



Università di Pisa

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica - Curriculum Professionalizzante

**Simulatore per trasmissioni multimediali
Real-Time da drone a terra**

Candidato:
Massimo Puddu

Tutore Aziendale:
Dott. F. Manlio Bacco

Tutore Accademico:
Dott. Gabriele Mencagli

Indice

Introduzione	6
1 Scenario	8
1.1 Sviluppo	8
1.2 Real Time Protocol (RTP)	9
1.3 Scheduling controller	9
1.4 Group of picture (GOP)	9
1.5 Peak Signal-to-Noise Ratio (PSNR)	9
2 Flussi multimediali RTP/UDP	12
2.1 Analisi RFC 3550	12
2.2 RTP caso d'uso	14
2.2.1 RFC 6184	14
2.2.1.1 RFC 6184: Data Header	15
2.2.1.2 Modalità di pacchettizzazione	16
2.2.1.3 Fragmentation Units (FUs)	17
2.2.2 Algoritmi di compressione video	19
2.2.2.1 H.264 approfondimento	20
2.2.2.2 FU(s) come ottenere il frame type corrente	21
3 Simulatore: Architettura e Funzionamento	22
3.1 Sender	23
3.1.1 Canali Fisici	23
3.1.2 Classe Operatore	23
3.1.3 Classe Sender	23
3.1.4 Scheduler	25
3.1.5 Comunicazione Scheduler-Process	26
3.1.6 Classe Timing	27
3.2 Receiver	28
3.2.1 Threads e Process	28

3.2.2	Comunicazioni	29
3.2.3	listenerThread	31
3.2.4	GOPThread	31
3.2.5	ReaderPacket	33
3.2.6	Ottenere i pacchetti	34
3.2.6.1	Algoritmo per il salvataggio del GOP	34
3.2.7	DecoderThread	35
3.2.8	PlotProc	36
3.2.9	statThread	37
3.2.9.1	Ritardo	38
3.2.9.2	Variabilità del ritardo (jitter)	38
3.2.9.3	Pacchetti fuori ordine	38
3.2.9.4	Lunghezza media della perdita	39
3.2.9.5	Numero di pacchetti persi	39
3.2.10	Segnali	39
3.2.11	Gstreamer	40
4	Manuale utente e sviluppatore	42
4.1	Requisiti e dipendenze	42
4.1.1	Dipendenze di sistema	42
4.1.2	Dipendenze del sender	44
4.1.3	Dipendenze del client	44
4.2	Primo utilizzo	45
4.3	Config	45
4.3.1	File di configurazione del Sender	45
4.3.2	File di configurazione del Receiver	46
4.4	execute.sh	47
4.5	ImagePicker	48
4.6	Script	48
4.7	Sender	48
4.8	statistic	48
4.9	Come utilizzare il sistema	49
4.10	Modifiche al sistema	51
4.10.1	Aggiungere statistiche	51
4.10.2	Loopback Interface	52
4.10.3	Array di dimensione prestabilita	52

5	Test e dimostrazione d'uso	54
5.1	Test	54
5.1.1	Risultati	55
5.2	Dimostrazione d'uso	56
6	Conclusioni	60

Introduzione

L’obiettivo di questo tirocinio è l’implementazione di un emulatore che riproduce la trasmissione di un video da un drone a terra. Il drone dispone di una videocamera. Al fine di rendere più robusta la comunicazione e avere maggiore disponibilità di banda, l’emulatore permette di creare un canale aggregato che sfrutta più canali fisici reali. A tale scopo un modulo di *scheduling* viene utilizzato per decidere quali e quanti canali utilizzare sulla base delle statistiche di ciascun canale fisico. Inoltre, per rendere la comunicazione anche più robusta, è prevista la possibilità di trasmettere il flusso multimediale su più canali contemporaneamente (in modo duplicato o non), così da aumentare la probabilità di una corretta ricezione a destinazione. Il modulo di scheduling dovrà utilizzare i canali in modo intelligente, ovvero sfruttare meno canali e meno repliche possibili, mantendo la migliore qualità video. A tale scopo un modulo definito di *scheduling controller* ha il compito di monitorare lo stato dei canali in termini di ritardo, jitter e tasso di perdita dei pacchetti, così da aggiornare attraverso una funzione di ottimizzazione i pesi dello scheduler. Il ricevitore si deve occupare di ordinare ed eventualmente scartare ricezioni multiple dovute alle repliche inviate sui canali.

Nella fase iniziale del lavoro ci si è concentrati sull’analisi dei requisiti del sistema. I requisiti sono descritti nei capitoli sottostanti:

- Scenario, descrizione di un uso tipico del sistema e della sua composizione 1;
- Real-time Transport Protocol (RTP), per la trasmissione flussi multimediali 2;
- Simulatore: Architettura e Funzionamento, mostra la composizione del sistema 3;
- Manuale sviluppatore, chiarisce gli algoritmi che si sono utilizzati e come modificare alcune parti del sistema 4;

- Manuale utente, spiega a un utente base come utilizzare il sistema 4.2;
- Test, mostra il sistema in esecuzione e con quali opzioni è stato testato 5;
- Conclusione, per una visione complessiva di ciò che è stato sviluppato 6.

In questi capitoli viene descritto come si è arrivati alla implementazione del sistema, che si ritiene essere la più adatto per il caso d'uso proposto.

Capitolo 1

Scenario

Un utilizzo reale del sistema è quello in cui si deve trasmettere un video da drone a terra. È preferibile che il video sia sempre disponibile. Per questa ragione al drone vengono montate tre schede Sim di diversi operatori. Quando il drone inizia la trasmissione si vuole evitare di utilizzare tutti i canali contemporaneamente. In questo caso i canali sono rappresentati dalla scheda degli Operatori. Prima di inviare i pacchetti, bisogna effettuare analisi riguardanti la qualità del servizio e banda, altrimenti si rischia di utilizzare tutta la banda o che il video sia distorto a causa di un segnale debole. Si dovrà quindi costruire un sistema che tenga conto di questi fattori.

1.1 Sviluppo

La progettazione del sistema inizia dalla consapevolezza che si riceve in input un file PCAP, dove ogni pacchetto nel suo payload contiene pezzi di video.

Si deve implementare un modulo denominato di scheduling che deve essere in grado di leggere il file PCAP e di schedulare un determinato pacchetto, nelle giuste tempistiche, a uno o più canali fisici, in base a statistiche di canale che si raccolgono man mano che questi si utilizzano. Tutto questo sottosistema viene detto Sender.

Oltre a questo modulo, occorre un ricevitore che sia in grado di ottenere i pacchetti spediti dal Sender. Questo ha il nome di Receiver ed ha bisogno di N canali di ricezione, tanti quanti i canali di invio del Sender. Ricevuti i pacchetti si va a comporre un video con un processo di decodificazione su di essi. Il Receiver deve anche permettere la visualizzazione del video ottenuto attraverso i pacchetti e deve essere in grado di capirne la qualità visiva.

La tipologia di comunicazione tra Sender e Receiver è denominata *multipath*. Si definisce multipath una connessione in grado di usare nello stesso momento più canali fisici di comunicazione tra sorgente e destinazione.

1.2 Real Time Protocol (RTP)

All'interno del PCAP, che viene passato in input, deve essere presente un flusso RTP. Quando si parla di flusso RTP si intende un insieme di pacchetti di rete che è possibile decodificare come pacchetti RTP. Tale flusso consente il trasferimento di video multimediali, ed è una caratteristica che si chiede al sistema.

1.3 Scheduling controller

L'utilizzo di uno Scheduler controller risulta utile per distribuire il carico di lavoro tra i canali. Si tratta di uno modulo che prende in input dati statistici associati a un canale. Attraverso questi dati compie dei calcoli statistici che permette di determinare quanti e quali canali utilizzare per saturare il meno possibile la banda, mantenendo una buona qualità video.

1.4 Group of picture (GOP)

Un GOP è creato da un codec video ed è un insieme di più immagini in sequenza. Un video è composto da più GOP che si susseguono continuamente. Un GOP è formato da frames che si susseguono, dove ogni frame ha un significato preciso all'interno del GOP e può incidere sul frame successivo. Sono tipicamente utilizzati dal codec di H.264.

1.5 Peak Signal-to-Noise Ratio (PSNR)

Il PSNR è una metrica che si utilizza per calcolare la differenza di qualità di un'immagine compressa rispetto all'originale. La definizione di questo valore è dato dal rapporto tra la massima potenza di un segnale e la potenza di rumore che può invalidare la fedeltà della sua rappresentazione compressa. Per esprimere il PSNR si usa una scala logaritmica di decibel. Dei valori tipici per le immagini compresse sono dai 30 ai 50 dB. Se un'immagine non presenta distorsioni si otterrà un PSNR molto alto. Per scelte implementative, quando

l'immagine distorta è molto fedele all'originale, si impone che il suo PSNR sia uguale a 60 dB.

Capitolo 2

Flussi multimediali RTP/UDP

Il Real-time Transport Protocol (RTP) è un protocollo sviluppato per spedire audio e video su reti IP. È utilizzato spesso per la comunicazione e per sistemi di intrattenimento, come ad esempio la telefonia, video conferenze e in generale nello streaming multimediale.

RTP è solitamente usato utilizzando a livello trasporto l'UDP. Può essere usato insieme all'RTPCP (RTP Control Protocol), allo scopo di monitorare il quality of service e di migliorare la sincronizzazione. Al giorno d'oggi RTP è uno standard sviluppato dall' organizzazione Audio-Video Transport Working Group of the Internet Engineering Task Force, questo standard è definito dal RFC 3550¹. È utile analizzarlo per capire esattamente come è formato un pacchetto RTP e quali sono gli elementi di interesse per la realizzazione del sistema.

2.1 Analisi RFC 3550

I pacchetti RTP sono creati a livello applicativo. Ogni pacchetto RTP, creato da un' applicazione, inizia con un RTP data header, il quale sarà formato nel seguente modo:

¹<https://tools.ietf.org/html/rfc3550>

RTP packet header																																								
Offsets	Octet	0								1								2								3														
Octet	Bit [a]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31							
0	0	Version	P	X	CC	M	PT							Sequence number																										
4	32	Timestamp																																						
8	64	SSRC identifier																																						
12	96	CSRC identifiers ...																																						
12+4xCC	96+32xCC	Profile-specific extension header ID																Extension header length																						
16+4xCC	128+32xCC	Extension header ...																																						

Figura 2.1: Specifica di un RTP header.
Le parti segnate in rosso sono opzionali.

RTP header ha un minimo di 12 bytes. I campi del header sono i seguenti:

- Version: (2 bits) Indica la versione del protocollo, attualmente si usa la 2.
- P (Padding): (1 bit) Usato per indicare se ci saranno degli extra padding bytes alla fine del pacchetto. Può essere usato per riempire un blocco di una certa dimensione come richiesto da algoritmo di crittografia. L'ultimo byte del padding indica quanti bytes sono stati riempiti.
- X (Extension): (1 bit) Indica la presenza di un Extension header. Si trova immediatamente prima del payload. L'extension è per i programmi o per un profilo specifico.
- CC (CSRC count): (4 bits) Indica il numero di CSRC entry, segue l'SSRC.
- M (Marker): (1 bit) Segnale usato a livello applicativo in modo specifico per il profilo, Se 1 significa che il pacchetto ha qualche rilevanza per l'applicazione.
- PT (Payload type): (7 bits) Indica il formato del payload e quindi ne determina l'interpretazione per l'applicazione. I valori sono specifici per il profilo e possono essere assegnati dinamicamente.
- Sequence number: (16 bits) Il numero di sequenza è incrementato per ogni RTP data spedito ed è utilizzato dal receiver per indicare un packet

loss e per calcolare l'out of order. Il valore iniziale del numero di sequenza dovrà essere randomizzato per rendere gli attacchi informatici più difficili da attuare.

- Timestamp: (32 bits) Usato dal receiver per riprodurre i campioni al giusto momento ed intervallo. La granularità è decisa dall'applicazione che lo genera.
- SSRC: (32 bits) *Synchronization source identifier* identifica univocamente l'origine dello stream. La Synchronization source in una stessa sessione RTP sarà univoca.
- CSRC: (32 bits ognuno, il numero di entries è indicato dal campo CSRC count) I contributing source IDs enumerano le sorgenti di contribuzione a uno stream che è stato configurato per avere più sorgenti.
- Header extension: (opzionale, la sua presenza è indicata dal campo Extension) Le prime parole da 32 bit contengono un identificatore specifico di profilo(16 bits) e lunghezza(16 bits), che specifica la lunghezza dell'extension in unità da 32 bit. escludendo i 32 bits dell'extentions header.

2.2 RTP caso d'uso

L'uso di un flusso RTP varia in base all'applicazione e al profilo specifico che si vuole utilizzare. Nel nostro caso si è utilizzato il profilo con payload type 96, in quanto è un tipo dinamico che gestisce audio e video. Tutti i vari tipi di payload type sono descritti dal RFC 3551². // Durante il tirocinio si è utilizzato un RTP data che contiene video in H.264. L'uso di quest'ultimo insieme al RTP è a sua volta uno standard, si può andare a visionare RFC 6184³ per maggiori informazioni.

2.2.1 RFC 6184

Ricapitolando, si è deciso che si deve usare il payload type 96 in quanto è adatto allo streaming video. Questo ci dice che nei dati RTP può portare vari formati video. Analizziamo desso come un player video decodifica questi dati; esso ha bisogno di un file di supporto che gli specifica la tipologia di video che

²<https://tools.ietf.org/html/rfc3551>

³<https://tools.ietf.org/html/rfc6184>

gli è stata inviata. Si tratta dunque di mappare il payload type ad un Session Description Protocol (SDP). RFC 6184 dice che si deve mappare un SDP ad un video in H.264. Tutte le informazioni saranno rappresentate attraverso una stringa. Il formato è in genere formato da **Nome_parametro=valore**.

Un SDP semplice sarà formato nel seguente modo:

- la linea contenente il parametro "m=" del SDP dovrà seguire il valore "video", quindi "m=video".
- nella stessa linea in cui è presente "m=" si potrà specificare anche il payload type del flusso RTP aggiungendo a fianco del valore di "m=val" la stringa "RTP/AVP 96".
- si deve specificare la decodifica con la seguente stringa "a=rtpmap" nella stessa linea si continua scrivendo "H264|90000" in questo modo viene indicata la tipologia di video e clockrate.

2.2.1.1 RFC 6184: Data Header

Dopo all'RTP data header si susseguono i dati nel payload. Abbiamo visto che all'interno del header è presente il payload type, il che ci dà un'indicazione su cosa andremo a trovare nel payload. Tuttavia c'è una domanda che ancora rimane senza risposta:

Cosa succede se un determinato tipo di formato video ha bisogno di informazioni supplementari per essere decodificato?

Per rispondere a questa domanda dobbiamo andare ad analizzare come è fatto il payload del nostro caso d'uso. All'interno del payload si trova una struttura dati chiamata Network Abstraction Layer (NAL). Il codificatore NAL incapsula uno slice(cfr. 2.2.2.1) output, di un'applicazione che codifica il video in un Network Abstraction Layer Units (NALUs), questo è ideale per trasmissioni di pacchetti di rete o per ambienti multiplexed packet-oriented. Il processo di incapsulazione viene fatto utilizzando Annex B di H.264 per trasmettere il NALUs su reti bytestream-oriented. Internamente un NAL è composto da NAL units. Un NAL unit consiste in un byte header e in un payload byte string. L'header indica il tipo del NAL unit, la potenziale presenza di bit di errore o errori di sintassi nel NAL unit payload e contiene informazioni riguardanti l'importanza di un determinato NAL unit per il processo di decodifica. La specificazione presa in esame del RTP payload è fatta in modo da ignorare il bit string nel NAL unit payload.

Vediamo ora come è formato un NALUs. Tutti i NAL units sono composti da un singolo otteto di NAL unit type, che serve anche esso come secondo header del formato del RTP payload. Il NAL unit header è il seguente:

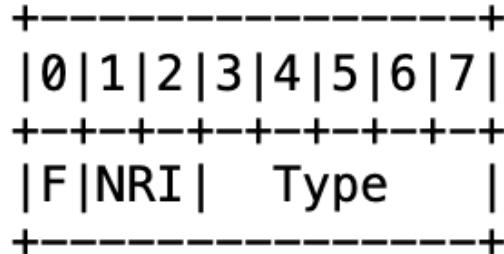


Figura 2.2: Serve per capire come interpretare il resto del pacchetto RTP.

Ogni elemento ha il seguente significato:

- F (1 bit) forbidden_zero_bit. Se vale 0 significa che l'otteto del NAL unit type e payload non dovrà contenere bit di errore o violazioni semantiche syntax violations. Se vale 1 indica che l'otteto NAL unit type e payload potrebbero contenere gli errori sopracitati. H.264 per specifica richiede che questo bit sia uguale a 0.
- NRI (2 bits) nal_ref_idc. Il valore 00 indica che il contenuto del NAL unit non è utilizzato per costruire immagini che predicono frame successivi. Questo NAL può essere scartato senza rischiare di compromettere l'integrità della foto di referenza. Valori maggiori di 0 indicano che la decodifica è essenziale per mantenere l'integrità della foto di referenza. La semantica del valore 00 e un valore che non sia zero rimane invariato per la specificazione di H.264.
- type (5 bites) nal_unit_type. Questo componente specifica il packet type della tabella che verrà illustrata successivamente (cfr. 2.3).

È necessario descrivere i vari nal_unit_type per comprendere quale di questi risulta essere il più idoneo ai fini del tirocinio.

2.2.1.2 Modalità di pacchettizzazione

Ci sono tre diverse varianti in cui si può decidere di pacchettizzare un pacchetto RTP in base a come esso dovrà essere utilizzato:

- Single NAL unit mode, usato per sistemi di conversazione per soddisfare ITU-T Recommendation H.241

- Non-interleaved mode, è anche esso usato per sistemi di conversazione, ma possono non soddisfare ITU-T Recommendation H.241.
- Interleaved mode, è pensato per sistemi che non richiedono una bassa latenza. La interleaved mode permette la trasmissione di NAL unit fuori dall'ordine di decodifica delle NAL unit.

In base al tipo di pacchettizzazione si possono utilizzare dei specifici NAL unit type. Di seguito la tabella che elenca le varie possibilità:

Payload Type	Packet Type	Single NAL Unit Mode	Non-Interleaved Mode	Interleaved Mode
0	reserved	ig	ig	ig
1-23	NAL unit	yes	yes	no
24	STAP-A	no	yes	no
25	STAP-B	no	no	yes
26	MTAP16	no	no	yes
27	MTAP24	no	no	yes
28	FU-A	no	yes	yes
29	FU-B	no	no	yes
30-31	reserved	ig	ig	ig

Figura 2.3: Riassunto dei possibili NAL types, per ogni tipo di pacchettizzazione (yes = allowed, no = disallowed, ig = ignore).

Nel nostro caso si utilizza FU-A che andiamo subito ad approfondire.

2.2.1.3 Fragmentation Units (FUs)

Questo tipo di payload permette la frammentazione del NAL unit in parecchi pacchetti RTP. Utilizzando la frammentazione a livello applicativo, invece che affidarsi a un layer inferiore di rete (come IP), si ottengono i seguenti vantaggi:

- Il formato del payload permette il trasporto di NAL units più grandi di 64 kbytes su una rete IPv4, risulta particolarmente utile se si utilizza un video pre註冊ato, soprattutto se quest'ultimo è in alta definizione.
- Il meccanismo di frammentazione permette la frammentazione di singole NAL unit, facilitando così la correzione di errori.

La frammentazione è definita solo per singole NAL unit e non per ogni aggregazione di pacchetti. Un frammento di una NAL unit è composta da un numero intero di otteti consecutivi del NAL unit. Ogni otteto del NAL unit

deve obbligatoriamente far parte di un frammento del NAL unit. I frammenti di una stessa NAL unit devono essere spediti in ordine consecutivo con il numero di sequenza del pacchetto RTP crescente. Analogamente un NAL unit deve essere riassemblato utilizzando i numeri di sequenza dei pacchetti RTP in ordine crescente.

Quando un'unità NAL viene frammentata e convogliata all'interno di unità di frammentazione (FUs) questa viene definita come fragmented NAL unit. Attenzione però i FUs non devono essere nidificati, cioè un FU non deve contenere un altro FU.

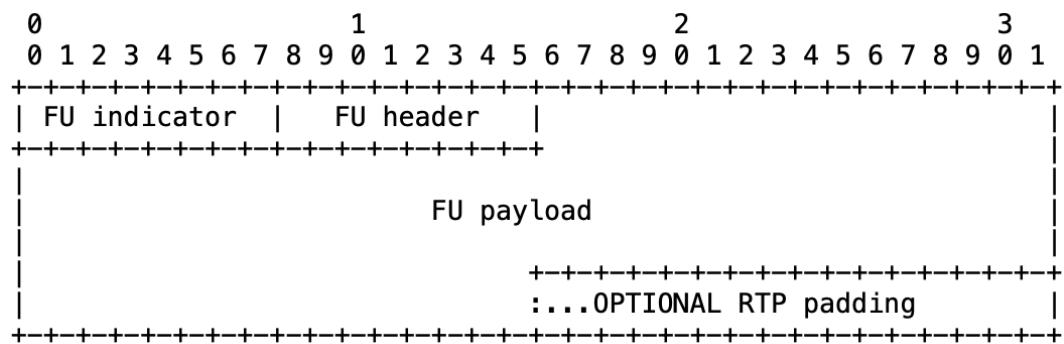


Figura 2.4: Un FU-A consiste in un fragment unit undicator rappresentato da un otteto, di un fragmentation unit header composto anche esso da un otteto e un fragmentation unit payload

L'ottetto del FU indicator ha il seguente formato:

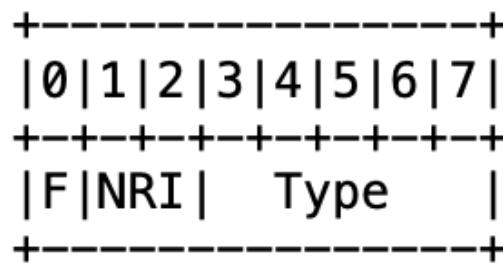


Figura 2.5: FU-A indicator

Se nel campo type sono presenti i valori 28 e 29 allora l'ottetto del FU indicator identifica rispettivamente un FU-A e un FU-B. L'uso di F è stato descritto precedentemente, mentre il valore di NRI deve essere impostato rispettando il valore del NRI del fragment NAL unit.

L'ottetto del FU header è la seguente:

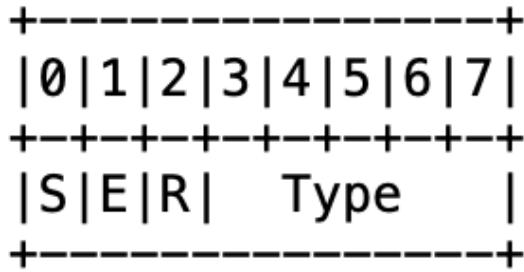


Figura 2.6: FU-A header

- S (1 bit): Quando vale 1 viene indicato l'inizio di un fragment NAL unit. Se vale 0 il FU payload successivo non è l'inizio del fragment NAL unit payload.
- E (1 bit) Se è equivalente a 1 viene indicata la fine di un fragment NAL unit. In questo caso l'ultimo byte del payload è anche l'ultimo byte del fragment NAL unit. Quando il successivo FU payload non è l'ultimo frammento del fragmented NAL unit, l'End bit vale 0 .
- R (1 bit) il Reserved bit deve essere impostato a 0 e ignorato dal ricevitore.
- Type (5 bit) indica il NAL unit payload type (cfr. 2.2.2.2)

2.2.2 Algoritmi di compressione video

Abbiamo visto che i pacchetti RTP possono ospitare vari formati video, facciamo quindi una piccola digressione per capire che tipo di codifica inserire e come essa si possa ottenere. Si entra quindi nel campo della compressione video. Quando un video viene compresso si ottengono svariati frame compressi che sono generati attraverso un algoritmo. Ci sono tanti algoritmi che portano a una compressione video, ognuno dei quali ha vantaggi e svantaggi rispetto ad altri algoritmi. L'aspetto più importante di questi è ottenere la miglior qualità, comprimendo più dati possibili. Questi algoritmi vengono chiamati *picture types* o *frame types*.

I principali pictures types, utilizzati più spesso nei video, sono tre:

- **I-frames** (Intra-coded picture) sono i meno compressi, ma non richiedono altri frame per essere decodificati. Rappresentano un intera immagine come ad esempio un PNG.

- **P-frames** (Predicted picture) possono usare dati da frame precedenti per essere decompressi. Questi frame sono più comprimibili degli I frame. Contiene solo i cambiamenti che sono avvenuti rispetto a un frame precedente. Il codificatore non ha bisogno di memorizzare le parti dell'immagine che non hanno subito modifiche, questo comporta una riduzione di spazio.
- **B-frames** possono usare frame precedenti ma anche frame successivi per essere decodificati. Questi hanno il maggior grado di compressione. Questo è possibile grazie al processo di salvataggio delle differenze tra le immagini precedenti e quella successiva, per specificarne il contenuto.



Figura 2.7: Rappresenta la composizione tipica di un GOP (Group of picture).

2.2.2.1 H.264 approfondimento

Si introduce ora il concetto di slice. Uno slice è una regione spazialmente distinta di un frame che è codificato separatamente rispetto a ogni altra regione dello stesso frame. I-slices, P-slices, e B-slices prendono il posto di I, P, e B frames.

Un concetto simile sono i macroblocks. Di solito i frame sono divisi in macroblocchi. Un prediction types può essere selezionato in un macroblocco invece che sull'intero frame. In una compressione generale i macroblocchi vengono utilizzati nel seguente modo:

- **I-frames** possono contenere solo intra macroblocks.
- **P-frames** possono contenere intra macroblocks e prediction macroblocks.
- **B-frames** possono contenere intra, predicted, o bi-predicted macroblocks.

In H.264 i frame possono essere segmentati in macroblocchi detti slices, l'encoder decide il prediction style in ogni slice individualmente. Si può andare ora a specificare meglio i type frame di H.264:

- **(I) Intra-coded frames/slices** contengono un'immagine completa. Sono codificati senza considerare altri frame, ma solo se stessi. Possono

anche essere generati nel caso in cui è impossibile creare un immagine con P-frame o B-frame.

- **(P) Predicted frames/slices** richiede la decodifica di altre immagini prima di poter essere decodificato. Utilizza una o più immagini precedenti per poter essere decodificato. Può fare riferimento a immagini precedenti nel momento della decodifica. Potrebbe anche richiedere un certo ordine tra i frame precedenti.
- **(B) Bi-directional predicted frames/slices (macroblocks)** Richiede la codifica di un insieme di blocchi prima di poter essere decodificato. Viene consentita una, due o più di due immagini precedentemente decodificate come riferimenti durante la decodifica e può avere qualsiasi relazione arbitraria di ordine di visualizzazione relativa alle immagini utilizzate per la sua previsione. Più frame "vicini" vengono impiegati per la codifica del B-slice, meno spazio il B-frame utilizzerà.

2.2.2.2 FU(s) come ottenere il frame type corrente

Come visto precedentemente FU-A è composto da un FU indicator e un FU header. L'informazione per sapere su quale tipo di slice stiamo per andare a lavorare si trova nell'ottetto di FU header, in particolare nel campo type. Infatti se questo vale 5 vuol dire che stiamo lavorando su un frame I, invece se vale 1 stiamo lavorando con un frame P. Nel caso del tirocinio si è utilizzato un codificatore che utilizza solo I-frame e P-frame. Quindi il nostro GOP sarà formato nel seguente modo:



Figura 2.8: Viene rappresentata la codifica dei group of picture (GOP) utilizzata nel tirocinio.

Capitolo 3

Simulatore: Architettura e Funzionamento

L'architettura del simulatore si compone di due moduli principali:

- Sender, crea N canali di invio che saranno comandati da un modulo chiamato TEE e un modulo di scheduling.
- Receiver, ha N canali di destinazione. Questi canali inoltrano i pacchetti a un modulo che li gestisce. Qui saranno analizzati per raccogliere statistiche, le quali verranno inviate al modulo di feedback del sender. Inoltre sempre in questo modulo i pacchetti vengono filtrati per fare in modo che arrivi un nuovo flusso RTP a Gstreamer che si occuperà della riproduzione video. I moduli rimanenti si occupano di creare i frame del video, salvarli nel disco e calcolarne il PSNR con delle immagini campione corrispondenti.

Il grafico seguente mostra l'architettura del sistema creato:

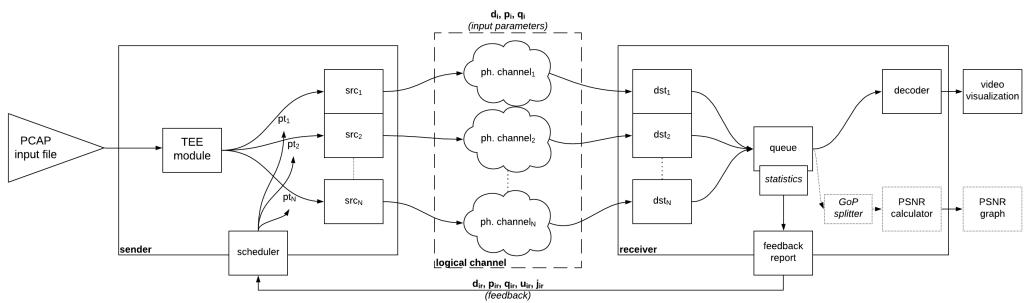


Figura 3.1: Schema del sistema.

3.1 Sender

Da un punto di vista logico, all'avvio della simulazione, il Sender crea N canali per la trasmissione dei flussi RTP. Successivamente si occupa di creare i Python Process (**PP**) delle statistiche e il PP che crea le immagini campione che saranno utilizzate dal Receiver per il calcolo del PSNR. Una volta creati, prima di iniziare l'inoltro dei pacchetti RTP, si attende che il Receiver si connetta al PP delle statistiche perché si tratta di una connessione TCP. Quando la connessione è stata stabilita entra in esecuzione lo scheduler. Quest'ultimo si occupa di inoltrare l'i-esimo pacchetto da spedire a un certo numero di canali e su quali canali inoltrare in base alle disposizioni ricevute dal modulo di scheduling controller. Esso prende queste decisioni in base ai dati statistici raccolte dal canale di feedback.

3.1.1 Canali Fisici

Un canale fisico è implementato come un PP che internamente va ad istanziare queste due classi:

- Operatore, è la classe che si occupa di impostare un pacchetto RTP in modo che la classe Sender si trovi con un pacchetto pronto all'uso. E di inoltrare un pacchetto RTP al Sender.
- Sender, si occupa del invio in rete di un pacchetto RTP.

3.1.2 Classe Operatore

Quando viene istanziato prende in input un Sender, un ip, una porta e un nome che rappresenta l'operatore. Va quindi a creare un socket per l'inoltro dei pacchetti RTP. Questa classe ha al suo interno una funzione che si chiama send(). Prende in input un pacchetto di rete ma deve contenere solo il livello applicativo, nel nostro caso dovrà essere tutta e sola la parte del pacchetto RTP e il formato del pacchetto deve essere in bytes. Operatore presenta un'altra funzione, utilizzata solo dallo scheduler, che modifica un pacchetto per renderlo utilizzabile dalla funzione send().

3.1.3 Classe Sender

Durante l'istanziazione prende in input valori alpha e scale per calcolare le distribuzioni gamma da cui andrà a prendere valori come delay, numero di

pacchetti da perdere oppure per decidere se perdere pacchetti o meno. Non sempre questi valori verrano usati, infatti si potrà selezionare una modalità in cui questi valori saranno ignorati. Andiamo ad affrontare però solo il caso in cui verranno utilizzati, perché è il caso di interesse. Prima di descrivere l'algoritmo è opportuna precisare la presenza di una delayed list e di un Thread che si occupa di spedire i pacchetti. Ogni pacchetto dovrà subire un delay e non potendolo spedire immediatamente questo verrà inserito nella delayed list. Nella lista il pacchetto sarà affiancato da un valore che specifica il momento in cui dovrà essere spedito. Il Thread viene chiamato send_delayed. Tra questo e la classe Sender è presente una coda che permette di comunicare tra di loro. Inizialmente send_delayed si blocca sulla coda in attesa di pacchetti

L'algoritmo è il seguente:

1. Viene controllato se si è in un evento di perdita, cioè se precedentemente è stato indicato di scartare N pacchetti successivi. Se questo risulta vero si scarta il pacchetto e si decrementa il numero di pacchetti da perdere terminando così la spedizione.
2. In caso contrario estraggo un valore dalla distribuzione gamma dell'evento perdita. Che verrà confrontato con un numero casuale.
 - (a) Se il numero casuale è maggiore di quello della distribuzione si estrae un numero casuale dalla distribuzione gamma del delay e si affianca questo valore al pacchetto creando una coppia. Successivamente viene aggiunto un timestamp, sempre in bytes, alla fine del pacchetto, in questo modo viene memorizzato il tempo di invio senza tenere conto del delay simulato. Questo dato è utile al Receiver per calcolare delle statistiche. Infine una volta modificato il pacchetto si inserisce la coppia <pacchetto, delay> nella coda che comunica tra Thread e Sender.
 - (b) Invece se è minore del numero estratto, scarta il pacchetto, entro in un evento di perdite e quindi estraggo un numero dalla distribuzione del numero di pacchetti da scartare. Questo valore è quello che si è utilizzato nel passo (2).
3. send_delayed si sblocca perché si è inserito un pacchetto nella coda. Dalla coppia si preleva il delay. Questo delay viene memorizzato in una variabile di minimo.

4. Si entra in un ciclo da cui si esce solo quando il tempo di sistema supera quello di delay del pacchetto. All'interno di questo ciclo si cerca di estrarre nuovi pacchetti dalla coda. Questa volta in modalità non bloccante. Se arriva un nuovo pacchetto e il suo delay è più piccolo di min si aggiorna min con il nuovo valore.
5. Usciti dal ciclo si controlla nella delayed list quali pacchetti sono da inviare al Receiver. Se il delay del pacchetto è stato superato dal tempo di sistema questi vengono spediti.
6. Se la delayed list è vuota il Thread ritorna al punto (3) altrimenti riparte dal punto (4) con un valore di minimo aggiornato rispetto alla lista.

3.1.4 Scheduler

Questo modulo serve per schedulare i pacchetti RTP ai canali descritti precedentemente. Lo Scheduler prende in input un file PCAP. I file PCAP per ogni pacchetto memorizzano un timestamp relativo. Il primo pacchetto parte da 0 secondi relativi e i successivi contengono la differenza di tempo, in valori assoluti, tra il pacchetto di tempo i-esimo e il tempo del primo pacchetto. Premesso questo, si deve trovare una formula che mi permetta di inoltrare questi pacchetti rispettando il timestamp dei singoli pacchetti. Un altro compito dello Scheduler è quello di capire quali sono i pacchetti che deve inoltrare, ossia di capire se l'i-esimo pacchetto preso in esame dal file PCAP è un pacchetto RTP o meno. Per fare ciò si è utilizzato il package Scapy. Questo package contiene una funzione chiamata bind_layers, che permette di decodificare dei pacchetti a un certo protocollo. Prende in input due parametri, entrambi sono dei protocolli, se il pacchetto risulta avere il protocollo indicato come primo parametro allora cerca di decodificare il livello successivo con il protocollo del secondo argomento, per esempio bind_layers(UDP, RTP) decodifica i pacchetti del file PCAP nel protocollo RTP se il livello trasporto usa il protocollo UDP. Quindi l'algoritmo è il seguente:

1. Viene letto il primo pacchetto del file PCAP e memorizza il tempo di arrivo del pacchetto con il timestamp della macchina. Si utilizza una variabile start_time e la si imposta a 0, quest'ultima variabile serve per avere un timestamp di partenza del sistema, verrà inizializzata a dovere nel passo (5).
2. viene eseguito bind_layers(UDP, RTP)

3. Si passa ora a leggere pacchetto per pacchetto in un ciclo while, consapevoli che se il pacchetto utilizza il protocollo UDP sarà decodificato come RTP se lo è.
4. Si controlla se il pacchetto preso in esame utilizza il protocollo IP e RTP e se li utilizza entrambi:
 - (a) Viene controllato se start_time è uguale a 0, se lo è gli viene assegnato il time.time() del sistema, questo per avere un tempo di inizio assoluto quanto più fedele possibile.
 - (b) Si prende il pacchetto e gli si tolgono i primi livelli di rete fino a che non si lascia solo il payload del protocollo UDP.
 - (c) A questo punto si utilizza la funzione delay_calculator() (cfr. 3.1.6) che indica quanto tempo si deve attendere prima di inoltrare un pacchetto ai canali. Questo dato viene ridotto del 15% per evitare ritardi del sistema. Si usa ora una funziona di sleep che risveglia lo scheduler quando è il momento di inoltrare il pacchetto ai canali. La funzione di sleep viene descritta nella sottosezione (cfr. 3.1.6).
 - (d) Dopo che si è atteso si invia il pacchetto ai canali. La comunicazione è spiegata nel seguente paragrafo 3.1.5

3.1.5 Comunicazione Scheduler-Process

Come detto prima all'avvio del sistema Sender si vanno a creare i Python Process dei canali, questi sono dei processi e per questa ragione non si possono usare i meccanismi di comunicazione dei Thread. Fortunatamente Python ha a disposizione le Queue, che implementa una coda. Queue ha i classici metodi di una coda, quali put() che è non bloccante e get() che si blocca se non ci sono elementi attualmente disponibili. Come si può intuire, alla creazione del sender si creano tanti Queue quanti PP. Quando si istanzia un Process questo non inizia la esecuzione in automatico, infatti prima di mandarli in esecuzione è possibile passargli degli argomenti, è in questo momento che gli si passa la sua Queue. Una volta passati tutti gli argomenti di suo interesse il PP si può far partire con il metodo start().

3.1.6 Classe Timing

Questa classe viene usata per calcolare il tempo di attesa che deve intercorrere tra un pacchetto e il successivo. Inizialmente si era pensato di usare soluzioni diverse in base alla piattaforma che si utilizza.

- Windows: Non permette un meccanismo di sleep-wake abbastanza prestante. Infatti sono richieste sleep precise nell'ordine dei nanosecondi, sfortunatamente per restrizioni del sistema Windows si può essere precisi solo ai ms. Quindi in questo caso si utilizza un semplice `while()`.
- BDS\Unix si è pensato di creare un wrapper che permetesse di chiamare le funzioni sleep di posix come `usleep` e `nanosleep`. Sfortunatamente il wrapper è troppo lento ad eseguirle, ciò impedisce al sistema di inviare i pacchetti nelle giuste tempistiche.

La seconda scelta è impraticabile, quindi si è deciso di utilizzare in tutti i sistemi la soluzione adottata su Windows, anche se questa risulta CPU-intensive. Si esce dal `while` quando il tempo di sistema supera il tempo di invio del pacchetto. La funzione che si occupa di questo si chiama `nsleep()` e prende in input il tempo di attesa in secondi.

Questa classe si occupa anche di calcolare quanto bisogna attendere prima di inviare un pacchetto. La funzione che si occupa di questo ha il nome di `delay_calculator()`. Essa prende in input quattro parametri:

- `current_pkt_time` specifica il tempo entro il quale il pacchetto dovrebbe essere spedito secondo il PCAP
- `first_pkt_time` è il tempo del sistema assoluto nel quale è stato ricevuto il primo pacchetto
- `start_calculation_time` è il tempo di sistema nel quale si è iniziato il calcolo del ritardo del primo pacchetto.

Questa classe si occupa del calcolo di due variabili:

- ideale, cioè si suppone che non ci siano ritardi nell'invio dei pacchetti, viene memorizzata quale è la differenza di tempo secondo il timestamp del file PCAP.
- attuale, anche questa è una differenza, ma questa volta si calcola la differenza di tempo effettivamente passata tra l'invio del primo pacchetto e l' i -esimo pacchetto preso in esame secondo il timestamp attuale del sistema.

Le formule che li calcolano sono le seguenti, tenendo in considerazione che `time.time()` restituisce il timestamp attuale del sistema.

Le formule sono le seguenti:

$$ideale = current_pkt_time - first_pkt_time$$

$$attuale = time.time() - start_calculation_time$$

Infine per ottenere la differenza di tempo, in secondi, del tempo da attendere per spedire un pacchetto si deve eseguire la seguente sottrazione:

$$diff = ideale - attuale$$

Diff può essere negativo o positivo, nel primo caso si uscirà immediatamente dalla funzione `nsleep()`, nel secondo caso si controllerà la guardia del `while` finché questa non sarà falsa.

3.2 Receiver

Si analizza la struttura del client osservandone il funzionamento, dal momento in cui si riceve un pacchetto RTP fino al momento in cui viene liberato dalla memoria. Si può successivamente notare come quel pacchetto è stato utilizzato per creare un GOP. Questo si utilizza per raccogliere statistiche sulla qualità del servizio ed è interessante capire come si arriva a determinare una stima di qualità per ogni frame del GOP.

3.2.1 Threads e Process

La comunicazione tra tutti i Threads, che compongono il Receiver, avviene attraverso un meccanismo di produttore-consumatore. I Thread principali sono creati nel Main, mentre alcuni di loro sono creati dai Thread istanziati dal main. Questo è stato fatto per motivi di leggibilità e di gerarchia. Viene ora descritto quali sono i Thread e come vengono utilizzati, per poi spiegare nello specifico come comunicano tra loro.

- `listenerThread`, creato/i nel main, si occupa di ottenere i pacchetti RTP da inoltrare ad altri Threads, si può quindi dire che questo sia il Thread che rappresenta un canale fisico lato receiver.

- GOPThread, creato dal listenerThread, ha l'obiettivo principale di analizzare un pacchetto RTP, per esempio si occupa di ottenere il NAL unit, per capire come andrà decodificato il pacchetto. È utilizzato per immagazzinare dati temporanei che serviranno per calcolare le statistiche in statThread. Questi dati sono grezzi infatti momentaneamente non vengono effettuati calcoli. Il GOPThread inserisce in una struttura dati i pacchetti ricevuti, ordinati secondo il numero di sequenza del pacchetto RTP, per il ReaderPacket. Il suo ultimo compito è quello di inoltrare i pacchetti opportuni per la creazione di un video finale.
- ReaderPacket, viene creato dal main, ha il compito di salvare un GOP sul disco. Inizia il suo lavoro da un pacchetto i-esimo, che rappresenta l'inizio di un GOP, fino all'ipotetico pacchetto finale di quest'ultimo. Termina la sua operazione con il salvataggio del file sul disco.
- DecoderThread, creato/i dal Thread ReaderPacket, si occupa di aprire il GOP salvato dal ReaderPacket e di decodificarlo in modo da ottenere i frame che lo compongono per effettuare analisi sulla qualità dell'immagine.
- statThread, creato dal main, si occupa di analizzare le informazioni raccolte dal GOPThread per poi inviare queste informazioni sul canale di feedback.
- Segnali, è istanziato dal main, resta in attesa di un segnale per la chiusura del client. Quando arriva si occupa di chiudere tutti gli altri Thread.

Per quanto riguarda il Process questo viene creato con una popen, che istanzia uno script python di nome plot.py. Si occupa di creare un grafico e calcolare il PSNR. Chiameremo per semplicità questo processo *PlottingProc*

3.2.2 Comunicazioni

Come detto si utilizza un meccanismo di produttore-consumatore ma è opportuno specificare chi comunica con chi e se necessario come.

1. listenerThread comunica con GOPThread attraverso una lista. Vengono inseriti i pacchetti RTP attraverso una struttura dati che può contenere anche altri dati supplementari chiamata el_rtp.
2. GOPThread comunica con ReaderPacket. Si utilizza una lista. Nella lista viene inserita una struttura che specifica il numero di sequenza iniziale

di un GOP e quello finale, insieme ad altri dati accessori, nella struttura detonimata el_gop. GOPThread si interfaccia a Gstreamer inoltrandogli pacchetti che consentono di creare il video desiderato. Comunica indirettamente con ReaderPacket attraverso una tabella hash condivisa e con il Thread delle statistiche.

3. ReaderPacket comunica con DecoderThread con produttore-consumatore. Viene inoltrato l'el_gop passato da GOPThread. Con el_gop si riesce a risalire al path del file creato da ReaderPacket.
4. DecoderThread comunica con PlottingProc utilizzando una pipe. Viene passato tutto come stringa nel stdin di PlottingProc. È possibile comprendere quando si spediscono tutti i dati relativi a un determinato frame perché questo termina con \n.
5. Il Thread dei segnali comunica indirettamente con tutti gli altri Thread. Quando si riceve un segnale di chiusura il Thread imposta una variabile per indicare la chiusura del Receiver. Segnali invia un segnale al statThread per permettergli di terminare il prima possibile, e successivamente invia un SIGALARM a ogni listenerThread. Questo si fa perché i listenerThread sono bloccati su una system call. Quando viene ricevuto il segnale si sbloccano e inizia una reazione a catena nella quale ogni listenerThread inserirà un pacchetto di chiusura a un GOPThread. Esso leggendolo invia un el_gop di chiusura a ReadPacket che inoltrerà l'elemento ai DecoderThread.

Lo schema seguente riassume le comunicazioni tra Thread:

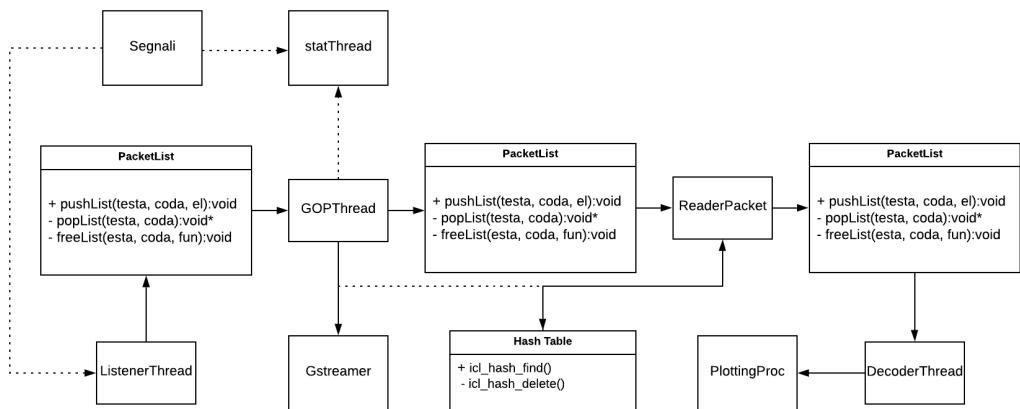


Figura 3.2: Le linee continue indicano una comunicazione diretta tramite liste, le altre una comunicazione indiretta con strutture dati condivise.

3.2.3 listenerThread

Alla sua creazione gli viene assegnato un intero che identifica il suo canale e dunque su che porta dovrà ascoltare per acquisire nuovi pacchetti. Prima di attendere per nuovi pacchetti si crea il GOPThread. Questo potrà essere terminato solo dal listenerThread che lo ha creato. Alla ricezione di ogni pacchetto si alloca spazio per memorizzarlo in el_rtp in cui successivamente si andranno a memorizzare dati riguardanti il suo NALu e il timestamp di arrivo. Questi elementi vengono inseriti nella lista che comunica col GOPThread.

3.2.4 GOPThread

Ha il compito principale di dissezionare il pacchetto RTP in modo che risulti semplice per ReaderPacket creare il GOP di appartenenza del pacchetto. Per farlo si inizializza la struttura dati detta el_gop. Questa contiene dati come il primo numero di sequenza di un GOP, il suo numero di sequenza finale e numero di pacchetti persi di un GOP. Viene quindi utilizzata come struttura di supporto per capire su quale GOP si lavora e viene passata a ReaderPacket per fargli capire da che numero di sequenza deve iniziare a lavorare. Riesce a fare ciò perché riceve gli el_rtp che contengono i pacchetti da analizzare da listenerThread. Mentre questo Thread lavora per raccogliere questi dati fa anche le seguenti operazioni:

- Vengono inseriti in una struttura delle statistiche dei dati relativi al pacchetto, come ad esempio su che canale è stato ricevuto il pacchetto, il numero di sequenza e il timestamp di ricezione e invio.
- Ad ogni pacchetto in arrivo si cerca di inserire in una tabella Hash il nuovo elemento. Se l'elemento è stato inserito significa che durante l'esecuzione non ci sono stati pacchetti con lo stesso numero di sequenza.
- Se è stato inserito nella tabella hash viene creato un nuovo pacchetto di rete da spedire al modulo di Gstreamer, che si occuperà di visualizzare il video.

Inoltre, è opportuno chiarire come si viene a conoscenza dell'inizio e della fine di un GOP e dell'algoritmo che gestisce la concorrenza tra tutti i GOPThread. In questo caso si usa la mutua esclusione sia per le statistiche che per l'inserimento di una tabella hash, ma anche per il completamento dei dati della struttura el_gop. La maggior parte di queste informazioni sono contenute nella

funzione workOnPacket() dopo l'inserimento nella tabella Hash e in addPacketToGOP(). Nella prima si va a guardare il FU indicator, in particolare si memorizza il valore di Type, nel campo denominato slice_type di el_gop. Se si tratta di un pacchetto di tipo sps o pps, ossia di pacchetti utilizzati come trailer di un GOP per la sua decodifica, si memorizza rispettivamente il valore 7 o 8. La presenza di questo Type indica l'inizio di un nuovo GOP, quindi si può utilizzare questa informazione per iniziare a inviare el_gop a ReaderPacket. Se il campo Type contiene il valore 28, si tratta di un FU-A, non si memorizza ancora il valore ma si va a guardare il campo Type del FU header. Ciò è elaborato da addPacketToGOP(), se il Type di FU indicator è 28 viene memorizzato il valore Type del FU header in slice_type, come descritto precedentemente, questo campo nel nostro caso preso in esame contiene il tipo del frame, quindi può rappresentare un frame I o P. Nell'ultima funzione citata si va ad usare un campo di el_gop chiamato gop_over, essa è una variabile booleana che indica con -1 che l'istanza di el_gop è ancora quella attuale di tutti i Thread che analizzano il GOP, viceversa qualche Thread ha ricevuto almeno un pacchetto di un nuovo GOP e quindi si stanno facendo modifiche su un GOP vecchio. Sempre in questa funzione si vanno a riempire vari campi di el_rtp del pacchetto RTP, come il campo state che indica se si tratta di un pacchetto di inizio frame o uno di fine, oppure il campo decoder che è utilizzato all'inizio di un frame per tutta la sua decodifica. Si tratta di otto bit ma sono i più importanti da ottenere. Per ottenere questo byte si deve effettuare una semplice operazione bit a bit. Si usa il Bitwise OR tra i primi 3 bit del FU indicator e i 5 bit del campo Type del FU header. In ReaderPacket viene descritto come usare questo valore salvato nel campo decoder. Per concludere un pacchetto in addPacketToGOP() viene analizzato come segue se si tratta di un I frame:

1. Se gop_over è uguale a -1:
 - (a) Si controlla se il numero di sequenza del pacchetto appena ricevuto è minore dell'inizio del numero di sequenza di inizio GOP ed è maggiore della fine dell'ultimo pacchetto del vecchio GOP, se è True si aggiorna il valore del primo numero di sequenza del GOP.
 - (b) Altrimenti si controlla se il numero di sequenza appena ricevuto è maggiore del primo pacchetto P ricevuto e si controlla anche se si è ricevuto un pacchetto di tipo P del GOP corrente. Questi due controlli permettono di stabilire che il pacchetto ricevuto è di

un nuovo GOP, quindi viene memorizzato su gop_over il numero di sequenza con il valore maggiore ricevuto dal GOP. Si procede perciò a sostituire l'istanza di el_gop con una nuova. La nuova istanza non verrà usata immediatamente, questo è il motivo per cui si usa gop_over.

2. altrimenti si fa lo stesso controllo di (a), cambia (b) perchè ora si deve aggiornare l'inizio del nuovo GOP.

Se si tratta di un P frame:

1. Se non c'è ancora stato un pacchetto RTP di tipo P e in particolare è un pacchetto P del GOP corrente, allora si segnala che nel el_gop corrente è stato trovato un pacchetto P e si memorizza un numero di sequenza appartenente a questa tipologia di pacchetti.
2. Poiché si cerca un numero di sequenza il più vicino possibile alla fine del GOP questo aumenta quando gop_over è uguale a -1 e il numero di sequenza finale di el_gop è minore del numero di sequenza del pacchetto si aggiorna questo valore.

3.2.5 ReaderPacket

Preleva dalla coda che comunica con GOPThread un istanza di el_gop. Una volta ricevuto attende 200 ms nel caso ci siano pacchetti che arrivano in ritardo. Una volta scaduti imposta un valore che rappresenta un numero di sequenza. Se nel Receiver arriva un pacchetto che supera questo valore, questo viene immediatamente scartato. Bisogna ora andare a salvare il GOP nel disco. Per fare questo ci serviamo di tre while che fanno sostanzialmente la stessa cosa. Ognuno si occupa di un momento diverso del GOP. È possibile distinguere tra le tre fasi grazie ai dati raccolti in el_rtp, questi tre momenti sono:

1. I primi pacchetti sono di sps e pps, saranno indicati da uno slice_type != 5.
2. I pacchetti successivi sono gli I-frame quando lo slice_type è uguale a 5.
3. Gli ultimi sono P-frame e quindi hanno uno slice_type < 5.

I pacchetti di inizio gop cioè sps e pps sono teoricamente univoci per tutta la trasmissione del flusso RTP quindi basta ottenerli una volta. Per sicurezza

ho strutturato l'algoritmo in modo tale che ogni volta che si presentano si memorizzano e utilizzo i nuovi dati.

3.2.6 Ottenere i pacchetti

In GOPThread vengono salvati i pacchetti in una tabella Hash il loro numero di sequenza è la chiave, mentre la struttura el_rtp contenente il pacchetto che utilizza quel numero di sequenza è il suo valore. Siccome l'istanza di el_gop contiene sia l'inizio che la fine in numero di sequenza di un GOP, è facile ottenere tutti i pacchetti, uno per uno, con un numero di sequenza sempre crescente.

3.2.6.1 Algoritmo per il salvataggio del GOP

L'algoritmo è descritto nella funzione saveGOP(). Si parte prendendo da el_gop dati relativi all'identificazione del GOP, ad esempio il numero totale di GOP attualmente creati. Si crea quindi un file nel disco che lo identifica. Viene estratto ora il primo pacchetto dalla tabella Hash, in base al numero di inizio sequenza di el_gop. La prima fase ha la seguente procedura. Finché lo slice_type è diverso da 5 si tiene in considerazione il pacchetto. Si scrive in un buffer il suo contenuto eliminando il timestamp presente alla fine del pacchetto, e si incrementa una variabile contatore. Ogni nuovo pacchetto sarà concatenato con quelli passati. Finito questo primo ciclo viene controllato che il contatore abbia come valore due. Se la condizione risulta vera si sostituiscono i vecchi valori di sps e pps, altrimenti si continuano a usare i vecchi. Viene salvato sul file il buffer che contiene i dati del buffer con i valori di sps e pps. Si prende ora un nuovo pacchetto dalla tabella Hash. A questo punto secondo la richiesta del cliente viene memorizzato il numero di sequenza dei pacchetti che non sono arrivati a destinazione in un array presente in el_gop. Questa funzione viene chiamata ogni volte che si ottiene un nuovo pacchetto dalla tabella Hash. Si passa ora alla seconda fase:

1. Finchè slice_type == 5:
2. Si controlla se il pacchetto è il primo pacchetto del frame. Se lo è, si salva il contenuto del decoder di el_rtp su un buffer.
3. Nel buffer viene memorizzato il payload del pacchetto RTP nel buffer usato nel punto 2, si escludono però i primi 2 byte di header payload e il timestamp aggiunto lato sender.

4. si salva il buffer nel file del GOP in append ai dati precedenti.
5. Viene preso il nuovo pacchetto, si salvano nuovamente i numeri di sequenza non utilizzati nell'array di el_gop. Infine si ritorna al punto 1.

Si usa lo stesso algoritmo per la terza fase. Non si è ancora descritto come si ricerca un nuovo pacchetto dalla tabella Hash.

1. Viene controllato che il pacchetto attuale è stato spedito.
2. Se ciò non è avvenuto si ritorna al primo punto.
3. Se è stato spedito si entra in un altro while, si esce solo quando si è trovato un nuovo numero di sequenza oppure si è superato l'ultimo numero di sequenza del GOP.

Quando ReaderPacket finisce di salvare il GOP invia el_gop a DecoderThread tramite una lista.

3.2.7 DecoderThread

Si estre el_gop dalla lista che comunica con ReaderPacket e DecoderThread. Viene utilizzato per aprire il file che rappresenta il GOP sul disco. Questo Thread utilizza la libreria libav di Gstreamer per la decodifica dei GOP. Come è facile intuire ogni frame, a parte il prima, dipende dai precedenti, quindi non è possibile rendere la decodifica di un singolo GOP multithread. Le operazioni da eseguire per decodificare i frame sono le seguenti:

1. Aprire il file; la libreria libav viene incontro alle nostre esigenze, avendo una funzione in grado di parsare il file e di creare dei dati in grado di capire il tipo di video.
2. Si entra a questo punto in un while. Si esce dal while solo quando sono stati analizzati tutti i frame del GOP.
 - (a) I-esimo frame viene decodificato nella funzione decode_packet().
Questa restituisce un frame, con la sua bitmap.
 - (b) Il profilo colori attualmente in uso è yuv, questo consente di visualizzare l'immagine solo in bianco e nero. Per ottenere i colori si converte il profilo colori in RGB24. Operazione effettuata nella funzione pixel_to_rgb24().

- (c) Viene chiamata la funzione decode_to_png() dove si converte l'immagine da bmp a png con le apposite funzioni di libav.
- (d) Viene salvato il png.
- (e) Si invia a PlotProc, tramite stdin, il nome del frame appena creato sul disco e il nome del file campione corrispondente. Prima di terminare il messaggio con un newline si invia il numero di frame e quali pacchetti sono stati persi.

Si è utilizzato il caso in cui tutte le funzioni funzionano correttamente, in caso di fallimento si ritorna al punto due. Tutte queste funzioni sono descritte nel file h264topng.h

3.2.8 PlotProc

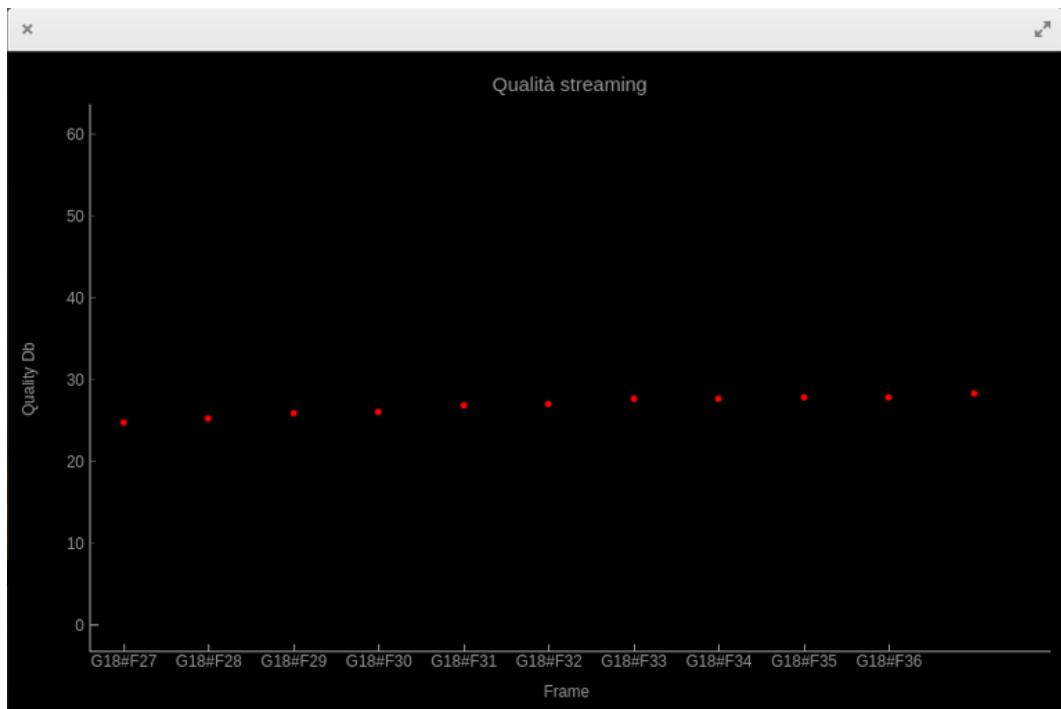


Figura 3.3: Grafico che mostra la qualità dell'immagine.

Questo modulo è scritto interamente in python, utilizza le dipendenze cv2 e pyqtgraph. I dati relativi a nuovi frame arrivano dal DecoderThread. Viene istanziato attraverso una popen di C per questo motivo è possibile comunicare scrivendo nel suo stdin. Tutti i dati relativi a un frame terminano con l'arrivo di un newline. Se ci sono stati errori nella decodifica di un frame arriva un path identico sia per l'immagine decodificata che per l'immagine campione. In

questo caso si calcola il PSNR con l'ultima immagine disponibile. Se i nomi sono diversi viene calcolato il PSNR con le immagini dei path inviati. I dati calcolati vengono salvati su un file csv. Questo file, oltre che hai dati sul PSNR-frame conterrà anche i dati sui pacchetti che non sono stati ricevuti. Questo perché si cerca una correlazione tra qualità dell'immagine e quali e quanti pacchetti non sono stati ricevuti. le ordinate rappresenta il PSNR in dB. Nel grafico le ascisse indicano un numero di GOP con la lettera G seguita dal numero che lo identifica e con #F si intendo il numero di frame del GOP. Il puntino rosso indica il rumore del frame in dB.

3.2.9 statThread

Quando viene istanziato crea una struttura di copia di stat_t. Questa struttura viene usata da GOPThread per memorizzare dati base per il calcolo delle statistiche. Viene chiamata finestra una certa quantità di tempo. Questa finestra viene utilizzato come tempo per raccogliere i dati e inviarli al canale di feedback del Sender. Al termine della finestra si scambia la struttura con i dati raccolti con l'istanza della copia di stat_t, in questo caso l'oggetto utilizzato per raccogliere i dati diventa la nuova copia. Quindi al termine della finestra queste due istanze vengono scambia ogni volta. Effettuato lo scambio si eseguono le seguenti operazioni:

1. Viene controllato se nel canale i-esimo è stato ricevuto qualche pacchetto. Se non si è ricevuto vengono inviati nel canale di feedback dati che permettono al sender di capire l'evento.
2. Altrimenti, si copia l'array contenente i numeri di sequenza che sono arrivati nell'arco della finestra.
3. Viene calcolato il delay tra un pacchetto e l'altro della finestra.
4. Viene ordinato l'array dei numeri di sequenza, è il motivo per cui è stata effettuata una copia. In questo modo si mantiene sia l'ordine di arrivo che l'ordine crescente dei numeri di sequenza.
5. Viene calcolato il jitter, out-of-order, lunghezza media dei pacchetti persi e il numero di pacchetti persi all'interno di una finestra.
6. Queste statistiche vengono raccolte nella struttura send_stat e inviate ai canali di feedback

Rimane da chiarire come vengono calcolate queste statistiche.

3.2.9.1 Ritardo

Alla ricezione di un pacchetto il sistema operativo fornisce il timestamp di ricezione. Questo viene salvato in un campo di el_rtp. Quando un pacchetto viene inviato dal Sender viene aggiunto alla fine del payload un timestamp di invio.

Quando il GOPThread memorizza le statistiche relative a un pacchetto si occupa di salvare in una struttura detta pkt_info, numero di sequenza, timestamp di ricezione e timestamp di invio. Il Thread delle statistiche calcola il ritardo di un canale nel seguente modo:

$$delay = \frac{\sum(received_timestamp - sended_timestamp)}{\#pacchetti}$$

Dove received_timestamp e sended_timestamp rappresentano i timestamp dell'i-esimo pacchetto.

3.2.9.2 Variabilità del ritardo (jitter)

Per calcolare il jitter è necessario ordinare i pacchetti della finestra in base al loro numeri di sequenza. Si prende anche in questo caso il timestamp di arrivo e di invio. Prima di calcolare il jitter bisogna calcolare la differenza del tempo di transito di due pacchetti:

$$D(i, j) = (R_j - S_j) - (R_i - S_i)$$

Il jitter calcolato sull'i-esimo pacchetto è definito come segue:

$$J(i) = J(i-1) + \frac{(|D(i-1, i)| - J(i-1))}{16}$$

3.2.9.3 Pacchetti fuori ordine

Per questo calcolo sono interessanti solo i numeri di sequenza. Servono due array che contengono gli stessi numeri di sequenza, uno ordinato in modo crescente e l'altro in base all'arrivo nel sistema. Per calcolarlo si controlla che nell'i-esima posizione i valori dei due array coincidono. Se non coincidono si cerca nell'array non ordinato il valore della array ordinato in posizione i-esima. Si fa quindi uno shift di valori nell'array non ordinato fino a che non si trova l'elemento coincidente. Quindi si sposta l'elemento nella posizione j che coincide con i-esimo elemento dell'array ordinato nell'array non ordinato all'i-esima posizione. Si incrementa di uno l'out of order e si continua a fare

questi controlli fino a che non sono stati visitati tutti gli elementi dell'array ordinato.

3.2.9.4 Lunghezza media della perdita

Si prende l'array coi numeri di sequenza ordinati. Per capire se ci sono pacchetti che non sono arrivati si calcola la differenza con il numero di sequenza in posizione i-esima con il numero di sequenza in posizione i-1 e si sottrae uno da questa differenza. Chiamiamo questa differenza num_lost. Se num_lost è maggiore di zero si incrementa la variabile che rappresenta il numero di pacchetti persi con num_lost, viene incrementata anche la variabile che rappresenta il numero di buchi dei pacchetti. Il calcolo è il seguente:

$$\frac{\sum \text{num_lost}}{\#\text{buchi}}$$

3.2.9.5 Numero di pacchetti persi

Il calcolo di questo è molto semplice. Si prendo il numero di pacchetti della finestra meno uno, chiamiamo questo valore size. Si calcola ora la sottrazione tra il numero di sequenza maggiore meno il numero di sequenza più piccolo, e si chiama questo risultato length. Il numero di pacchetti è dato dal valore assoluto di size - length se size è maggiore di length, altrimenti si invertono sottraendo e diminuendo.

3.2.10 Segnali

Questo thread è pensato per far decidere all'utente quando interrompere la simulazione. Attende i seguenti segnali: SIGUSR1, SIGINT, SIGTERM e SIGQUIT. Se si riceve uno di questi, si inizia la procedura di chiusura del receiver.

- Si invia un SIGALARM a statThread per farlo uscire il più velocemente possibile dalla sleep.
- Viene impostata una variabile di uscita, ogni Thread potrà leggere questo valore e capire che non si devono più ricevere pacchetti o nuovi elementi.
- Per ogni listenerThread si invia un SIGALARM per uscire dalla chiamata bloccante.

- viene svuotata la lista che comunica tra listenerThread e GOPThread e viene aggiunto un pacchetto che serve per sbloccare GOPThread e permettere una terminazione graceful del Thread.
- Si ripete l'operazione (4) per la lista tra GOPThread e ReaderPacket e per la lista tra ReaderPacket e DecoderThread.

Quando arriva il segnale di chiusura ad ogni Thread questo si occupa di liberare la memoria che occupano e di ripetere il quarto punto per la loro lista associata. Quando tutti i Thread sono chiusi il main scrive nel stdin di PlotProc un messaggio di chiusura, questo viene inviato e si può procedere chiudendo la pipe di PlotProc. Terminando così l'esecuzione del Receiver.

3.2.11 Gstreamer

Questo modulo viene avviato da uno script bash, questo stesso script avvia anche il client. Viene avviato impostando il tipo di traffico che si dovrà andare a gestire e su quale porta deve ascoltare per ricevere i pacchetti che contengono il video da visualizzare.

Capitolo 4

Manuale utente e sviluppatore

4.1 Requisiti e dipendenze

Il Sender e il Receiver sono stati scritti in maniera molto diversa tra loro, infatti il Sender è scritto in python mentre il receiver è scritto in linguaggio C con l'eccezione che utilizza uno script in Python. La differenze tra i due rende diversa l'installazione delle dipendenze.

4.1.1 Dipendenze di sistema

Per poter funzionare correttamente si richiede che nel sistema operativo siano installati, oltre che i normali tool per poter compilare programmi scritti in C e i build-essential, i seguenti programmi:

1. Python¹[3.7]
2. Gstreamer²[1.16], da installarsi coi libav-tool. Si consiglia l'installazione con il seguente comando:

```
sudo apt-get install libgstreamer1.0-0
gstreamer1.0-plugins-base gstreamer1.0-
plugins-good gstreamer1.0-plugins-bad
gstreamer1.0-plugins-ugly gstreamer1.0-libav
gstreamer1.0-doc gstreamer1.0-tools
gstreamer1.0-x gstreamer1.0-alsa gstreamer1
.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5
gstreamer1.0-pulseaudio
```

¹<https://www.python.org/>

²<https://gstreamer.freedesktop.org/>

3. Libpcap³[1.8.1], installabile dal terminale con il comando:

```
sudo apt-get install libpcap-dev
```

. Viene utilizzato dal Receiver per ottenere il timestamp e sniffare i pacchetti inviati dal Sender.

4. FFmpeg⁴[4.2.2] Viene utilizzato per la decodifica e la codifica da GOP a PNG, soprattutto grazie alla sua libreria interna libav-tool. Si consiglia anche in questo caso l'installazione tramite linea di comando:

```
sudo apt-get update -qq && sudo apt-get -y
    install autoconf automake yasm nasm cmake git
    -core libass-dev libfreetype6-dev libgnutls28
    -dev libsd12-dev libtool libva-dev libvdpau-
    dev libvorbis-dev libxcb1-dev libxcb-shm0-dev
    libxcb-xfixes0-dev pkg-config texinfo wget
    libx264-dev libvpx-dev zlib1g-dev
    wget https://ffmpeg.org/releases/ffmpeg-4.2.2.
    tar.bz2
    tar -xf ffmpeg-4.2.2.tar.bz2
    rm ffmpeg-4.2.2.tar.bz2
    cd ffmpeg-4.2.2
    ./configure --enable-gpl --enable-gnutls --
    enable-libass --enable-libfreetype --enable-
    libvorbis --enable-libvpx --enable-libx264 --
    enable-shared --enable-nonfree --enable-
    pthreads
    make
    sudo make install
    sudo /sbin/ldconfig
    cd ..
    rm -rf ffmpeg-4.2.2
```

Attenzione questa installazione potrebbe richiedere qualche minuto, ma è essenziale.

³<https://www.tcpdump.org/>

⁴<https://ffmpeg.org/>

4.1.2 Dipendenze del sender

1. Scapy⁵ 2.4.3, Utilizzato per leggere PCAP file e gestire i pacchetti di rete.
2. ctypes⁶[15.17] Di solito preinstallato con Python. Viene utilizzato dallo scheduler per utilizzare le sleep del C, in quanto python non implementa una sleep abbastanza reattiva (tuttavia python è troppo lento nell'eseguire il wrapping delle sleep, quindi si usano altri modi per ottenere il risultato voluto). È anche utilizzato per ricevere le statistiche dal canale di feedback.
3. SciPy⁷[1.5.1] Utilizzato per calcolare le distribuzioni gamma.

4.1.3 Dipendenze del client

La parte in python si occupa di visualizzare il grafico del PSNR; le sue dipendenza sono:

1. pyqtgraph⁸[0.11.0] Si occupa di creare il grafico e di visualizzare i dati.
2. PyQt5⁹[5.15.0] Viene utilizzato da pyqtgraph per gestire l'interfaccia grafica del plot. Si consiglia l'installazione su piattaforme Unix con il seguente comando:

```
sudo apt-get install python3-pyqt5
```

3. opencv-python¹⁰[4.3.0.36] Utilizzato per calcolare il PSNR (cioè il calcolo della qualità dell'immagine).

⁵<https://scapy.net/>

⁶<https://docs.python.org/2/library/ctypes.html>

⁷<https://www.scipy.org/>

⁸<http://pyqtgraph.org/>

⁹<https://www.qt.io/>

¹⁰<https://opencv.org/>

4.2 Primo utilizzo

Alla prima apertura il sistema si configura nel seguente modo:

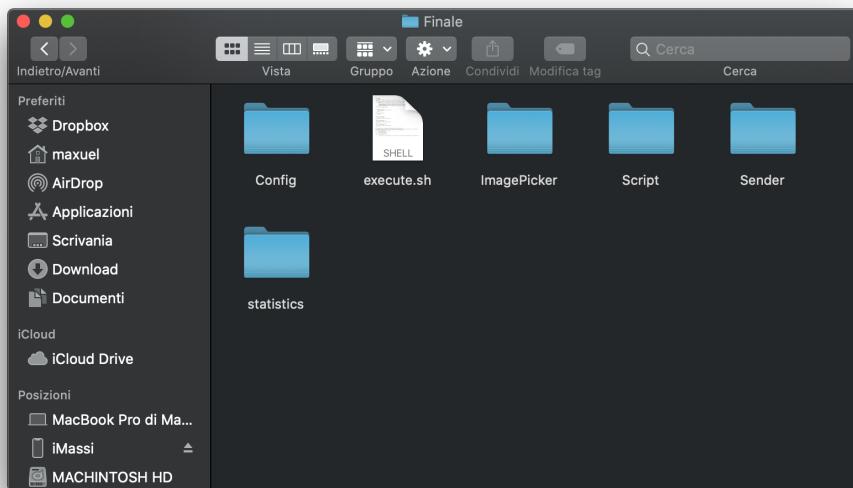


Figura 4.1: Cartelle del sistema.

4.3 Config

Questa cartella contiene dei file di configurazione di esempio. I file di configurazione al loro interno contengono gli argomenti da utilizzare per il corretto avvio del Sender e del Receiver. Ogni parametro dei file di configurazione deve essere del seguente formato `parametro = valore`; è importante rispettare gli spazi. L'unica eccezione è il parametro `channels.stats` in cui il campo valore richiede una formattazione particolare.

4.3.1 File di configurazione del Sender

I parametri richiesti sono i seguenti, nessuno è opzionale!

- **number_of_links**: rappresenta il numero di canali da utilizzare per spedire i pacchetti, si consiglia di utilizzare un valore tra 1 e 3.
- **port**: numero di porta in cui si spedisce un pacchetto. Questo valore sarà utilizzato come base per il valore delle altre porte. Ad esempio se si utilizza il valore tre in `number_of_links` e il valore 5000 in `port`, si andranno ad utilizzare le porte da 5000 a 5002.

- **receiver_IP**: si inserisce l'indirizzo IP del Receiver per spedirgli i pacchetti.
- **incoming_feedback_port**: porta in cui si ricevono le statistiche raccolte dal Receiver nella finestra.
- **simulate**: viene utilizzato per sapere se si vuole simulare un contesto di rete reale oppure no. Si accettano 2 valori, 1 e 0. Se vale 1 allora si simula un contesto reale e si dovrà utilizzare il parametro sottostante, altrimenti non si tiene conto del parametro channels_stats.
- **channels_stats**: il campo valore deve essere formattato nel seguente modo: [ch1_gammas_distr ; ch2_gammas_distr ; ... ; chN_gammas_distr]. ch_gammas_distr rappresenta una sestupla in cui sono presenti i seguenti valori delle distribuzioni gamma nel seguente formato [alpha_e scale_e alpha_p scale_p alpha_d scale_d] dove alpha_e, scale_e rappresentano i valori della distribuzione per entrare in un evento perdita, alpha_p, scale_p per creare la distribuzione che indica quanti pacchetti perdere e alpha_d, scale_d i valori per il delay dei pacchetti.
- **pcap_src_file**: deve contenere come valore il path assoluto in cui è presente un PCAP con pacchetti RTP.
- **GOP_folder**: il suo valore deve essere un path assoluto di una cartella in cui si salvano i GOP lato Sender.
- **VF_folder**: si richiede il path assoluto di una cartella in cui salvare i frame del GOP.

4.3.2 File di configurazione del Receiver

number_of_links: rappresenta il numero di canali da utilizzare per ricevere i pacchetti. È opportuno utilizzare lo stesso valore che si è utilizzato per il Sender.

port: numero di porta in cui si riceve un pacchetto. Questo valore sarà utilizzato come base per il valore delle altre porte. Ad esempio se si utilizza il valore tre in number_of_links e il valore 5000 in port, si andranno ad utilizzare le porte da 5000 a 5002.

sender_IP: indirizzo IP del sender. Viene utilizzato sia per capire quali pacchetti ricevere, sia per inviare al canale di feedback le statistiche della finestra.

feedback_port_at_sender: porta in cui inviare le statistiche al Sender.

nic: interfaccia di rete per ascoltare i pacchetti. Si può ottenere la lista delle interfacce utilizzando ifconfig nel terminale. Se si inserisce un nic sbagliato il Receiver non sarà in grado di ottenere i pacchetti RTP.

window_length: lunghezza della finestra in microsecondi. Si consiglia di avere finestre lunghe 200ms. Si evitino finestre troppo lunghe, perchè in questo caso non si è in grado di rispondere abbastanza velocemente ai cambi repentinii della rete, oppure troppo corte, in modo da non ottenere statistiche che non raccolgono abbastanza informazioni.

video_port: porta in cui verranno inoltrati i pacchetti per Gstreamer per la visualizzazione finale del video.

GOP_folder: il suo valore deve essere un path assoluto di una cartella in cui si salvano i GOP lato Receiver.

VF_folder: si richiede il path assoluto di una cartella in cui salvare i frame del GOP.

original_VF_folder: deve contenere il path assoluto in cui sono presenti dei frame campione con cui confrontare le immagini presenti in VF_folder.

decoding_threads: numero di Thread da utilizzare per decodificare il GOP. Se non si dispone di un computer abbastanza potente si consiglia di utilizzarne uno. Anche avendo un computer che riesce a gestirne più di un Thread, si consiglia di non eccedere e di usarne al massimo 3 perché saranno molto rare le occasioni in cui si riusciranno a sfruttare tutti assieme.

4.4 execute.sh

È uno script bash che permette l'avvio del Receiver, Sender o entrambi. Prima di poterlo utilizzare è necessario usare il comando chmod +x per dare i permessi di esecuzione allo script. Execute.sh dovrà essere eseguito con i permessi di amministratore e si dovrà specificare il path del file di configurazione del Sender e del Receiver, anche nell'eventualità in cui non si dovranno utilizzare entrambi. Lo script prende come argomento un intero e i due path. L'intero serve per distinguere quale dei due file di configurazione utilizzare. Si utilizza il valore 1 per indicare che si vuole avviare il Sender e che si utilizzerà il primo path, con 2 si esprime la volontà di voler avviare il Receiver e in questo caso si utilizza il secondo path, mentre se si utilizza 0 verranno avviati entrambi rispettando l'ordine precedente per i path. Si consiglia di interagire solo con

questo script. Infatti se execute.sh dovrà eseguire qualche altro componente farà tutti i controlli necessari per permetterne l'esecuzione in autonomia. In generale il sistema è pensato per essere avviato e utilizzato da questo script.

4.5 ImagePicker

Al suo interno sono contenuti tutti i file relativi al codice del Receiver. Non se ne consiglia l'apertura a un utente non esperto. Tra i file di maggiore interesse si può trovare lo script start.sh, si occupa di avviare gstreamer per la visualizzazione del video e l'avvio del Receiver. Il file plot.py è utilizzato per la visualizzazione del grafico e per salvare le statistiche del grafico, da visualizzare solo se si vuole modificare la cartella in cui si salvano le statistiche. Operazione che è possibile effettuare modificando la variabile path presente tra le prime righe del codice.

4.6 Script

Questa cartella contiene gli script per la lettura dei file di configurazione. Ognuno degli script contenuti in questa cartella effettua semplici controlli, per esempio controlla che ogni parametro di input sia non vuoto, senza verificare che abbiano un significato semantico corretto. Su argomenti specifici come channels.stats viene effettuato un controllo più specifico, verificando il tipo di dato inserito. Gli script della cartella, infine, avviano il sistema utilizzando i dati presenti nei file di configurazione.

4.7 Sender

Contiene gli script Python che si occupano del Sender. Se si desidera modificare il path delle statistiche è necessario modificare la variabile path nel file Statistiche.py.

4.8 statistic

Al suo interno sono presenti altre due cartelle: plot e stat. Nel primo vengono memorizzate le statistiche create dal Receiver, nel secondo quelle create dal Sender. Dentro queste due cartelle, se il sistema è già stato eseguito, si troveranno dei file .csv che contengono le statistiche. Questa cartella non viene

ricreata in automatico dal sistema, ed è fortemente consigliato non eliminarla. Se si vuole cambiare la destinazione delle statistiche è consigliato seguire le istruzioni della sezione ImagePicker e Sender.

4.9 Come utilizzare il sistema

La prima volta, quando ci si trova di fronte alla schermata della figura (4.1) bisogna andare a modificare i file di configurazione in modo che si usino dei valori adatti per il nostro sistema. Si vada nella cartella di Config e si apra il file sender.conf. Il valore da modificare ai fini della corretta esecuzione del Sender è pcap_src_file. Gli altri parametri non sono dipendenti dal sistema d'uso, si consiglia di modificarli per adattarli alle proprie esigenze. Si apra ora receiver.conf e si modifichi il parametro nic. Si possono ottenere i nomi utili usando il comando *ipconfig*. Se si utilizza l'ip locale del computer si usi il nome dell'interfaccia di loopback, altrimenti si utilizzi il nome della scheda di rete. Gli altri parametri sono a discrezione dell'utente.

Si torni ora alla directory precedente e si apra il terminale in questa directory. Bisogna abilitare lo script execute.sh e, come descritto precedentemente, questo si può ottenere eseguendo il comando *chmod +x execute.sh*. Si aprano ora due finestre del terminale; nella prima si avvia il Sender con il comando:

```
./execute.sh 1 Config/sender.conf Config/receiver.conf
```

Nella seconda si avvia il Receiver scrivendo nel terminale:

```
./execute.sh 2 Config/sender.conf Config/receiver.conf
```

Ovviamente se e solo se si usano i due file di configurazione forniti dal sistema, altrimenti si dovranno usare i file creati. Per interrompere l'esecuzione si può inviare un SIGINT al Sender per chiudere anche il Receiver. Se si invia questo segnale solo al Receiver verrà chiuso solo quest'ultimo. Alla chiusura del sistema (anche durante l'esecuzione se necessario), si può andare a visionare la cartella dove sono state salvate le immagini. In genere si occupa molto spazio molto velocemente. All'esecuzione successiva del sistema i file saranno sovrascritti. Quindi si consiglia di effettuare una copia delle cartelle interessate o di cambiare la destinazione nel file di configurazione se si vogliono analizzare successivamente i frame.

La cartella, che contiene le immagini decodificate durante l'esecuzione del sistema, sarà composta nel seguente modo:

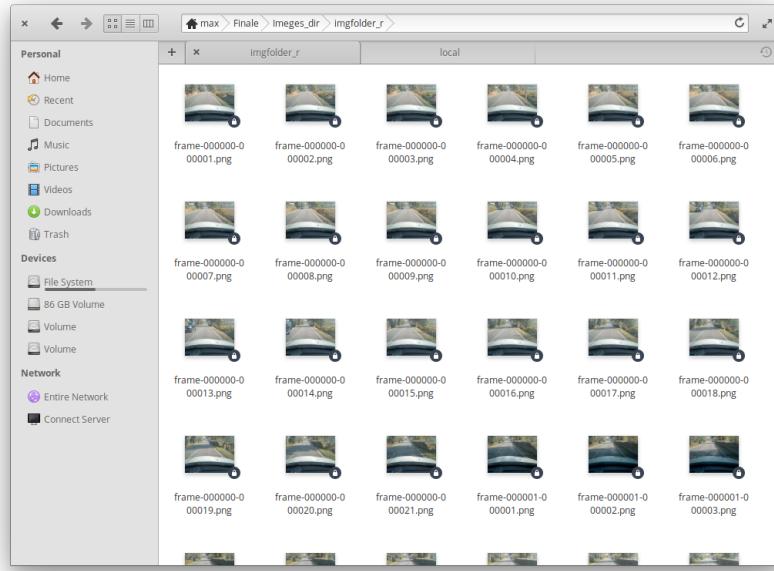


Figura 4.2: Questa cartella contiene i frames creati dal receiver.

Ognuna di queste immagini ha un nome che permette di identificare il suo GOP di appartenenza e il numero di frame all'interno del gruppo. Il nome è diviso in due sestetti: il primo identifica il GOP, il secondo il frame.

Se si desidera controllare le statistiche create si devono aprire gli ultimi file generati nelle cartelle plot e stat, contenuti a loro volta in statistics. Le statistiche si presentano nel seguente modo:

#GOP_number	frame_number_within_GOP	PSNR	distance_from_original_img	losted packet	
0	1	60	0	[]	
0	2	60	0	[]	
0	3	60	0	[]	
0	4	26.832486941276848	0	[14223]	
0	5	28.069397507703545	0	[]	
0	6	28.815776809224158	0	[]	
0	7	29.075603214941424	0	[]	
0	8	29.349965352949773	0	[]	
0	9	29.41152812744609	0	[]	
0	10	24.50315743551327	0	[14253]	
0	11	22.987715396092977	0	[]	
#Perdite	Lunghezza perdite	Delay	Jitter	Fuori ordine	Number of packets
0	0	0.006846904754638672	0.0	0	1
0	0	0.0010110855102539063	0.0002293628640472889	0	5
5	5	0.0034464597702026367	0.00041253864765167236	0	2
5	1	0.0002473890781402588	7.457606335137257e-05	0	8
4	1	0.00016329288482666015	2.734404770515439e-05	0	10
7	2	0.00011307001113891602	3.964931238442659e-06	0	4
0	0	0.00012993812561035156	0.0	0	1
1	1	0.00023937225341796875	3.096668660873547e-05	0	5
4	4	0.00014698505401611328	1.341104507446289e-07	0	2

Figura 4.3: Rispettivamente le statistiche di Receiver e Sender.

Il Receiver raccoglie dati per ogni frame creato. Con i primi due parametri si identifica il frame a cui ci si sta riferendo nel resto della riga. Nel PSNR viene calcolato il PSNR in dB. `distance_from_original_img` è diverso da 0 se non si riesce a ottenere il frame della riga. Altrimenti contiene un intero che indica la distanza con l'ultimo frame che è stato possibile decodificare. Con queste due immagini si calcola il PSNR. `losted packet` che rivela se durante la ricezione di questo frame non sono arrivati determinati pacchetti.

Nelle statistiche del Sender, ogni riga rappresenta le statistiche raccolte da un canale in una finestra. Tra una finestra e l'altra si trova una riga bianca.

Il campo `#Perdite` indica il numero di pacchetti persi. Lunghezza perdita è la media del numero di pacchetti persi di fila. `Delay` e `Jitter` sono espressi in secondi. `Fuori ordine` e `Number of packets` servono per sapere se si sono ricevuti pacchetti in un ordine diverso rispetto a quello in cui sono stati inviati e il numero di pacchetti ricevuti dal Receiver.

4.10 Modifiche al sistema

Potrebbe capitare di voler effettuare delle modifiche al sistema. Se si effettuano delle modifiche al Receiver sarà necessario compilarlo nuovamente. Questo viene effettuato ogni volta che quest'ultimo viene eseguito attraverso lo script `execute.sh` (cfr. 4.4).

4.10.1 Aggiungere statistiche

In futuro potrebbe essere necessario effettuare nuove analisi. In questo caso sarà necessario modificare Receiver e Sender per ottenere nuove statistiche. Nel primo caso ci si può trovare di fronte a due necessità diverse: nella prima non c'è necessità di raccogliere nuovi dati per definire una nuova statistica, al contrario della seconda. Andando a descrivere il secondo caso si risolve anche il primo. Le struct successive sono tutte presenti nel file header `struct.h`. La procedura da effettuare è la seguente:

1. Se è necessario ottenere nuovi dati dal pacchetto RTP, occorre aggiungere il campo che si desidera dalla struct `stat_t`. Inoltre, bisogna modificare la funzione `stat_calc()` presente nel file `receiver.c` in modo da inserire i dati nel campo aggiunto di `stat_t`.
2. A questo punto nella funzione `statThread()` si dispone dei dati necessari al calcolo delle nuove statistiche perché si utilizza la struttura `stat_t`.

riempita precedentemente, in particolare, se si è eseguito il primo punto correttamente, essa contiene i dati raccolti. Per spedire questi dati bisogna aggiungere un campo nella struttura `send_stat`, in modo che questa contenga il nuovo dato. Si possono andare ad effettuare delle operazioni nel `statThread` se si desidera lavorare sui dati.

Lato Receiver non è necessario modificare altro per la spedizione, nonostante il cambio di dimensioni della struttura `send_stat`.

Per quanto concerne il Sender bisogna modificare la classe `Stat` presente nel file `Statistiche.py`. Seguendo la stessa sintassi degli elementi in `_fields_` si deve aggiungere, nella stessa posizione in cui si è aggiunto il nuovo campo in `send_stat`, la nuova statistica che si deve ricevere. Quindi l'elemento `_fields_` dovrà presentare una nuova coppia del tipo (`nome, tipo_ctypes`). La lettura dal socket di ricezione del canale di feedback può rimanere identica.

4.10.2 Loopback Interface

Durante la prima esecuzione del Receiver in base al sistema operativo in uso, si potrebbe incappare nel seguente messaggio di errore:

```
Cannot find loopback interface , other than loopback  
interface you need a loopback with DLT_NULL ...
```

Questo significa che potrebbe non essere presente l'interfaccia di rete richiesta nel sistema, oppure che il tipo di interfaccia di rete non è utilizzabile. Nell'eventualità che questo accada si deve modificare un controllo presente nella funzione `get_loopback()` in `main.c`. In questa funzione sono presenti le righe:

```
//printf("%s %d\n", d->name, d->flags); // tipo di interfaccia  
if(d->flags == PCAP_IF_LOOPBACK || d->flags == 55)
```

L'`if` viene utilizzato per controllare la tipologia di link-layer Header. Se l'interfaccia non è del tipo `PCAP_IF_LOOPBACK` o 55 si visualizzerà l'errore mostrato prima. Per risolvere il problema è sufficiente aggiungere nell'`if`, un OR in cui si controlla un nuovo tipo di link-layer Header dell'interfaccia di loopback. Questo valore è ottenibile cancellando il commento che contiene la `printf` presente nella riga precedente all'`if`.

4.10.3 Array di dimensione prestabilita

Nel receiver sono presenti array statici ma anche dinamici che presentano una dimensione fissa comune. Se si utilizza un flusso RTP che trasporta un

video in 720p non dovrebbero creare problemi. Tuttavia se si decidesse di aumentare considerevolmente la qualità video questi buffer potrebbero saturarsi. Ci sono due soluzioni possibili a questo problema, utilizzare una realloc e rendere gli array dinamici dove necessario, oppure aumentarne la dimensione. Uno di questi array si trova in struct.h nella struttura gop_info. È possibile modificare la dimensione degli altri array cambiando il valore della macro DIMARRSTAT presente in define.h. In questo file è anche possibile modificare la dimensione della tabella hash modificando HSIZE.

Capitolo 5

Test e dimostrazione d'uso

Una volta creati Sender e Receiver si è potuto procedere con la fase di testing sul funzionamento del sistema. Si è cercato, per quanto possibile, di dimostrare il funzionamento del sistema e la qualità del video trasmesso attraverso il calcolo del PSNR.

Partendo dall'assunto che si sono utilizzati i file di configurazione, si è potuto constatare il corretto funzionamento del sistema, riuscendo poi a ottenere tutti gli elementi che permettono di capire la reale qualità della trasmissione anche offline.

Nonostante i risultati ottenuti siano buoni, ci si aspetta una modifica futura del modulo di Scheduling, in particolare dello scheduler_controller, per ottimizzare maggiormente l'uso dei canali e del modulo di feedback, se sarà necessario raccogliere più statistiche.

5.1 Test

Nella fase di testing ci si è concentrati sull'uso di tre canali, perché sarà probabilmente il caso di maggior interesse per il contesto reale. I test sono stati effettuati utilizzando sempre lo stesso file PCAP. In questo file è presente un flusso RTP che non ha subito perdite. Nei test effettuati si sono ottenuti dei risultati che dipendono fortemente dalle probabilità di perdita e dalla durata del delay che si è deciso di utilizzare per pacchetto. Si riporta il caso in cui si è deciso di utilizzare nel Sender tre canali, dove ogni canale utilizza gli stessi valori di alpha e scale per le distribuzioni gamma, come descritto in (rfc. 4.3.1) [0.125 0.4 100 0.01 0.002 0.66; 0.125 0.4 100 0.01 0.002 0.66; 0.125 0.4 100 0.01 0.002 0.66]. La trasmissione tra Sender e Receiver è avvenuta in locale.

5.1.1 Risultati

In base al numero di canali che si intende utilizzare, e nel caso in cui si decida o non di utilizzare il simulate mode, si ottengono diversi risultati.

Nel caso in cui si decida di usare il simulate mode il numero di canali non incide sulla qualità del video finale. L'unico caso in cui il simulate mode può creare problemi è quando si utilizzano troppi canali e si va a saturare l'interfaccia di rete o, in generale, quando si utilizzano troppe risorse del computer.

Togliendo questi casi sfavorevoli, si va ad approfondire il caso in cui non si utilizza il simulate mode. Ovviamente aumentando il numero di canali si ritrova una certa congruenza nella qualità dello streaming. A seconda della tipologia del pacchetto che si va a perdere la qualità del GOP ne può risentire parecchio o essere per lo più ininfluente:

- Primo pacchetto di un I-frame: Disastro! non si riesce a costruire il GOP.
- Pacchetto non iniziale del I-frame: la qualità rimane molto bassa e non si riesce chiaramente a capire cosa succede nel video, soprattutto negli ultimi frame del GOP.
- Perdita di un pacchetto di uno dei primi P-frame: ogni frame successivo subirà un contraccolpo, il calcolo del PSNR restituirà bene o male sempre lo stesso risultato.
- Perdita di un pacchetto di uno degli ultimi P-frame: solo i frame finali avranno una qualità peggiore.

Se si utilizza *un canale*, con le impostazioni descritte precedentemente si perde almeno un pacchetto per GOP. Se si dovessero perdere più di due pacchetti di fila per GOP, la qualità ne risente molto, in particolare se si tratta di I-frame. Con l'introduzione di *due canali* questa eventualità è molto ridotta, ma si è notato che incide comunque un GOP ogni 5. Utilizzando *tre canali* può accadere che qualche pacchetto si perda, ma è un evento rarissimo. Dai test effettuati si è notato che effettivamente l'utilizzo di 3 canali è molto conveniente.

5.2 Dimostrazione d'uso

All'avvio del Sender, se anche il Receiver si è connesso, si presenta davanti la seguente schermata del terminale.

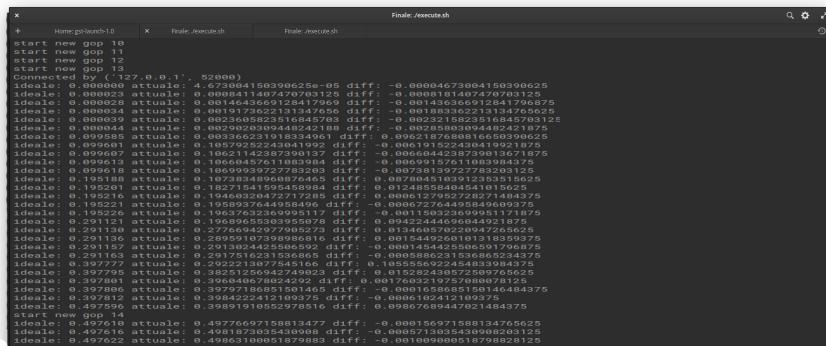


Figura 5.1: Il Sender ha appena avviato la trasmissione dati.

In questa finestra del terminale si fa riferimento a start new gop X. Questo indica che il Sender sta per creare e salvare i frames del GOP X nel disco. Connected by() serve per sapere che la comunicazione con Receiver è iniziata. Le righe sottostanti vengono stampate dallo Scheduler ogni volta che si deve schedulare un nuovo pacchetto. *Ideale* indica il tempo di spedizione del pacchetto i-esimo del PCAP, mentre *attuale* indica la differenza di tempo tra la schedulazione del primo pacchetto avvenuta nel sistema e il tempo attuale del sistema. Questi due valori sono utili per il calcolo di *diff*. Se *diff* è negativo vuol dire che lo scheduler doveva aver già schedulato il pacchetto, se è positivo vuol dire che anticiperà l'invio del pacchetto. Una schermata tipica del Receiver è quella sottostante:

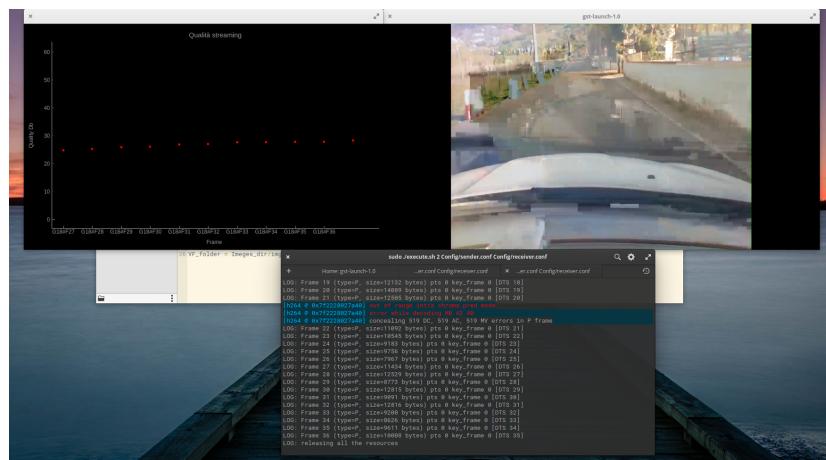


Figura 5.2: Esecuzione del Receiver con una distorsione dell'immagine.

In questa immagine si trova in basso la finestra del terminale che lo esegue, in alto a destra si trova la schermata di Gstreamer che mostra il video e in alto a sinistra è presente il grafico che mostra la distorsione dell'immagine. Come si può notare nel terminale è presente una striscia colorata, la quale indica che non sono stati ricevuti tutti i frame di un determinato GOP e questo si riflette nell'immagine che si sta visualizzando. Infatti il frame che Gstreamer sta mostrando risulta leggermente distorto. Dal grafico è possibile quantificare la distorsione della qualità dell'immagine (cfr. 3.2.8), che mostra un valore del PSNR di circa 30 dB.

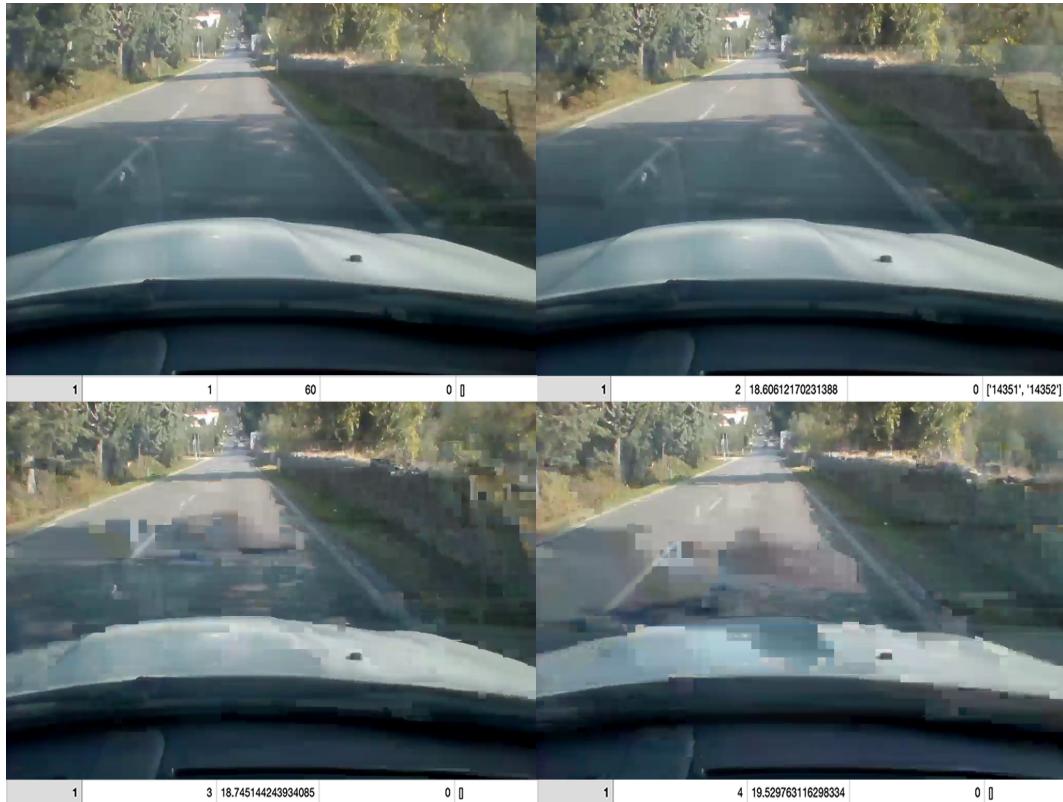


Figura 5.3: Quattro frame dello stesso GOP, viene mostrata la progressiva perdita di qualità dell'immagine.

In questa immagine sono presenti quattro frame dello stesso GOP. Sotto ad ogni immagine sono stati riportati dati che lo riguardano nel seguente ordine: numero di GOP, numero di frame nel GOP, PSNR e un array degli eventuali pacchetti persi. Nella prima immagine in alto a sinistra si ha un'immagine creata senza subire distorsioni di alcun tipo, infatti il suo PSNR è di 60 dB. Il frame successivo ha subito una leggera distorsione perché non si sono ricevuti due pacchetti che la compongono. Anche se si riporta un PSNR basso l'immagine risulta ancora molto chiara. I due frame in basso sono quelli che

seguono la seconda immagine. Anche se non si sono subite perdita la qualità cala drasticamente vedendo addirittura dei pixel a schermo. In questi casi il PSNR risulta essere maggiore del secondo frame, anche se di poco. È interessante notare come anche una piccola perdita di pacchetti all'inizio del GOP può portare a una distorsione sempre maggiore nei frame successivi pur non incidendo sul valore del PSNR.

Capitolo 6

Conclusioni

L’obiettivo di questo tirocinio è lo sviluppo di un simulatore/emulatore per trasmissione di flussi multimediali su più canali fisici. Ciò viene fatto per ragioni di affidabilità e qualità della trasmissione, che si intendono perseguire. Lo sviluppo ha riguardato la creazione di tre moduli logici principali: il sender, i canali fisici, e il receiver. Il primo si occupa di inviare il video al receiver tramite gli N canali fisici disponibili in accordo con le statistiche di ciascun canale, ricevute tramite un anello di feedback dal receiver; lo scheduler, in accordo con queste ultime, decide quali e quanti canali fisici utilizzare. Ciascun canale è infatti caratterizzato da una serie di parametri, quali: ip del canale, porta da utilizzare, coppia di valori alpha e scale per andare a costruire le distribuzioni gamma di evento di perdita, numero di perdite e delay. Infine, il receiver si occupa di: ricostruire il flusso multimediale ricevuto, visualizzarlo, fornire in modo quasi istantaneo il PSNR di ciascun frame video, nonché inviare al sender le statistiche di canale, analizzando i flussi di pacchetti ricevuti da ciascun canale.

Tale simulatore è d’interesse per diversi scenari applicativi, come ad esempio quello qui considerato, cioè la trasmissione di un flusso video da bordo drone a terra per il tramite di più canali fisici, per esempio connessioni alla rete cellulare multiple. Oltre a descrivere l’architettura del simulatore, i suoi componenti principali, ho descritto un breve caso di test per mostrarne il funzionamento. Il simulatore ha potenzialità di sviluppo future ed è quindi rilasciato pubblicamente al link <https://github.com/bohmax/tirocinio>.

Durante lo sviluppo si è incontrata una maggiore difficoltà nell’analisi dello standard RTP. Infatti non è stato facile individuare i passi necessari per la decodifica del flusso. Superata questa difficoltà ci si è concentrati sulla soluzione di altre problematiche, quali scartare ricezioni multiple di uno stesso pacchetto

o in particolare è stato molto difficile identificare l'inizio e la fine di un GOP, perché si è dovuto stare dietro a ritardi di rete, perdite di pacchetti e arrivi disordinati di questi.

Mi auguro che i risultati ottenuti saranno utili e renderanno la vita più semplice ai colleghi che dovranno sviluppare il sistema in un contesto reale.

Bibliografia

- [1] *Real-time Transport Protocol*. URL: https://en.wikipedia.org/wiki/Real-time_Transport_Protocol.
- [2] *RTP Payload Format for H.264 Video*. 2011. URL: <https://tools.ietf.org/html/rfc6184>.
- [3] *RTP Profile for Audio and Video Conferences with Minimal Control*. 2003. URL: <https://tools.ietf.org/html/rfc3551>.
- [4] *RTP: A Transport Protocol for Real-Time Applications*. 2003. URL: <https://tools.ietf.org/html/rfc3550>.
- [5] «Video compression picture types». URL: https://en.wikipedia.org/wiki/Video_compression_picture_types.