subflows++;
483         }
484         break;
485     }
486     //Attivo Flusso 1/2 Disattivo Flusso 3
487     case 3:{
488         if(!data->active_s1 && data->join_s1)
489         {
490             g_print("Activate Subflow 1 ...\n\n");
491             g_object_set (mprtpsch, "join-subflow", 1, NULL);
492             data->active_s1=TRUE;
493             num_subflows++;
494         }
495         if(!data->active_s2 && data->join_s2)
496         {
497             g_print("Activate Subflow 2 ...\n\n");
498             g_object_set (mprtpsch, "join-subflow", 2, NULL);
499             data->active_s2=TRUE;
500             num_subflows++;
501         }
502         if(data->active_s3 && num_subflows>1)
503         {
504             g_print("Disabling Subflow 3 ...\n\n");
505             g_object_set (mprtpsch, "detach-subflow", 3, NULL);
506             data->active_s3=FALSE;
507             num_subflows--;
508         }
509         break;
510     }
511     //Attivo Flusso 1/3 Disattivo Flusso 2
512     case 4:{
513         if(!data->active_s1 && data->join_s1)
514         {
515             g_print("Activate Subflow 1 ...\n\n");
516             g_object_set (mprtpsch, "join-subflow", 1, NULL);
517             data->active_s1=TRUE;
518             num_subflows++;
519         }
520         if(data->active_s2 && num_subflows>1)
521         {
522             g_print("Disabling Subflow 2 ...\n\n");
523             g_object_set (mprtpsch, "detach-subflow", 2, NULL);
524             data->active_s2=FALSE;
525             num_subflows--;
526         }
```

```
527         if(!data->active_s3 && data->join_s3)
528         {
529             g_print("Activate Subflow 3 ...\n\n");
530             g_object_set (mprtpsch, "join-subflow", 3, NULL);
531             data->active_s3=TRUE;
532             num_subflows++;
533         }
534         break;
535     }
536     //Attivo Flusso 2/3 Disattivo Flusso 1
537     case 5:{
538         if(data->active_s1 && num_subflows>1)
539         {
540             g_print("Disabling Subflow 1 ...\n\n");
541             g_object_set (mprtpsch, "detach-subflow", 1, NULL);
542             data->active_s1=FALSE;
543             num_subflows--;
544         }
545         if(!data->active_s2 && data->join_s2)
546         {
547             g_print("Activate Subflow 2 ...\n\n");
548             g_object_set (mprtpsch, "join-subflow", 2, NULL);
549             data->active_s2=TRUE;
550             num_subflows++;
551         }
552         if(!data->active_s3 && data->join_s3)
553         {
554             g_print("Activate Subflow 3 ...\n\n");
555             g_object_set (mprtpsch, "join-subflow", 3, NULL);
556             data->active_s3=TRUE;
557             num_subflows++;
558         }
559         break;
560     }
561     //Attivo tutti
562     default:{
563         if(!data->active_s1 && data->join_s1)
564         {
565             g_print("Activate Subflow 1 ...\n\n");
566             g_object_set (mprtpsch, "join-subflow", 1, NULL);
567             data->active_s1=TRUE;
568             num_subflows++;
569         }
570         if(!data->active_s2 && data->join_s2)
571         {
```

```
572             g_print("Activate Subflow 2 ...\n\n");
573             g_object_set (mprtpsch, "join-subflow", 2, NULL);
574             data->active_s2=TRUE;
575             num_subflows++;
576         }
577         if(!data->active_s3 && data->join_s3)
578         {
579             g_print("Activate Subflow 3 ...\n\n");
580             g_object_set (mprtpsch, "join-subflow", 3, NULL);
581             data->active_s3=TRUE;
582             num_subflows++;
583         }
584         break;
585     }
586
587
588     }
589     contatore++;
590     //Vado a interrompe dopo aver fatto per 10 volte Join/Detach
591     if(contatore==6){
592         g_print("Interruzione di Join/Detach\n");
593         data->stop = TRUE;
594     }
595     return data->stop ? G_SOURCE_REMOVE : G_SOURCE_CONTINUE;
596 }
597
598
599 static void add_stream (GstPipeline * pipe, GstElement * rtpBin
        , SessionData * session)
600 {
601     gchar *padName;
602     GstElement *rtpSink_1, *rtpSink_2, *rtpSink_3, *
            async_tx_rtcpSink_1, *async_tx_rtcpSink_2, *
            async_tx_rtcpSink_3;
603     GstElement *rtcpSink, *rtcpSrc, *async_rx_rtcpSrc_1, *
            async_rx_rtcpSrc_2, *async_rx_rtcpSrc_3;
604     GstElement *myencoder_1,*myencoder_2,*myencoder_3,*mprtpsnd
            , *mprtprcv, *mprtpsch, *mq;
605     GstElement *kodoSrc1, *kodoSrc2, *kodoSrc3;
606     gboolean res;
607
608     //Creazione pad sink per ricezione frame video su tre diversi canali
609     rtpSink_1 = gst_element_factory_make ("udpsink", "rtpsink_1
            ");
610     g_assert(rtpSink_1);
```

```
611     rtpSink_2 = gst_element_factory_make ("udpsink", "rtpsink_2
            ");
612     g_assert(rtpSink_2);
613     rtpSink_3 = gst_element_factory_make ("udpsink", "rtpsink_3
            ");
614     g_assert(rtpSink_3);
615
616     //Creazione moduli myencoder
617
618     if(test_parameters_.coding_active == NO_CODING){
619         myencoder_1= gst_element_factory_make ("identity", "
                myencoder_1");
620         g_assert(myencoder_1);
621         myencoder_2= gst_element_factory_make ("identity", "
                myencoder_2");
622         g_assert(myencoder_2);
623         myencoder_3= gst_element_factory_make ("identity", "
                myencoder_3");
624         g_assert(myencoder_3);
625
626     }else
627         if(test_parameters_.coding_active == KODO){
628             myencoder_1= gst_element_factory_make ("mykodoenc",
                    "myencoder_1");
629             g_assert(myencoder_1);
630             myencoder_2= gst_element_factory_make ("mykodoenc",
                    "myencoder_2");
631             g_assert(myencoder_2);
632             myencoder_3= gst_element_factory_make ("mykodoenc",
                    "myencoder_3");
633             g_assert(myencoder_3);
634
635             /*Creo gli elementi udp src per la ricezione del
                    PLR */
636
637             //Creazione pad src per la trasmissione asincrona di
                    pacchetti RTCP
638             kodoSrc1 = gst_element_factory_make ("udpsrc", "
                    kodoSrc1");
639             g_assert(kodoSrc1);
640             kodoSrc2 = gst_element_factory_make ("udpsrc", "
                    kodoSrc2");
641             g_assert(kodoSrc2);
642             kodoSrc3 = gst_element_factory_make ("udpsrc", "
                    kodoSrc3");
```

```
643             g_assert(kodoSrc3);
644
645             g_object_set (kodoSrc1, "port",  kodo_rx_port_1,
                    NULL);
646             g_object_set (kodoSrc2, "port",  kodo_rx_port_2,
                    NULL);
647             g_object_set (kodoSrc3, "port",  kodo_rx_port_3,
                    NULL);
648
649         }
650
651     //Creazione pad sink per ricezione asincrona di pacchetti RTCP
652     async_tx_rtcpSink_1 = gst_element_factory_make ("udpsink",
            "async_tx_rtcpsink_1");
653     g_assert(async_tx_rtcpSink_1);
654     async_tx_rtcpSink_2 = gst_element_factory_make ("udpsink",
            "async_tx_rtcpsink_2");
655     g_assert(async_tx_rtcpSink_2);
656     async_tx_rtcpSink_3 = gst_element_factory_make ("udpsink",
            "async_tx_rtcpsink_3");
657     g_assert(async_tx_rtcpSink_3);
658
659     //Creazione pad sink per la ricezione di pacchetti RTCP
660     rtcpSink = gst_element_factory_make ("udpsink", "rtcpSink")
            ;
661     g_assert(rtcpSink);
662
663     //Creazione pad src per la trasmissione di pacchetti RTCP
664     rtcpSrc = gst_element_factory_make ("udpsrc", "rtcpSrc");
665     g_assert(rtcpSrc);
666
667     //Creazione pad src per la trasmissione asincrona di pacchetti RTCP
668     async_rx_rtcpSrc_1 = gst_element_factory_make ("udpsrc", "
            async_rx_rtcpsrc_1");
669     g_assert(async_rx_rtcpSrc_1);
670     async_rx_rtcpSrc_2 = gst_element_factory_make ("udpsrc", "
            async_rx_rtcpsrc_2");
671     g_assert(async_rx_rtcpSrc_2);
672     async_rx_rtcpSrc_3 = gst_element_factory_make ("udpsrc", "
            async_rx_rtcpsrc_3");
673     g_assert(async_rx_rtcpSrc_3);
674
675     //Creazione sender/receiver MPRTP
676     mprtpsnd = gst_element_factory_make ("mprtpsender", "
            senderMPRTP");
```

```
677     g_assert(mprtpsnd);
678     mprtprcv = gst_element_factory_make ("mprtpreceiver", "
            receiverMPRTP");
679     g_assert(mprtprcv);
680
681     //Creazione dello scheduler MPRTP
682     mprtpsch = gst_element_factory_make ("mprtpscheduler", "
            scheduler");
683     g_assert(mprtpsch);
684
685     //Creazione di una coda multipla
686     mq = gst_element_factory_make ("multiqueue", "rtpq");
687     g_assert(mq);
688
689     //Inizializzazione Dati per Attivazione/Disattivazione flussi
690     join_detach_data.stop = FALSE;
691     join_detach_data.active_s1 = FALSE;
692     join_detach_data.active_s2 = FALSE;
693     join_detach_data.active_s3 = FALSE;
694     join_detach_data.join_s1 = FALSE;
695     join_detach_data.join_s2 = FALSE;
696     join_detach_data.join_s3 = FALSE;
697     join_detach_data.mprtpsch = mprtpsch;
698
699     //Aggiungo tutti gli elementi principali alla pipeline
700     gst_bin_add_many (GST_BIN (pipe),rtpSink_1, rtpSink_2,
            rtpSink_3,mprtprcv, mprtpsnd, mprtpsch, rtcpSink,
            rtcpSrc,
701               mq, session->input, myencoder_1,myencoder_2,
                      myencoder_3,NULL);
702
703
704     //Nel caso in cui venga scelto Auto rate e congestion control aggiungo
            alla pipeline anche i pad asincroni per RTCP
705     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
706          gst_bin_add_many (GST_BIN (pipe), async_tx_rtcpSink_1,
                 async_tx_rtcpSink_2, async_tx_rtcpSink_3,
707                    async_rx_rtcpSrc_1, async_rx_rtcpSrc_2,
                           async_rx_rtcpSrc_3, NULL);
708     }
709
710
711
712
```

```
713
714     //Abilito la ritrasmissione settando rtprtcsend come un elemento "aux"
            di rtpbin
715     g_signal_connect (rtpBin, "request-aux-sender",(GCallback)
            request_aux_sender, session);
716
717     //Gestione bitrate
718     g_signal_connect (mprtpsch, "mprtp-subflows-utilization",(
            GCallback) changed_event, NULL);
719
720     //Massimo numero di buffer nella coda
721     g_object_set (mq, "max-size-buffers", 1, NULL);
722
723     //Setto numero di porta e indirizzo di destinazione per i pad RTP per
            trasmissione di frame video
724     g_object_set (rtpSink_1, "port", path1_tx_rtp_port, "host",
            "10.0.0.2", NULL);
725     g_object_set (rtpSink_2, "port", path2_tx_rtp_port, "host",
            "10.0.1.2", NULL);
726     g_object_set (rtpSink_3, "port", path3_tx_rtp_port, "host",
            "10.0.2.2", NULL);
727
728     /*Nel caso in cui sia stato scelto Auto rate e congestion
            control vado a settare anche i pad
729          asincroni per per pacchetti RTCP*/
730     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
731          g_object_set (async_tx_rtcpSink_1, "port",
                path1_tx_rtcp_port, "host", "10.0.0.2", "sync",
                FALSE, "async", FALSE, NULL);
732          g_object_set (async_tx_rtcpSink_2, "port",
                path2_tx_rtcp_port, "host", "10.0.1.2", "sync",
                FALSE, "async", FALSE, NULL);
733          g_object_set (async_tx_rtcpSink_3, "port",
                path3_tx_rtcp_port, "host", "10.0.2.2", "sync",
                FALSE, "async", FALSE, NULL);
734          g_object_set (async_rx_rtcpSrc_1, "port",
                path1_rx_rtcp_port, NULL);
735          g_object_set (async_rx_rtcpSrc_2, "port",
                path2_rx_rtcp_port, NULL);
736          g_object_set (async_rx_rtcpSrc_3, "port",
                path3_rx_rtcp_port, NULL);
737     }
738
739     //Setto numero di porta ed host di destinazione per pad di ricezione
            pacchetti RTCP
```

```
740     g_object_set (rtcpSink, "port", rtpbin_tx_rtcp_port, "host"
            , "10.0.0.2","sync",FALSE, "async", FALSE, NULL);
741
742     //Setto porta ricezione pacchetti RTCP
743     g_object_set (rtcpSrc, "port", rtpbin_rx_rtcp_port, NULL);
744
745     //Collegamento dal pad src del bin contenente la sessione video ed il
            pad sink di rtpbin
746     padName = g_strdup_printf ("send_rtp_sink_%u", session->
            sessionNum);
747     if(padName == NULL)
748         g_error("padName: stringa vuota");
749     res = gst_element_link_pads (session->input, "src", rtpBin,
            padName);
750     if(!res)
751         g_error ("Fallimento nel collegamento tra bin ed rtpbin
                ");
752     g_free (padName);
753
754     padName = g_strdup_printf ("send_rtp_src_%u", session->
            sessionNum);
755     if(padName == NULL)
756         g_error("padName: stringa vuota");
757     //Collegamento tra pad src di rtpbin e pad sink dello scheduler
758     res = gst_element_link_pads (rtpBin, padName, mprtpsch, "
            rtp_sink");
759     if(!res)
760         g_error ("Fallimento nel collegamento tra rtpbin e
                scheduler");
761
762     //Collegamento tra pad src dello scheduler e pad sink del sender MPRTP
763     res = gst_element_link_pads (mprtpsch, "mprtp_src",
            mprtpsnd, "mprtp_sink");
764     if(!res)
765         g_error ("Fallimento nel collegamento tra scheduler e
                sender MPRTP");
766     g_free (padName);
767
768     //Collegamento fra tre pad src del sender MPRTP e tre pad sink della
            coda multipla
769     res = gst_element_link_pads (mprtpsnd, "src_1", mq, "sink_1
            ");
770
771
772     if(!res)
```

```
773         g_error ("Fallimento nel collegamento 1 tra sender
                MPRTP e Multiqueue");
774     res = gst_element_link_pads (mprtpsnd, "src_2", mq, "sink_2
            ");
775     if(!res)
776         g_error ("Fallimento nel collegamento 2 tra sender
                MPRTP e Multiqueue");
777     res = gst_element_link_pads (mprtpsnd, "src_3", mq, "sink_3
            ");
778     if(!res)
779         g_error ("Fallimento nel collegamento 3 tra sender
                MPRTP e Multiqueue");
780
781
782     //Collegamento fra tre pad src della coda multipla e tre pad sink del
            myencoder
783     res = gst_element_link_pads (mq, "src_1", myencoder_1, "
            sink");
784
785     if(!res)
786         g_error ("Fallimento nel collegamento 1 tra Multiqueue
                e coder");
787     res = gst_element_link_pads (mq, "src_2", myencoder_2, "
            sink");
788     if(!res)
789         g_error ("Fallimento nel collegamento 2 tra Multiqueue
                e coder");
790     res = gst_element_link_pads (mq, "src_3", myencoder_3, "
            sink");
791     if(!res)
792         g_error ("Fallimento nel collegamento 3 tra Multiqueue
                e coder");
793
794
795     //Collegamento fra tre pad src del myencoder e tre pad sink si
            trasmissione RTP
796     res = gst_element_link_pads (myencoder_1, "src", rtpSink_1,
             "sink");
797
798     if(!res)
799         g_error ("Fallimento nel collegamento 1 tra myencoder e
                 rtpSink");
800     res = gst_element_link_pads (myencoder_2, "src", rtpSink_2,
             "sink");
801     if(!res)
```

```
802         g_error ("Fallimento nel collegamento 2 tra myencoder e
                rtpSink");
803     res = gst_element_link_pads (myencoder_3, "src", rtpSink_3,
            "sink");
804     if(!res)
805         g_error ("Fallimento nel collegamento 3 tra myencoder e
                rtpSink");
806
807
808     //Permette di unire i sottoflussi che si vogliono usare
809     if(test_parameters_.subflow1_active){
810         g_object_set (mprtpsch, "join-subflow", 1, NULL);
811         join_detach_data.active_s1 = TRUE;
812         join_detach_data.join_s1 = TRUE;
813         num_subflows++;
814     }
815     if(test_parameters_.subflow2_active){
816         g_object_set (mprtpsch, "join-subflow", 2, NULL);
817         join_detach_data.active_s2 = TRUE;
818         join_detach_data.join_s2 = TRUE;
819         num_subflows++;
820     }
821     if(test_parameters_.subflow3_active){
822         g_object_set (mprtpsch, "join-subflow", 3, NULL);
823         join_detach_data.active_s3 = TRUE;
824         join_detach_data.join_s3 = TRUE;
825         num_subflows++;
826     }
827
828     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
829         g_object_set (mprtpsch, "auto-rate-and-cc", TRUE, NULL)
                ;
830     }else if(test_parameters_.test_directive ==
            MANUAL_RATE_CONTROLLING){
831         g_timeout_add (1000, _random_rate_controller, mprtpsch)
                ;
832     }
833
834     if(test_parameters_.random_detach){
835         g_timeout_add (10000, _join_detach_test, &
                join_detach_data);
836     }
837
838     //è attivo un registro per i sottoflussi
```

```
839     g_object_set(mprtpsch, "logging", TRUE, NULL);
840
841
842     padName = g_strdup_printf ("send_rtcp_src_%u", session->
            sessionNum);
843     if(padName == NULL)
844         g_error("padName: stringa vuota");
845     //Collegamento tra pad src di rtpBin e pad sink per invio pacchetti
            RTCP
846     res = gst_element_link_pads (rtpBin, padName, rtcpSink, "
            sink");
847     if(!res)
848         g_error ("Fallimento nel collegamento rtpbin e rtcpSink
                ");
849     g_free (padName);
850
851     padName = g_strdup_printf ("recv_rtcp_sink_%u", session->
            sessionNum);
852     if(padName == NULL)
853         g_error("padName: stringa vuota");
854     /*Nel caso in cui sia stato scelto Auto rate e congestion
            control vado a collegare anche i sink ed src asincroni*/
855     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
856         res = gst_element_link_pads (async_rx_rtcpSrc_1, "src",
                mprtprcv, "async_sink_1");
857         if(!res)
858             g_error ("Fallimento nel collegamento 1 tra
                    async_rtcpSrc e Receiver MPRTP");
859         gst_element_link_pads (async_rx_rtcpSrc_2, "src",
                mprtprcv, "async_sink_2");
860         if(!res)
861             g_error ("Fallimento nel collegamento 2 tra
                    async_rtcpSrc e Receiver MPRTP");
862         gst_element_link_pads (async_rx_rtcpSrc_3, "src",
                mprtprcv, "async_sink_3");
863         if(!res)
864             g_error ("Fallimento nel collegamento 3 tra
                    async_rtcpSrc e Receiver MPRTP");
865
866         res = gst_element_link_pads (mprtpsnd, "async_src_1",
                async_tx_rtcpSink_1, "sink");
867         if(!res)
868             g_error ("Fallimento nel collegamento 1 tra Sender
                    MPRTP e async_rtcpSin");
```

```
869         res = gst_element_link_pads (mprtpsnd, "async_src_2",
                async_tx_rtcpSink_2, "sink");
870         if(!res)
871             g_error ("Fallimento nel collegamento 2 tra Sender
                    MPRTP e async_rtcpSin");
872         res = gst_element_link_pads (mprtpsnd, "async_src_3",
                async_tx_rtcpSink_3, "sink");
873         if(!res)
874             g_error ("Fallimento nel collegamento 3 tra Sender
                    MPRTP e async_rtcpSin");
875     }
876
877     //Collegamento tra pad src del Receiver MPRTP ed il pad sink dello
            scheduler
878     res = gst_element_link_pads (mprtprcv, "mprtcp_rr_src",
            mprtpsch, "mprtcp_rr_sink");
879     if(!res)
880         g_error ("Fallimento nel collegamento tra Receiver
                MPRTP e scheduler");
881
882     //Collegamento tra pad src dello scheduler MPRTP ed il pad sink del
            Sender MPRTP
883     res = gst_element_link_pads (mprtpsch, "mprtcp_sr_src",
            mprtpsnd, "mprtcp_sr_sink");
884     if(!res)
885         g_error ("Fallimento nel collegamento tra scheduler e
                Sender MPRTP");
886
887     //Collegamento tra pad src di ricezione pacchetti RTCP e pad sink di
            rtpbin
888     res = gst_element_link_pads (rtcpSrc, "src", rtpBin,
            padName);
889     if(!res)
890         g_error ("Fallimento nel collegamento tra rtcpSrc e
                rtpBin");
891     g_free (padName);
892
893
894     /* Aggiunge e linka gli udpsrc per il PLR */
895     if(test_parameters_.coding_active == KODO){
896
897         gst_bin_add_many (GST_BIN (pipe), kodoSrc1, kodoSrc2,
                kodoSrc3, NULL);
898
899         //Collegamento fra udpSrc PLR e kodo
900         res = gst_element_link_pads (kodoSrc1, "src",
                myencoder_1, "sink_2");
```

```
901
902         if(!res)
903             g_error ("Fallimento nel collegamento 1 tra udpsrc
                    e kodo");
904         res = gst_element_link_pads (kodoSrc2, "src",
                myencoder_2, "sink_2");
905         if(!res)
906             g_error ("Fallimento nel collegamento 2 tra udpsrc
                    e kodo");
907         res = gst_element_link_pads (kodoSrc3, "src",
                myencoder_3, "sink_2");
908         if(!res)
909             g_error ("Fallimento nel collegamento 3 tra udpsrc
                    e kodo");
910
911     }
912
913
914
915     session_unref (session);
916 }
917
918
919 int main (int argc, char **argv)
920 {
921     GstPipeline *pipe;
922     GstBus *bus;
923     SessionData *videoSession;
924     // SessionData *audioSession;
925     GstElement *rtpBin;
926     GMainLoop *loop;
927     GError *error = NULL;
928     GOptionContext *context;
929
930
931     /*Creazione di un nuovo GOptionContext (struttura che
            definisce quali opzioni sono accettate da linea di
            comando)
          Se mandi in esecuzione ./file --help vengono restituite
              le possibili opzioni da usare  per l'esecuzione*/
932     context = g_option_context_new ("- test tree model
            performance");
933     //carico da opt.h le possibili opzioni
934     g_option_context_add_main_entries (context, entries, NULL);
935     g_option_context_parse (context, &argc, &argv, &error);
936
```

```
937     if(info){
938         _print_info();
939         return 0;
940     }
941     //setto parametri a seconda del profilo scelto
942     _setup_test_params(profile);
943 
944     //Inizializzazione
945     gst_init (NULL, NULL);
946     loop = g_main_loop_new (NULL, FALSE);
947 
948     //Creazione pipeline e bus della pipeline con rispettivo gestore dei
            segnali
949     pipe = GST_PIPELINE (gst_pipeline_new ("senderMPRTP"));
950     bus = gst_element_get_bus (GST_ELEMENT (pipe));
951     g_signal_connect (bus, "message::state-changed", G_CALLBACK
            (cb_state), pipe);
952     gst_bus_add_signal_watch (bus);
953     gst_object_unref (bus);
954 
955     //Creazione di rtpbin
956     rtpBin = gst_element_factory_make ("rtpbin", NULL);
957     //setto profilo Audio/Visual con feedback (RFC 4585)
958     g_object_set (rtpBin, "rtp-profile", GST_RTP_PROFILE_AVPF,
            NULL);
959     //buffer-mode controlla la modalità di buffering e timestamp usato dal
            jitterbuffer
960     gst_bin_add (GST_BIN (pipe), rtpBin);
961 
962     //Apertura file per scrittura dati sottoflussi
963     fd=fopen("dati.txt", "w");
964     if( fd==NULL ) {
965         perror("Errore in apertura del file");
966         exit(1); }
967 
968 
969 
970 
971     //Scelta tra sessione v4l2src per cattura video da webcam o sessione
            filevideo per invio video da file
972     switch(test_parameters_.video_session){
973     case FILE_VIDEO:
974         videoSession = make_video_filevideo_session(
                NUM_SESSIONE);
975         break;
976     case VL2SRC:
```

```
977             videoSession = make_video_vl2src_session(NUM_SESSIONE);
978             break;
979         default:
980             videoSession = make_video_vl2src_session(NUM_SESSIONE);
981             break;
982         }
983 
984         //Creazione elementi trasmissione video MPRTP
985         add_stream (pipe, rtpBin, videoSession);
986 
987         g_print ("starting server pipeline\n");
988         {
989             GstStateChangeReturn changeresult;
990             GstState actual, pending;
991             changeresult = gst_element_set_state (GST_ELEMENT (pipe
                ), GST_STATE_PLAYING);
992             changeresult = gst_element_get_state(GST_ELEMENT (pipe)
                , &actual, &pending, 0);
993             g_print("actual state: %d, pending: %d, changeresult: %
                d\n", actual, pending, changeresult);
994         }
995         g_main_loop_run (loop);
996 
997         g_print ("stopping server pipeline\n");
998         gst_element_set_state (GST_ELEMENT (pipe), GST_STATE_NULL);
999         gst_object_unref (pipe);
1000        g_main_loop_unref (loop);
1001        fclose(fd);
1002        return 0;
1003    }
```

A.6 clientmod.c

```
1  #include <gst/gst.h>
2  #include <gst/rtp/rtp.h>
3  #include <stdlib.h>
4  #include "opt.h"
5  #define HOST "127.0.0.1"
6 
7 
8  GMainLoop *loop = NULL;
9 
10 //Tipo SessionData contenente tutti i parametri che caratterizzano una
       sessione
```

```
11 typedef struct _SessionData
12 {
13     int ref;
14     GstElement *rtpbin;
15     guint sessionNum;
16     GstCaps *caps;
17     GstElement *output;
18 } SessionData;
19
20
21
22
23 //Riferimento ad una sessione
24 static SessionData *session_ref (SessionData * data)
25 {
26     g_atomic_int_inc (&data->ref);
27     return data;
28 }
29
30 //Libera memoria occupata per i parametri di una sessione
31 static void session_unref (gpointer data)
32 {
33     SessionData *session = (SessionData *) data;
34     if (g_atomic_int_dec_and_test (&session->ref)) {
35         g_object_unref (session->rtpbin);
36         gst_caps_unref (session->caps);
37         g_free (session);
38     }
39 }
40
41 //Creazione di una nuova sessione
42 static SessionData *session_new (guint sessionNum)
43 {
44     SessionData *ret = g_new0 (SessionData, 1);
45     ret->sessionNum = sessionNum;
46     return session_ref (ret);
47 }
48
49 //Creazione di un ghost pad sink
50 static void setup_ghost_sink (GstElement * sink, GstBin * bin)
51 {
52     GstPad *sinkPad = gst_element_get_static_pad (sink, "sink")
        ;
53     if(sinkPad == NULL)
54         g_error("Pad sink non esistente");
```

```
55      GstPad *binPad = gst_ghost_pad_new ("sink", sinkPad);
56      gst_element_add_pad (GST_ELEMENT (bin), binPad);
57  }
58
59
60  //Creazione di una sessione video con memorizzazione su file
61  static SessionData *make_filevideo_session (guint sessionNum)
62  {
63      gboolean res;
64      //Framerate
65      gint framerate = 25;
66      GstBin *bin;
67      GstElement *queue,*depayloader,*decoder,*converter,*sink;
68      //Creazione di una nuova sessione
69      SessionData *ret = session_new (sessionNum);
70
71      //Creazione del bin che conterrà tutti gli elementi di questa sessione
72      bin = GST_BIN (gst_bin_new ("filevideo"));
73
74      //Creazione della coda in ingresso
75      queue = gst_element_factory_make ("queue", NULL);
76      g_assert(queue);
77
78      //Creazione elemento per spacchettamento
79      depayloader = gst_element_factory_make ("rtpvp8depay", NULL
            );
80      g_assert(depayloader);
81
82      //Decoder
83      decoder = gst_element_factory_make ("vp8dec", NULL);
84      g_assert(decoder);
85
86      //...  rtph264depay !  h264parse !  mp4mux !  filesink location=my.mp4
87
88      //Salvo video su file
89      sink = gst_element_factory_make("filesink","sinkvideo");
90      g_assert(sink);
91
92      //Indico locazione dove verrà memorizzato il video ricevuto
93      g_object_set (G_OBJECT (sink), "location", "test_rcv.yuv",
            NULL);
94
95      //Aggiungo elementi al bin e li collego tra di loro
96      gst_bin_add_many (bin, depayloader, decoder, queue, sink,
            NULL);
```

```
97      res = gst_element_link_many (queue, depayloader, decoder,
            sink, NULL);
98      if(!res)
99          g_error ("Fallimento nel collegamento tra gli elementi
                del bin");
100     //Creazione di un pad sink fantasma per il bin creato usando il pad
            sink della coda in ingresso
101     setup_ghost_sink (queue, bin);
102
103
104
105     //Metto il bin come output della sessione
106     ret->output = GST_ELEMENT (bin);
107
108     //setto capabilities sessione
109     ret->caps = gst_caps_new_simple ("application/x-rtp",
110             "media", G_TYPE_STRING, "video",
111             "clock-rate", G_TYPE_INT, 90000,
112             "width", G_TYPE_INT, 352,
113             "height", G_TYPE_INT, 264,
114             "framerate", GST_TYPE_FRACTION, framerate, 1,
115             "encoding-name", G_TYPE_STRING, "VP8", NULL
116     );
117     g_object_set (sink, "sync", FALSE, NULL);
118     return ret;
119 }
120
121
122
123 //Creazione di una sessione video
124 static SessionData *make_video_session (guint sessionNum)
125 {
126     gboolean res;
127     //Framerate
128     gint framerate = 25;
129     GstBin *bin;
130     GstElement  *queue,*depayloader,*decoder,*converter,*sink;
131     //Creazione di una nuova sessione
132     SessionData *ret = session_new (sessionNum);
133
134     //Creazione del bin che conterrà tutti gli elementi di questa sessione
135     bin = GST_BIN (gst_bin_new ("video"));
136
137
138
```

```
139     //Creazione della coda in ingresso
140     queue = gst_element_factory_make ("queue", NULL);
141     g_assert(queue);
142
143     //Creazione elemento per spacchettamento
144     depayloader = gst_element_factory_make ("rtpvp8depay", NULL
            );
145     g_assert(depayloader);
146
147     //Decoder
148     decoder = gst_element_factory_make ("vp8dec", NULL);
149     g_assert(decoder);
150
151     //Convertitore video
152     converter = gst_element_factory_make ("videoconvert", NULL)
            ;
153     g_assert(converter);
154
155     //Visualizzo video sullo schermo
156     sink = gst_element_factory_make ("autovideosink", NULL);
157     g_assert(sink);
158
159     //Aggiungo elementi al bin e li collego tra di loro
160     gst_bin_add_many (bin, depayloader, decoder, converter,
            queue, sink, NULL);
161     res = gst_element_link_many ( queue, depayloader, decoder,
            converter, sink, NULL);
162     if(!res)
163         g_error ("Fallimento nel collegamento tra gli elementi
                del bin");
164     //Creazione di un pad sink fantasma per il bin creato usando il pad
            sink della coda in ingresso
165     setup_ghost_sink (queue, bin);
166
167     //Metto il bin come output della sessione
168     ret->output = GST_ELEMENT (bin);
169
170
171     g_object_set(queue,"max-size-buffers",200000,NULL);
172     /*  seting up the queue threshold
173     g_object_set(queue,"min-threshold-buffers",20,NULL);
174
175     g_print("\nclient queue min-thresholdbuffers = 5");*/
176
177
```

```
178     /*if(test_parameters_.video_session == TEST_SOURCE)
179         framerate = 50;*/
180
181     //setto capabilities sessione
182     ret->caps = gst_caps_new_simple ("application/x-rtp",
183             "media", G_TYPE_STRING, "video",
184             "clock-rate", G_TYPE_INT, 90000,
185             "width", G_TYPE_INT, 640,
186             "height", G_TYPE_INT, 360,
187             "framerate", GST_TYPE_FRACTION, framerate, 1,
188             "encoding-name", G_TYPE_STRING, "VP8", NULL
189     );
190
191     /*TRUE for kodo, original = FALSE */
192     g_object_set (sink, "sync", TRUE, NULL);
193     return ret;
194 }
195
196 //Richiesta mappa payload type
197 static GstCaps *request_pt_map (GstElement * rtpbin, guint
       session, guint pt, gpointer user_data)
198 {
199     SessionData *data = (SessionData *) user_data;
200     g_print ("Looking for caps for pt %u in session %u, have %u
           \n", pt, session, data->sessionNum);
201     if (session == data->sessionNum) {
202         g_print ("Returning %s\n", gst_caps_to_string (data->
               caps));
203         return gst_caps_ref (data->caps);
204     }
205     return NULL;
206 }
207
208 //Gestione messaggi di End Of Stream del bus
209 static void cb_eos (GstBus * bus, GstMessage * message,
       gpointer data)
210 {
211     g_print ("Got EOS\n");
212     g_main_loop_quit (loop);
213 }
214
215 //Gestione messaggi di "cambiamento stato" del bus
216 static void cb_state (GstBus * bus, GstMessage * message,
       gpointer data)
217 {
```

```
218     GstObject *pipe = GST_OBJECT (data);
219     GstState old, new, pending;
220     gst_message_parse_state_changed (message, &old, &new, &
           pending);
221     if (message->src == pipe) {
222         g_print ("Pipeline %s changed state from %s to %s\n",
223                 GST_OBJECT_NAME (message->src),
224                 gst_element_state_get_name (old),
                       gst_element_state_get_name (new));
225     }
226 }
227
228 //Gestione messaggi di warning del bus
229 static void cb_warning (GstBus * bus, GstMessage * message,
       gpointer data)
230 {
231     GError *error = NULL;
232     gst_message_parse_warning (message, &error, NULL);
233     g_printerr ("Got warning from %s: %s\n", GST_OBJECT_NAME (
           message->src),
234             error->message);
235     g_error_free (error);
236 }
237
238 //Gestione messaggi di errore del bus
239 static void cb_error (GstBus * bus, GstMessage * message,
       gpointer data)
240 {
241     GError *error = NULL;
242     gst_message_parse_error (message, &error, NULL);
243     g_printerr ("Got error from %s: %s\n", GST_OBJECT_NAME (
           message->src),
244             error->message);
245     g_error_free (error);
246     g_main_loop_quit (loop);
247 }
248
249 //Gestione nuovo stream
250 static void handle_new_stream (GstElement * element, GstPad *
       newPad, gpointer data)
251 {
252     SessionData *session = (SessionData *) data;
253     gchar *padName;
254     gchar *myPrefix;
255
```

```
256     //Creazione di un nuovo pad
257     padName = gst_pad_get_name (newPad);
258     myPrefix = g_strdup_printf ("recv_rtp_src_%u", session->
            sessionNum);
259     if(myPrefix == NULL)
260         g_error("myPrefix: stringa vuota");
261 
262     g_print ("New pad: %s, looking for %s_*\n", padName,
            myPrefix);
263 
264     if (g_str_has_prefix (padName, myPrefix)) {
265         GstPad *outputSinkPad;
266         GstElement *parent;
267 
268         //Creazione nuovo elemento bin che eredita dati della sessione
                passata come argomento
269         parent = GST_ELEMENT (gst_element_get_parent (session->
                rtpbin));
270         gst_bin_add (GST_BIN (parent), session->output);
271         gst_element_sync_state_with_parent (session->output);
272         gst_object_unref (parent);
273 
274         //creazione pad fantasma di tipo sink
275         outputSinkPad = gst_element_get_static_pad (session->
                output, "sink");
276         if(outputSinkPad == NULL)
277             g_error("Pad sink non esistente");
278         g_assert_cmpint (gst_pad_link (newPad, outputSinkPad),
                ==, GST_PAD_LINK_OK);
279         gst_object_unref (outputSinkPad);
280         g_print ("Linked!\n");
281     }
282     g_free (myPrefix);
283     g_free (padName);
284 }
285 
286 
287 //Creazione di rtprtxreceive come elemento "aux" di rtpbin
288 static GstElement *request_aux_receiver (GstElement * rtpbin,
        guint sessid, SessionData * session)
289 {
290     GstElement *rtx, *bin;
291     GstPad *pad;
292     gchar *name;
293     GstStructure *pt_map;
```

```
294
295     GST_INFO ("creating AUX receiver");
296     bin = gst_bin_new (NULL);
297
298     //Creazione elemento rtprtxreceive
299     rtx = gst_element_factory_make ("rtprtxreceive", "aux");
300     g_assert(rtx);
301
302     //Mappa dei tipi di payload originale per i loro payload di
           ritrasmissione
303     pt_map = gst_structure_new ("application/x-rtp-pt-map", "96
           ", G_TYPE_UINT, 99, NULL);
304     g_object_set (rtx, "payload-type-map", pt_map, NULL);
305     gst_structure_free (pt_map);
306
307     //Aggiungo elemento al bin creato
308     gst_bin_add (GST_BIN (bin), rtx);
309
310     //Creazione ghost pad di tipo src per bin creato
311     pad = gst_element_get_static_pad (rtx, "src");
312     if(pad == NULL)
313         g_error("Pad src non esistente");
314     name = g_strdup_printf ("src_%u", sessid);
315     gst_element_add_pad (bin, gst_ghost_pad_new (name, pad));
316     g_free (name);
317     gst_object_unref (pad);
318
319     //Creazione ghost pad di tipo sink per bin creato
320     pad = gst_element_get_static_pad (rtx, "sink");
321     if(pad == NULL)
322         g_error("Pad sink non esistente");
323     name = g_strdup_printf ("sink_%u", sessid);
324     gst_element_add_pad (bin, gst_ghost_pad_new (name, pad));
325     g_free (name);
326     gst_object_unref (pad);
327
328     return bin;
329 }
330
331
332 static void join_session (GstElement * pipeline, GstElement *
       rtpBin, SessionData * session, guint32 clockrate)
333 {
334     GstElement *rtpSrc_1, *rtpSrc_2, *rtpSrc_3, *mydecoder_1, *
           mydecoder_2, *mydecoder_3;
```

```
335     GstElement *async_tx_rtcpSrc_1, *async_tx_rtcpSrc_2, *
            async_tx_rtcpSrc_3;
336     GstElement *rtcpSrc;
337     GstElement *rtcpSink, *async_rx_rtcpSink_1, *
            async_rx_rtcpSink_2, *async_rx_rtcpSink_3;
338     GstElement *mprtprcv, *mprtpsnd, *mprtpply;
339     GstElement *kodoSink1, *kodoSink2, *kodoSink3;
340     gchar *padName, *padname2;
341     guint basePort;
342     gboolean res;
343
344     g_print ("Joining session %p\n", session);
345
346     session->rtpbin = g_object_ref (rtpBin);
347
348     basePort = 5000 + (session->sessionNum * 20);
349
350     //Creazione pad di ricezione dati RTP
351     rtpSrc_1 = gst_element_factory_make ("udpsrc", "rtpSrc1");
352     g_assert(rtpSrc_1);
353     rtpSrc_2 = gst_element_factory_make ("udpsrc", "rtpSrc2");
354     g_assert(rtpSrc_2);
355     rtpSrc_3 = gst_element_factory_make ("udpsrc", "rtpSrc3");
356     g_assert(rtpSrc_3);
357
358
359     //Creazione codici di protezione
360
361     if(test_parameters_.coding_active == KODO){
362         mydecoder_1 = gst_element_factory_make ("mykododec", "
                mydecoder_1");
363         g_assert(mydecoder_1);
364         mydecoder_2 = gst_element_factory_make ("mykododec", "
                mydecoder_2");
365         g_assert(mydecoder_2);
366         mydecoder_3 = gst_element_factory_make ("mykododec", "
                mydecoder_3");
367         g_assert(mydecoder_3);
368
369         /*Creo gli udp sink per l'invio del PLR */
370
371         kodoSink1 = gst_element_factory_make ("udpsink", "
                kodoSink1");
372         g_assert(kodoSink2);
```

```
373         kodoSink2 = gst_element_factory_make ("udpsink", "
                kodoSink2");
374         g_assert(kodoSink2);
375         kodoSink3 = gst_element_factory_make ("udpsink", "
                kodoSink3");
376         g_assert(kodoSink3);
377
378         g_object_set (kodoSink1, "port", kodo_rx_port_1, "host"
                , "10.0.3.1", "sync", FALSE, "async", FALSE, NULL);
379         g_object_set (kodoSink2, "port", kodo_rx_port_2, "host"
                , "10.0.4.1", "sync", FALSE, "async", FALSE, NULL);
380         g_object_set (kodoSink3, "port", kodo_rx_port_3, "host"
                , "10.0.5.1", "sync", FALSE, "async", FALSE, NULL);
381
382     }else if(test_parameters_.coding_active == NO_CODING){
383         mydecoder_1 = gst_element_factory_make ("identity", "
                mydecoder_1");
384         g_assert(mydecoder_1);
385         mydecoder_2 = gst_element_factory_make ("identity", "
                mydecoder_2");
386         g_assert(mydecoder_2);
387         mydecoder_3 = gst_element_factory_make ("identity", "
                mydecoder_3");
388         g_assert(mydecoder_3);
389     }
390
391     //Creazione pad src per la ricezione asincrona dei pacchetti src
392     async_tx_rtcpSrc_1 = gst_element_factory_make ("udpsrc", "
            asyncrtcpSrc1");
393     g_assert(async_tx_rtcpSrc_1);
394     async_tx_rtcpSrc_2 = gst_element_factory_make ("udpsrc", "
            asyncrtcpSrc2");
395     g_assert(async_tx_rtcpSrc_2);
396     async_tx_rtcpSrc_3 = gst_element_factory_make ("udpsrc", "
            asyncrtcpSrc3");
397     g_assert(async_tx_rtcpSrc_3);
398
399     //Creazione pad src per la ricezione di pacchetti RTCP
400     rtcpSrc = gst_element_factory_make ("udpsrc", "rtcpSrc");
401     g_assert(rtcpSrc);
402
403     //Creazione pad sink per invio di pacchetti RTCP
404     rtcpSink = gst_element_factory_make ("udpsink", "rtcpSink")
            ;
405     g_assert(rtcpSink);
```

```
406
407     //Creazione pad sink per la ricezione asincrona di pacchetti RTCP
408     async_rx_rtcpSink_1 = gst_element_factory_make ("udpsink",
           "asyncrtcpSink1");
409     g_assert(async_rx_rtcpSink_1);
410     async_rx_rtcpSink_2 = gst_element_factory_make ("udpsink",
           "asyncrtcpSink2");
411     g_assert(async_rx_rtcpSink_2);
412     async_rx_rtcpSink_3 = gst_element_factory_make ("udpsink",
           "asyncrtcpSink3");
413     g_assert(async_rx_rtcpSink_3);
414
415     //Creazione Receiver/Sender MPRTP
416     mprtprcv = gst_element_factory_make ("mprtpreceiver", NULL)
           ;
417     g_assert(mprtprcv);
418     mprtpsnd = gst_element_factory_make ("mprtpsender", NULL);
419     g_assert(mprtpsnd);
420
421     //Creazione per emissione video
422     mprtpply = gst_element_factory_make ("mprtpplayouter", NULL
           );
423     g_assert(mprtpply);
424
425     if(test_parameters_.video_session == FILE_VIDEO)
426         g_object_set (mprtpply, "halt-time", 10000, NULL);
427
428     //Setto numeri di porta per la ricezione dei pacchetti RTP
429     g_object_set (rtpSrc_1, "port", path1_tx_rtp_port, "caps",
           session->caps, NULL);
430     g_object_set (rtpSrc_2, "port", path2_tx_rtp_port, "caps",
           session->caps, NULL);
431     g_object_set (rtpSrc_3, "port", path3_tx_rtp_port, "caps",
           session->caps, NULL);
432
433     if(test_parameters_.test_directive ==
           AUTO_RATE_AND_CC_CONTROLLING){
434         g_object_set (async_rx_rtcpSink_1, "port",
               path1_rx_rtcp_port, "host", "10.0.0.1", "sync",
               FALSE, "async", FALSE, NULL);
435         g_object_set (async_rx_rtcpSink_2, "port",
               path2_rx_rtcp_port, "host", "10.0.1.1", "sync",
               FALSE, "async", FALSE, NULL);
436         g_object_set (async_rx_rtcpSink_3, "port",
               path3_rx_rtcp_port, "host", "10.0.2.1", "sync",
```

```
            FALSE, "async", FALSE, NULL);
437         g_object_set (async_tx_rtcpSrc_1, "port",
                path1_tx_rtcp_port, NULL);
438         g_object_set (async_tx_rtcpSrc_2, "port",
                path2_tx_rtcp_port, NULL);
439         g_object_set (async_tx_rtcpSrc_3, "port",
                path3_tx_rtcp_port, NULL);
440     }
441
442
443     g_object_set (rtcpSrc, "port", rtpbin_tx_rtcp_port, NULL);
444     g_object_set (rtcpSink, "port", rtpbin_rx_rtcp_port, "host"
            , "10.0.0.1","async", FALSE, NULL);
445
446
447
448     /*g_object_set (rtpSrc_2,"buffer-size",5,NULL);
449     g_object_set (rtpSrc_2,"buffer-size",5,NULL);
450     g_object_set (rtpSrc_2,"buffer-size",5,NULL);
451         g_print("\nRtpSrc buffer =  20");
452      */
453
454     //Imposta la frequenza di clock del flusso di pivot utilizzato per il
            calcolo dello skew e del ritardo di playout
455     //al ricevitore
456     g_object_set (mprtpply, "pivot-clock-rate", clockrate, NULL
            );
457
458     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING)
459         g_object_set (mprtpply, "auto-rate-and-cc", TRUE, NULL)
                ;
460
461     //Aggiungo sottoflussi che si è scelto di usare
462     if(test_parameters_.subflow1_active)
463         g_object_set (mprtpply, "join-subflow", 1, NULL);
464     if(test_parameters_.subflow2_active)
465         g_object_set (mprtpply, "join-subflow", 2, NULL);
466     if(test_parameters_.subflow3_active)
467         g_object_set (mprtpply, "join-subflow", 3, NULL);
468
469
470     g_object_set(mprtpply, "logging", TRUE, NULL);
471
472     if(0 && test_parameters_.video_session == FILE_VIDEO)
```

```
473         g_object_set(mprtpply, "delay-offset", 2 * GST_SECOND,
                NULL);
474
475
476
477
478     g_print ("Connecting to %i/%i/%i/%i/%i/%i\n", basePort,
            basePort + 1, basePort + 5, basePort + 11,
479              basePort + 12, basePort + 10);
480
481
482     //Permetto ritrasmissione creando un rtprtxreceive come elemento "aux"
            di rtpbin
483     g_signal_connect (rtpBin, "request-aux-receiver", (
            GCallback) request_aux_receiver, session);
484
485     //Aggiungo gli elementi al bin
486     gst_bin_add_many (GST_BIN (pipeline), rtpSrc_1, rtpSrc_2,
            rtpSrc_3, rtcpSrc, rtcpSink,
487              mydecoder_1,mydecoder_2,mydecoder_3,mprtpsnd,
                    mprtprcv, mprtpply, NULL);
488
489
490     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
491          gst_bin_add_many (GST_BIN (pipeline),
                async_rx_rtcpSink_1, async_rx_rtcpSink_2,
                async_rx_rtcpSink_3,
492                   async_tx_rtcpSrc_1,async_tx_rtcpSrc_2,
                        async_tx_rtcpSrc_3, NULL);
493     }
494
495
496
497
498     //Gestione nuovo stream
499     g_signal_connect_data (rtpBin, "pad-added", G_CALLBACK (
            handle_new_stream), session_ref (session),
500              (GClosureNotify) session_unref, 0);
501
502     //Richiesta mappa payload type
503     g_signal_connect_data (rtpBin, "request-pt-map", G_CALLBACK
             (request_pt_map), session_ref (session),
504              (GClosureNotify) session_unref, 0);
505
```

```
506     //Collegamento tra pad di ricezione RTP e kodo decoder
507     res = gst_element_link_pads (rtpSrc_1, "src", mydecoder_1,
           "sink");
508     if(!res)
509         g_error("Fallimento nel collegamento 1 tra rtpsrc e
               decoder ");
510     res = gst_element_link_pads (rtpSrc_2, "src", mydecoder_2,
           "sink");
511     if(!res)
512         g_error("Fallimento nel collegamento 2 tra rtpsrc e
               decoder receiver ");
513     res = gst_element_link_pads (rtpSrc_3, "src", mydecoder_3,
           "sink");
514     if(!res)
515         g_error("Fallimento nel collegamento 3 tra rtpsrc e
               decoder receiver ");
516
517     //Collegamento tra pad di ricezione kodo decoder e MPRTP Receiver
518     res = gst_element_link_pads (mydecoder_1, "src", mprtprcv,
           "sink_1");
519     if(!res)
520         g_error("Fallimento nel collegamento 1 tra mydecoder e
               MPRTP receiver ");
521     res = gst_element_link_pads (mydecoder_2, "src", mprtprcv,
           "sink_2");
522     if(!res)
523         g_error("Fallimento nel collegamento 2 tra mydecoder e
               MPRTP receiver ");
524     res = gst_element_link_pads (mydecoder_3, "src", mprtprcv,
           "sink_3");
525     if(!res)
526         g_error("Fallimento nel collegamento 3 tra mydecoder e
               MPRTP receiver ");
527
528
529     //Collegamento tra MPRTP Receiver e MPRTP Playouter
530     res = gst_element_link_pads (mprtprcv, "mprtp_src",
           mprtpply, "mprtp_sink");
531     if(!res)
532         g_error("Fallimento nel collegamento tra MPRTP Receiver
               e MPRTP Playouter ");
533
534     //Collegamento tra Playouter e RTPBIN
535     padName = g_strdup_printf ("recv_rtp_sink_%u", session->
           sessionNum);
```

```
536     if(padName == NULL)
537         g_error("padName: stringa vuota");
538     res = gst_element_link_pads (mprtpply, "mprtp_src", rtpBin,
            padName);
539     if(!res)
540         g_error("Fallimento nel collegamento tra MPRTP
              Playouter ed RTPBIN ");
541     g_free (padName);
542
543     //Collegamento tra MPRTP Receiver e MPRTP Playouter (pacchetti Sender
        Report)
544     res = gst_element_link_pads (mprtprcv, "mprtcp_sr_src",
        mprtpply, "mprtcp_sr_sink");
545     if(!res)
546         g_error("Fallimento nel collegamento tra MPRTP Receiver
              e MPRTP Playouter (Sender Report) ");
547
548     //Collegamento tra pad ingresso RTCP e RTPBIN
549     padName = g_strdup_printf ("recv_rtcp_sink_%u", session->
        sessionNum);
550     if(padName == NULL)
551         g_error("padName: stringa vuota");
552     res = gst_element_link_pads (rtcpSrc, "src", rtpBin,
        padName);
553     if(!res)
554         g_error("Fallimento nel collegamento tra pad src RTCP e
              RTPBIN ");
555     g_free (padName);
556
557     //Collegamento tra MPRTP Playouter e MPRTP Sender (pacchetti Receiver
        Report)
558     res = gst_element_link_pads (mprtpply, "mprtcp_rr_src",
        mprtpsnd, "mprtcp_rr_sink");
559     if(!res)
560         g_error("Fallimento nel collegamento tra  MPRTP
              Playouter ed MPRTP Sender (Receiver Report) ");
561
562     //Collegamento tra RTPBIN e pad sink per invio pacchetti RTCP
563     padName = g_strdup_printf ("send_rtcp_src_%u", session->
        sessionNum);
564     if(padName == NULL)
565         g_error("padName: stringa vuota");
566     res = gst_element_link_pads (rtpBin, padName, rtcpSink, "
        sink");
567     if(!res)
```

```
568         g_error("Fallimento nel collegamento tra  RTPBIN e pad
                sink RTCP ");
569
570     if(test_parameters_.test_directive ==
            AUTO_RATE_AND_CC_CONTROLLING){
571         gst_element_link_pads (async_tx_rtcpSrc_1, "src",
                mprtprcv, "async_sink_1");
572         gst_element_link_pads (async_tx_rtcpSrc_2, "src",
                mprtprcv, "async_sink_2");
573         gst_element_link_pads (async_tx_rtcpSrc_3, "src",
                mprtprcv, "async_sink_3");
574         gst_element_link_pads (mprtpsnd, "async_src_1",
                async_rx_rtcpSink_1, "sink");
575         gst_element_link_pads (mprtpsnd, "async_src_2",
                async_rx_rtcpSink_2, "sink");
576         gst_element_link_pads (mprtpsnd, "async_src_3",
                async_rx_rtcpSink_3, "sink");
577     }
578
579     /*Aggiunge e linka gli udpsink per il PLR */
580     if(test_parameters_.coding_active == KODO){
581         gst_bin_add_many (GST_BIN (pipeline),kodoSink1,
                kodoSink2, kodoSink3, NULL);
582
583
584         //Collegamento tra kododec e udpsink
585         res = gst_element_link_pads (mydecoder_1, "src_2",
                kodoSink1, "sink");
586         if(!res)
587             g_error("Fallimento nel collegamento 1 tra kododec
                    e udpsink ");
588         res = gst_element_link_pads (mydecoder_2, "src_2",
                kodoSink2, "sink");
589         if(!res)
590             g_error("Fallimento nel collegamento 2 tra kododec
                    e udpsink ");
591         res = gst_element_link_pads (mydecoder_3, "src_2",
                kodoSink3, "sink");
592         if(!res)
593             g_error("Fallimento nel collegamento 3 tra kododec
                    e udpsink ");
594
595     }
596
597
```

```
598     g_free (padName);
599
600     session_unref (session);
601 }
602
603
604
605 int main (int argc, char **argv)
606 {
607     GstPipeline *pipe;
608     SessionData *videoSession;
609     SessionData *audioSession;
610     GstElement *rtpBin;
611     GstBus *bus;
612     GError *error = NULL;
613     GOptionContext *context;
614
615     /*Creazione di un nuovo GOptionContext (struttura che
          definisce quali opzioni sono accettate da linea di
          comando)
616      Se mandi in esecuzione ./file --help vengono restituite le
           possibili opzioni da usare  per l'esecuzione*/
617     context = g_option_context_new ("- test tree model
          performance");
618     g_option_context_add_main_entries (context, entries, NULL);
619     g_option_context_parse (context, &argc, &argv, &error);
620     if(info){
621         _print_info();
622         return 0;
623     }
624     _setup_test_params(profile);
625
626     //Inizializzazione
627     gst_init (NULL, NULL);
628     loop = g_main_loop_new (NULL, FALSE);
629
630     //Creazione della pipeline e del bus
631     pipe = GST_PIPELINE (gst_pipeline_new ("receiverMPRTP"));
632     bus = gst_element_get_bus (GST_ELEMENT (pipe));
633
634     //Gestione messaggi bus
635     g_signal_connect (bus, "message::error", G_CALLBACK (
          cb_error), pipe);
636     g_signal_connect (bus, "message::warning", G_CALLBACK (
          cb_warning), pipe);
```

```
637     g_signal_connect (bus, "message::state-changed", G_CALLBACK
            (cb_state), pipe);
638     g_signal_connect (bus, "message::eos", G_CALLBACK (cb_eos),
            NULL);
639     gst_bus_add_signal_watch (bus);
640     gst_object_unref (bus);
641
642     //Creazione di rtpbin e aggiunta di questo alla pipeline
643     rtpBin = gst_element_factory_make ("rtpbin", NULL);
644     gst_bin_add (GST_BIN (pipe), rtpBin);
645     //setto parametri rtpbin (Buoni parametri:  latency = 2000,
            buffer-mode = 1)
646     g_object_set (rtpBin, "latency", 1500, "do-retransmission",
            TRUE,"buffer-mode",1,"rtp-profile",
            GST_RTP_PROFILE_AVPF, NULL);
647
648     //Output Video
649     if(test_parameters_.record_to_file && test_parameters_.
            video_session == FILE_VIDEO)
650         videoSession = make_filevideo_session (NUM_SESSIONE);
651     else
652         //Creazione della nuova sessione video
653         videoSession = make_video_session (NUM_SESSIONE);
654
655     //Unione sessione
656     join_session (GST_ELEMENT (pipe), rtpBin, videoSession,
            90000);
657
658     g_print ("starting client pipeline\n");
659     gst_element_set_state (GST_ELEMENT (pipe),
            GST_STATE_PLAYING);
660
661     g_main_loop_run (loop);
662
663     g_print ("stoping client pipeline\n");
664     gst_element_set_state (GST_ELEMENT (pipe), GST_STATE_NULL);
665
666     gst_object_unref (pipe);
667     g_main_loop_unref (loop);
668
669     return 0;
670 }
```

A.7 opt.h

```
#ifndef OPTIONS_H_
#define OPTIONS_H_

#include <string.h>

#define println(str) g_print(str"\n")
#define NUM_SESSIONE 0

static const int path1_tx_rtp_port  = 5000;
static const int path1_tx_rtcp_port = 5001;
static const int path1_rx_rtp_port  = 5002;
static const int path1_rx_rtcp_port = 5003;

static const int path2_tx_rtp_port  = 5004;
static const int path2_tx_rtcp_port = 5005;
static const int path2_rx_rtp_port  = 5006;
static const int path2_rx_rtcp_port = 5007;

static const int path3_tx_rtp_port  = 5008;
static const int path3_tx_rtcp_port = 5009;
static const int path3_rx_rtp_port  = 5010;
static const int path3_rx_rtcp_port = 5011;

static const int rtpbin_tx_rtcp_port = 5013;
static const int rtpbin_rx_rtcp_port = 5015;

static const int kodo_rx_port_1 = 5016;
static const int kodo_rx_port_2 = 5017;
static const int kodo_rx_port_3 = 5018;

typedef enum{
  NO_CONTROLLING               = 0,
  MANUAL_RATE_CONTROLLING      = 1,
  AUTO_RATE_AND_CC_CONTROLLING = 2,
}TestSuite;

typedef enum{
  NO_CODING           = 0,
  KODO            = 1,
}TestSuite2;

typedef enum{
  FILE_VIDEO     = 0,
```

```
44    VL2SRC          = 1,
45  }VideoSession;
46
47  typedef struct _TestParams{
48    TestSuite    test_directive;
49    VideoSession video_session;
50    gboolean     random_detach;
51    gboolean     subflow1_active;
52    gboolean     subflow2_active;
53    gboolean     subflow3_active;
54    gboolean     record_to_file;
55    TestSuite2   coding_active;
56    guint        subflow_num;
57  }TestParams;
58
59  static TestParams test_parameters_;
60  static gint32 profile;
61  static gint32 info;
62
63  static GOptionEntry entries[] =
64  {
65      { "profile", 0, 0, G_OPTION_ARG_INT, &profile, "Profile",
          NULL },
66      { "info", 0, 0, G_OPTION_ARG_NONE, &info, "Info", NULL },
67    { NULL }
68  };
69
70
71  static void _print_info(void)
72  {
73    println("############################## Profiles
          ########################################");
74    println("#

          #");
75    println("# profile = 0|0|0|00|0|0|0|0
                                                          #");
76    println("#           ^ ^ ^ ^^ ^ ^ ^ ^
                                                          #");
77    println("#           | | | || | | | |0/1 - Deactivate/
          Activate subflow 1 ----> +1     #");
78    println("#           | | | || | | |0/1 - Deactivate/Activate
          subflow 2   ----> +2     #");
79    println("#           | | | || | |0/1 - Deactivate/Activate
          subflow 3     ----> +4     #");
```

```
80    println("#           | | | || |0 - File Video ,1 - v4l2src
                    ----> +8     #");
81    println("#           | | | ||00 - No rate control, 1 - random
          rate ctrl  ----> +16    #");
82    println("#           | | | ||10 - Auto rate and cc control
                    ----> +32    #");
83    println("#           | | |0/1 - Join detach subflows
         continously          ----> +64    #");
84    println("#           | |0 - Record to file, 1 - display to
         video          ----> +128   #");
85    println("#           |1 - enable RLNC
                                        ----> +256   #");
86    println("#

         #");
87    println("# Examples:

         #");
88    println("# --profile=1 <- subflow 1, test source, no rate
         controller                   #");
89    println("# --profile=3 <- subflow 1 and 2, test source, no
         rate controller            #");
90    println("# --profile=67 <- sub 1 and 2, test source, rate and
          cc                       #");
91    println("
         ##############################################################################
         ");
92  }
93
94
95  static void _setup_test_params(guint profile)
96  {
97    memset(&test_parameters_, 0, sizeof(TestParams));
98    if(profile == 0){
99        profile = 1;
100   }
101   g_print("Selected test profile: %d, it setups the following:\
         n", profile);
102
103   test_parameters_.subflow1_active = (profile & 1) ? TRUE :
         FALSE;
104   g_print("%s subflow 1\n", test_parameters_.subflow1_active?"
         Active":"Deactive");
105   test_parameters_.subflow2_active = (profile & 2) ? TRUE :
         FALSE;
```

```
106   g_print("%s subflow 2\n", test_parameters_.subflow2_active?"
          Active":"Deactive");
107   test_parameters_.subflow3_active = (profile & 4) ? TRUE :
          FALSE;
108   g_print("%s subflow 3\n", test_parameters_.subflow3_active?"
          Active":"Deactive");
109
110   test_parameters_.subflow_num+=test_parameters_.
          subflow1_active ? 1 : 0;
111   test_parameters_.subflow_num+=test_parameters_.
          subflow2_active ? 1 : 0;
112   test_parameters_.subflow_num+=test_parameters_.
          subflow3_active ? 1 : 0;
113
114   test_parameters_.video_session =(VideoSession)((profile & 8)
          >>3);
115   switch (test_parameters_.video_session) {
116     case VL2SRC:
117       g_print("Vl2 source is selected\n");
118       break;
119     case FILE_VIDEO:
120       g_print("FileVideo sequence is selected\n");
121       break;
122     default:
123       g_print("Vl2 source is selected\n");
124       break;
125   }
126
127   test_parameters_.test_directive = (TestSuite)((profile & 48)
          >>4);
128   switch (test_parameters_.test_directive) {
129       case MANUAL_RATE_CONTROLLING:
130         g_print("Manual Rate controller is selected.\n");
131         break;
132       case AUTO_RATE_AND_CC_CONTROLLING:
133         g_print("Automatic rate and congestion controller mode
              is selected.\n");
134         break;
135       case NO_CONTROLLING:
136         g_print("No rate or flow controlling is enabled.\n");
137         break;
138     }
139
140   test_parameters_.coding_active = (TestSuite2)((profile & 768)
          >>8);
```

```
141   g_print("Coding status: %d\n",test_parameters_.coding_active)
          ;
142   switch (test_parameters_.coding_active) {
143       case NO_CODING:
144         g_print("No protection code enabled..\n");
145         break;
146       case KODO:
147         g_print("Kodo enabled.\n");
148         break;
149       default: g_print("No protection code enabled..\n");
150     }
151
152
153   test_parameters_.random_detach = (profile & 64) > 0 ? TRUE :
          FALSE;
154   if(test_parameters_.random_detach){
155     g_print("Random join/detach is activated\n");
156   }
157
158   test_parameters_.record_to_file = (profile & 128) > 0 ? TRUE
          : FALSE;
159     if(test_parameters_.record_to_file){
160       g_print("Client: Record to file is activated\n");
161     }
162
163
164 }
165
166
167
168 #endif
```

A.8 setup_env.sh

```
1 #!/bin/sh
2 set -x
3
4 PATH1_VETH0_S="veth0"
5 PATH1_VETH0_R="veth1"
6
7 PATH2_VETH2_S="veth2"
8 PATH2_VETH2_R="veth3"
```

```
9
10 PATH3_VETH4_S="veth4"
11 PATH3_VETH4_R="veth5"
12
13 PATH4_VETH6_S="veth6"
14 PATH4_VETH6_R="veth7"
15
16 PATH5_VETH8_S="veth8"
17 PATH5_VETH8_R="veth9"
18
19 PATH6_VETH10_S="veth10"
20 PATH6_VETH10_R="veth11"
21
22 PEER_SND="peer_snd"
23 PEER_RCV="peer_rcv"
24
25 #Rimuovo eventuali NameSpace già esistenti
26 sudo ip netns del $PEER_SND
27 sudo ip netns del $PEER_RCV
28
29 #Rimuovo eventuali interfacce virtuali già esistenti
30 sudo ip link del $PATH1_VETH0_S
31 sudo ip link del $PATH2_VETH2_S
32 sudo ip link del $PATH3_VETH4_S
33 sudo ip link del $PATH4_VETH6_S
34 sudo ip link del $PATH5_VETH8_S
35 sudo ip link del $PATH6_VETH10_S
36
37 #Creazione dei collegamenti
38 sudo ip link add $PATH1_VETH0_S type veth peer name
      $PATH1_VETH0_R
39 sudo ip link add $PATH2_VETH2_S type veth peer name
      $PATH2_VETH2_R
40 sudo ip link add $PATH3_VETH4_S type veth peer name
      $PATH3_VETH4_R
41 sudo ip link add $PATH4_VETH6_S type veth peer name
      $PATH4_VETH6_R
42 sudo ip link add $PATH5_VETH8_S type veth peer name
      $PATH5_VETH8_R
```

```
43 sudo ip link add $PATH6_VETH10_S type veth peer name
      $PATH6_VETH10_R
44
45
46 #Eliminazione pipe già esistenti
47 sudo ipfw pipe 1 delete
48 sudo ipfw pipe 2 delete
49 sudo ipfw pipe 3 delete
50 sudo ipfw pipe 4 delete
51 sudo ipfw pipe 5 delete
52 sudo ipfw pipe 6 delete
53
54 #Sistemazione dei collegamenti
55 sudo ip link set dev $PATH1_VETH0_S up
56 sudo ip link set dev $PATH1_VETH0_R up
57
58 sudo ip link set dev $PATH2_VETH2_S up
59 sudo ip link set dev $PATH2_VETH2_R up
60
61 sudo ip link set dev $PATH3_VETH4_S up
62 sudo ip link set dev $PATH3_VETH4_R up
63
64 sudo ip link set dev $PATH4_VETH6_S up
65 sudo ip link set dev $PATH4_VETH6_R up
66
67 sudo ip link set dev $PATH5_VETH8_S up
68 sudo ip link set dev $PATH5_VETH8_R up
69
70 sudo ip link set dev $PATH6_VETH10_S up
71 sudo ip link set dev $PATH6_VETH10_R up
72
73 #Creazione dei NameSpace
74 sudo ip netns add $PEER_SND
75 sudo ip netns add $PEER_RCV
76
77 #Inserisco le interfacce nei NameSpace
78 sudo ip link set $PATH1_VETH0_S netns $PEER_SND
79 sudo ip link set $PATH1_VETH0_R netns $PEER_RCV
80
81 sudo ip link set $PATH2_VETH2_S netns $PEER_SND
```

```
82 sudo ip link set $PATH2_VETH2_R netns $PEER_RCV
83
84 sudo ip link set $PATH3_VETH4_S netns $PEER_SND
85 sudo ip link set $PATH3_VETH4_R netns $PEER_RCV
86
87 sudo ip link set $PATH4_VETH6_S netns $PEER_SND
88 sudo ip link set $PATH4_VETH6_R netns $PEER_RCV
89
90 sudo ip link set $PATH5_VETH8_S netns $PEER_SND
91 sudo ip link set $PATH5_VETH8_R netns $PEER_RCV
92
93 sudo ip link set $PATH6_VETH10_S netns $PEER_SND
94 sudo ip link set $PATH6_VETH10_R netns $PEER_RCV
95
96 #Configurazione loopback
97 sudo ip netns exec $PEER_RCV ip address add 127.0.0.1/8
      dev lo
98 sudo ip netns exec $PEER_RCV ip link set dev lo up
99
100
101 #Attivazione dei collegamenti
102 sudo ip netns exec $PEER_SND ip link set dev
      $PATH1_VETH0_S up
103 sudo ip netns exec $PEER_SND ip address add 10.0.0.1/24
      dev $PATH1_VETH0_S
104 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH1_VETH0_R up
105 sudo ip netns exec $PEER_RCV ip address add 10.0.0.2/24
      dev $PATH1_VETH0_R
106
107 sudo ip netns exec $PEER_SND ip link set dev
      $PATH2_VETH2_S up
108 sudo ip netns exec $PEER_SND ip address add 10.0.1.1/24
      dev $PATH2_VETH2_S
109 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH2_VETH2_R up
110 sudo ip netns exec $PEER_RCV ip address add 10.0.1.2/24
      dev $PATH2_VETH2_R
111
```

```
112 sudo ip netns exec $PEER_SND ip link set dev
      $PATH3_VETH4_S up
113 sudo ip netns exec $PEER_SND ip address add 10.0.2.1/24
      dev $PATH3_VETH4_S
114 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH3_VETH4_R up
115 sudo ip netns exec $PEER_RCV ip address add 10.0.2.2/24
      dev $PATH3_VETH4_R
116
117 sudo ip netns exec $PEER_SND ip link set dev
      $PATH4_VETH6_S up
118 sudo ip netns exec $PEER_SND ip address add 10.0.3.1/24
      dev $PATH4_VETH6_S
119 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH4_VETH6_R up
120 sudo ip netns exec $PEER_RCV ip address add 10.0.3.2/24
      dev $PATH4_VETH6_R
121
122 sudo ip netns exec $PEER_SND ip link set dev
      $PATH5_VETH8_S up
123 sudo ip netns exec $PEER_SND ip address add 10.0.4.1/24
      dev $PATH5_VETH8_S
124 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH5_VETH8_R up
125 sudo ip netns exec $PEER_RCV ip address add 10.0.4.2/24
      dev $PATH5_VETH8_R
126
127 sudo ip netns exec $PEER_SND ip link set dev
      $PATH6_VETH10_S up
128 sudo ip netns exec $PEER_SND ip address add 10.0.5.1/24
      dev $PATH6_VETH10_S
129 sudo ip netns exec $PEER_RCV ip link set dev
      $PATH6_VETH10_R up
130 sudo ip netns exec $PEER_RCV ip address add 10.0.5.2/24
      dev $PATH6_VETH10_R
131
132
133
134 #PIPE
```

```
135 sudo ipfw add pipe 1 ip from 10.0.0.1/24 to 10.0.0.2/24
      out
136 sudo ipfw add pipe 2 ip from 10.0.1.1/24 to 10.0.1.2/24
      out
137 sudo ipfw add pipe 3 ip from 10.0.2.1/24 to 10.0.2.2/24
      out
138
139 sudo ipfw add pipe 4 ip from 10.0.3.1/24 to 10.0.3.2/24
      out
140 sudo ipfw add pipe 5 ip from 10.0.4.1/24 to 10.0.4.2/24
      out
141 sudo ipfw add pipe 6 ip from 10.0.5.1/24 to 10.0.5.2/24
      out
```

A.9 setup_pipe.sh

```
1  #!/bin/bash
2  set -x
3
4  sudo ipfw pipe 1 config delay 0ms plr 0
5
6  sudo ipfw pipe 2 config delay 0ms plr 0
7
8  sudo ipfw pipe 3 config delay 0ms plr 0
9
10 sudo ipfw pipe 4 config delay 0ms plr 0
11
12 sudo ipfw pipe 5 config delay 0ms plr 0
13
14 sudo ipfw pipe 6 config delay 0ms plr 0
15
16 sudo echo "pipe 1 ---> RTT:0ms  PLR:0%" > stampa.txt
17 sudo echo "pipe 2 ---> RTT:0ms  PLR:0%" >> stampa.txt
18 sudo echo "pipe 3 ---> RTT:0ms  PLR:0%" >> stampa.txt
19
20 sleep 80
21
22 sudo ipfw pipe 1 config delay 0ms plr 0.1
23
```

```
24 sudo ipfw pipe 2 config delay 0ms plr 0.1
25
26 sudo ipfw pipe 3 config delay 0ms plr 0.1
27
28 sudo echo "pipe 1 ---> RTT:0ms  PLR:10%" > stampa.txt
29 sudo echo "pipe 2 ---> RTT:0ms  PLR:10%" >> stampa.txt
30 sudo echo "pipe 3 ---> RTT:0ms  PLR:10%" >> stampa.txt
```

A.10 $run_test.sh$

```
1  #!/bin/bash
2  programname=$0
3
4  #commento mandato in output nel caso in cui il numero di argomenti
       passato non sia corretto
5  function usage {
6      echo "usage: $programname [-p|-numero di profilo] [-d
           |-durata in secondi]"
7      echo "  -p --profile        determina il numero di
           profilo scelto dall'utente"
8      echo "      -d --duration           determina la
           durata del test (in secondi)"
9      exit 1
10 }
11
12 if [[ $# < 2 ]]
13 then
14   usage
15 fi
16
17 #Prendo gli argomenti passati e li inserisco nelle apposite variabili
18 PROFILE=0
19 DURATION=100
20
21 while [[ $# > 1 ]]
22 do
23 key="$1"
24 case $key in
25     -p|--profile)
```

```
26      PROFILE="$2"
27      shift
28      ;;
29      -d|--duration)
30      DURATION="$2"
31      shift
32      ;;
33      --default)
34      ;;
35      *)
36      ;;
37  esac
38  shift
39  done
40
41  #Stampa indicante la scelta del profilo e della durata
42  echo "
       .-------------------------------------------------------------.
       "
43  echo "| Il Test viene eseguito coi seguenti parametri
                    |"
44  echo "| RTP profile: "$PROFILE
45  echo "| Duration:    "$DURATION
46  echo "
       '-------------------------------------------------------------'
       "
47
48
49
50  PEERSND="peer_snd"
51  PEERRCV="peer_rcv"
52  SERVER="./server"
53  CLIENT="./client"
54
55  cleanup()
56  # example cleanup function
57  {
58    sudo killall client
59    sudo killall server
60  }
```

```
61
62 control_c()
63 # run if user hits control-c
64 {
65   echo -en "\n*** Uscita... ***\n"
66   cleanup
67   exit $?
68 }
69
70 trap control_c SIGINT
71
72 gcc clientmod.c -g -o client `pkg-config --cflags --libs
      gstreamer-1.0`
73 gcc servermod.c -g -o server `pkg-config --cflags --libs
      gstreamer-1.0`
74
75 ip netns exec $PEERSND $SERVER "--profile="$PROFILE  >
      file.txt 2>&1 &
76
77 sleep 2
78 ip netns exec $PEERRCV $CLIENT "--profile="$PROFILE &
79 sleep $DURATION
80
81 cleanup
```

Bibliografia

[1] *What is GStreamer?* 2015. URL: https://gstreamer.freedesktop.org/.

[2] *Multipath RTP (MPRTP)*. 2015. URL: https://tools.ietf.org/html/draft-ietf-avtcore-mprtp-01.

[3] *MPRTP plugin*. 2015. URL: https://github.com/multipath-rtp/gst-mprtp.

[4] Rudolf Ahlswede et al. «Network information flow». In: *IEEE Transactions on information theory* 46.4 (2000), pp. 1204–1216.

[5] *The Benefits of Coding over Routing in a Randomized Setting*. 2003. URL: http://www.its.caltech.edu/~tho/i1.pdf.

[6] *RTP: A Transport Protocol for Real-Time Applications*. 1996. URL: https://www.ietf.org/rfc/rfc1889.txt.

[7] *Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)*. 2003. URL: https://tools.ietf.org/html/rfc3605.

[8] Jörg Ott Varun Singh Saba Ahsan. «MPRTP: multipath considerations for real-time media». In: *MMSys '13 Proceedings of the 4th ACM Multimedia Systems Conference*. A cura di USA ©2013 ACM New York NY. 2013. URL: http://dl.acm.org/citation.cfm?id=2484002.

[9] Wim Taymans. *GStreamer Application Development Manual(1.2.3)*. 2013. URL: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf.

[10] *Kodo - Network Coding*. 2016. URL: http://docs.steinwurf.com/kodo/overview_kodo.html.

[11] *Gstreamer Modules*. URL: https://gstreamer.freedesktop.org/modules/.

[12] *RTP Payload Format for Generic Forward Error Correction*. 2011. URL: https://tools.ietf.org/html/rfc5109.

[13] Luca Ussi. *Implementazione di un modulo di comunicazione multipath per sistemi Real-Time*. 2016. URL: http://puma.isti.cnr.it/linkdoc.php?idauth=1&idcol=1&icode=2016-TH-001&authority=cnr.isti&collection=cnr.isti&langver=it.

[14] *Multipath RTP (MPRTP)*. 2016. URL: https://tools.ietf.org/html/draft-irtf-nwcrg-network-coding-taxonomy-01.

[15] *Librerie kodo-c*. 2016. URL: http://docs.steinwurf.com/kodo/kodo-c/index.html#kodo-c.

[16] Majid Ghaderi, Don Towsley e Jim Kurose. «Reliability gain of network coding in lossy wireless networks». In: *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE. 2008, pp. 2171–2179.

[17] *Network Coding vs. Erasure Coding: Reliable Multicast in Ad hoc Networks*. 2008. URL: http://nrlweb.cs.ucla.edu/publication/download/463/NCvsEC_ACITA2008.pdf.

[18] *GStreamer Plugin Writer's Guide*. 2016. URL: https://gstreamer.freedesktop.org/data/doc/gstreamer/devel/pwg/pwg.pdf.

[19] R. E. KALMAN. *A New Approach to Linear Filtering and Prediction Problems*. 2011. URL: https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

[20] *Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)*. 2007. URL: https://tools.ietf.org/html/rfc4820.

[21] Luigi Rizzo. *The dummynet project*. 2010. URL: http://info.iet.unipi.it/~luigi/dummynet/.

[22] Marta Carbone Luigi Rizzo. *Dummynet Revisited*. 2010. URL: http://info.iet.unipi.it/~luigi/papers/20100304-ccr.pdf.

[23] *29.4. IPFW Prev Chapter 29. Firewalls*. URL: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html.

[24] *speedometer*. 2015. URL: https://excess.org/speedometer/.

[25] Original Manpage by Eric W. Biederman. *IP-NETNS(8)*. 2013. URL: http://man7.org/linux/man-pages/man8/ip-netns.8.html.

[26] *Libav - open source audio and video processing tools*. 2016. URL: https://libav.org/.