



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	5
1 Scenario	7
1.1 Sviluppo	7
1.2 Real Time Protocol (RTP)	8
1.3 Scheduling controller	8
1.4 Group of picture (GOP)	8
1.5 Peak Signal-to-Noise Ratio (PSNR)	8
2 Flussi multimediali RTP/UDP	10
2.1 Analisi RFC 3550	10
2.2 RTP caso d'uso	12
2.2.1 RFC 6184	12
2.2.1.1 RFC 6184: Data Header	13
2.2.1.2 Modalità di pacchettizzazione	14
2.2.1.3 Fragmentation Units (FUs)	15
2.2.2 Algoritmi di compressione video	17
2.2.2.1 H.264 approfondimento	18
2.2.2.2 FU(s) come ottenere il frame type corrente	19
3 Simulatore: Architettura e Funzionamento	21
3.1 Sender	22
3.1.1 Canali Fisici	22
3.1.2 Classe Operatore	22
3.1.3 Classe Sender	22
3.1.4 Scheduler	24
3.1.5 Comunicazione Scheduler-Process	25
3.1.6 Classe Timing	26
3.2 Receiver	27
3.2.1 Threads e Process	27

3.2.2	Comunicazioni	28
3.2.3	listenerThread	30
3.2.4	GOPThread	30
3.2.5	ReaderPacket	32
3.2.6	Ottenere i pacchetti	33
3.2.6.1	Algoritmo per il salvataggio del GOP	33
3.2.7	DecoderThread	34
3.2.8	PlotProc	35
3.2.9	statThread	36
3.2.9.1	Ritardo	37
3.2.9.2	Variabilità del ritardo (jitter)	37
3.2.9.3	Pacchetti fuori ordine	37
3.2.9.4	Lunghezza media della perdita	38
3.2.9.5	Numero di pacchetti persi	38
3.2.10	Segnali	38
3.2.11	Gstreamer	39
4	Manuale utente e sviluppatore	41
4.1	Requisiti e dipendenze	41
4.1.1	Dipendenze di sistema	41
4.1.2	Dipendenze del sender	43
4.1.3	Dipendenze del client	43
4.2	Primo utilizzo	44
4.3	Config	44
4.3.1	File di configurazione del Sender	44
4.3.2	File di configurazione del Receiver	45
4.4	execute.sh	46
4.5	ImagePicker	47
4.6	Script	47
4.7	Sender	47
4.8	statistic	47
4.9	Come utilizzare il sistema	48
4.10	Modifiche al sistema	50
4.10.1	Aggiungere statistiche	50
4.10.2	Loopback Interface	51
4.10.3	Array di dimensione prestabilita	51

5 Test e dimostrazione d'uso	53
5.1 Test	53
5.1.1 Risultati	54
5.2 Dimostrazione d'uso	55
6 Conclusioni	59
A Codice	61
A.1 Codice Receiver	61
A.1.1 utility.h	61
A.1.2 utility_fun.c	63
A.1.3 main.c	64
A.1.4 utility_fun.c	70
A.1.5 struct_fun.c	75
A.1.6 receiver.c	77
A.1.7 utility_fun.c	83
A.1.8 lista.c	91
A.1.9 icl_hash.c	92
A.1.10 threads.c	98
A.2 Sender	105
A.2.1 Scheduler.py	105
A.2.2 Timing.py	113
A.2.3 Operatore.py	115
A.2.4 Sender.py	117
A.2.5 Statistiche.py	122
A.2.6 Scheduler_controller.py	125
A.2.7 Image_Handler.py	128
A.3 Bash Script	134
A.3.1 execute.sh	134
A.3.2 receiver.sh	135
A.3.3 sender.sh	138

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:

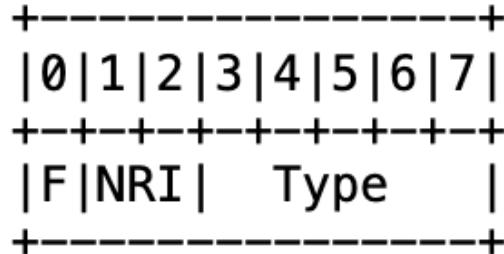


Figura 2.2: Serve per capire come interpretare il resto del pacchetto RTP.

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.

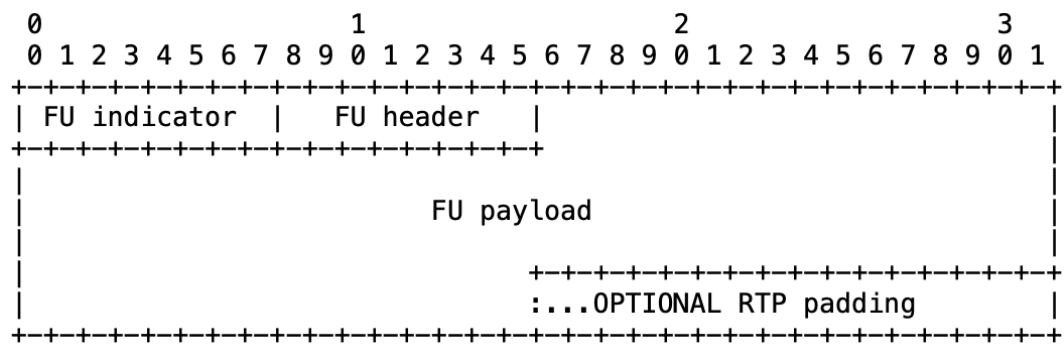


Figura 2.4: Un FU-A consiste in un fragment unit undicator rappresentato da un otteto, di un fragmentation unit header composto anche esso da un otteto e un fragmentation unit payload

L'ottetto del FU indicator ha il seguente formato:

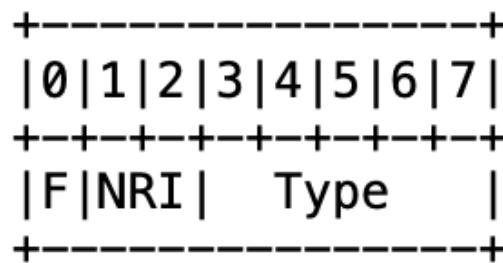


Figura 2.5: FU-A indicator

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:

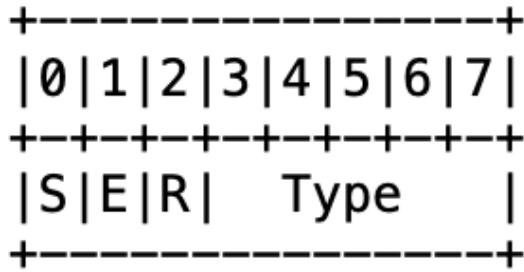


Figura 2.6: FU-A header

- 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:

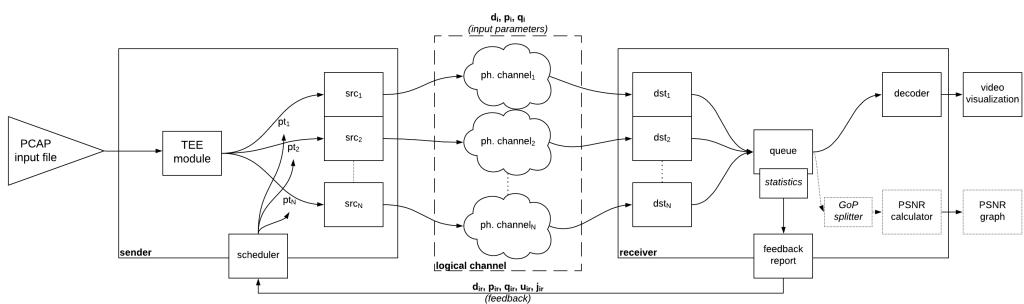


Figura 3.1: Schema del sistema.

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:

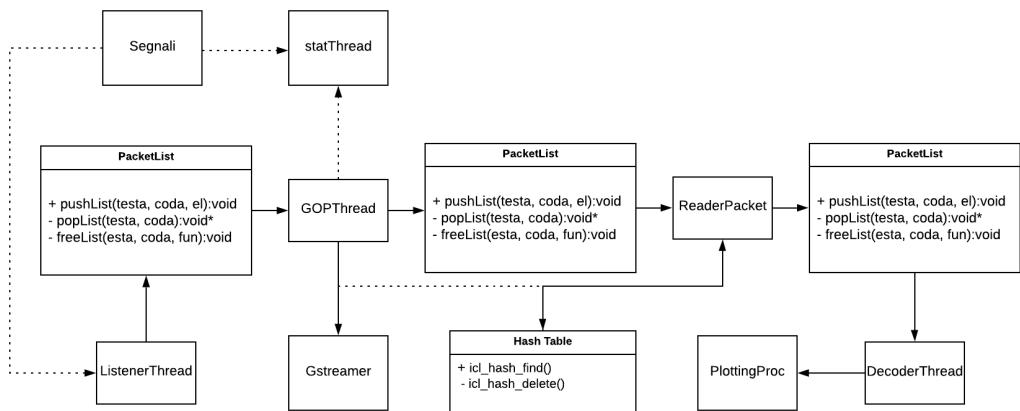


Figura 3.2: Le linee continue indicano una comunicazione diretta tramite liste, le altre una comunicazione indiretta con strutture dati condivise.

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

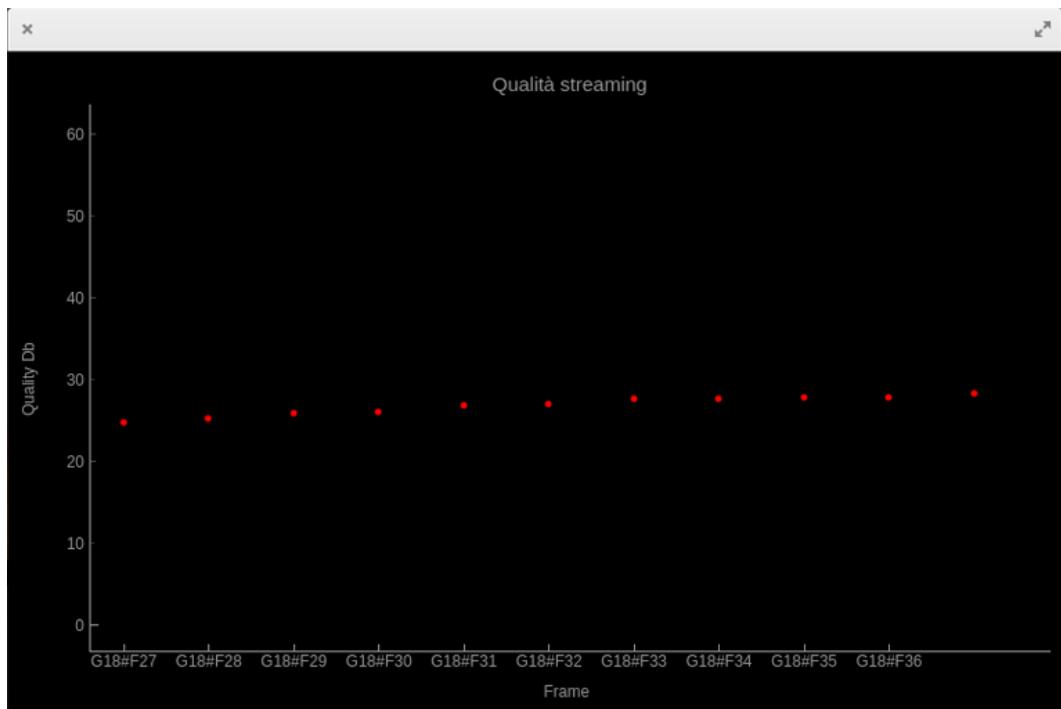


Figura 3.3: Grafico che mostra la qualità dell'immagine.

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-
    threads
    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:

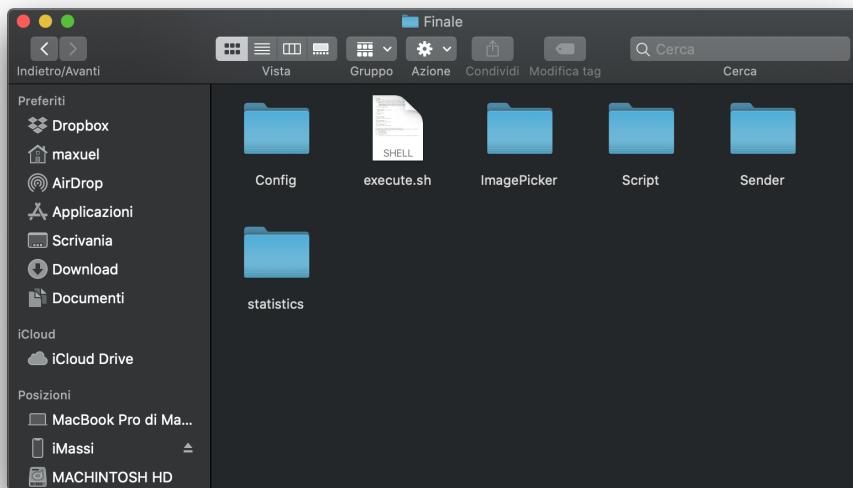


Figura 4.1: Cartelle del sistema.

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:

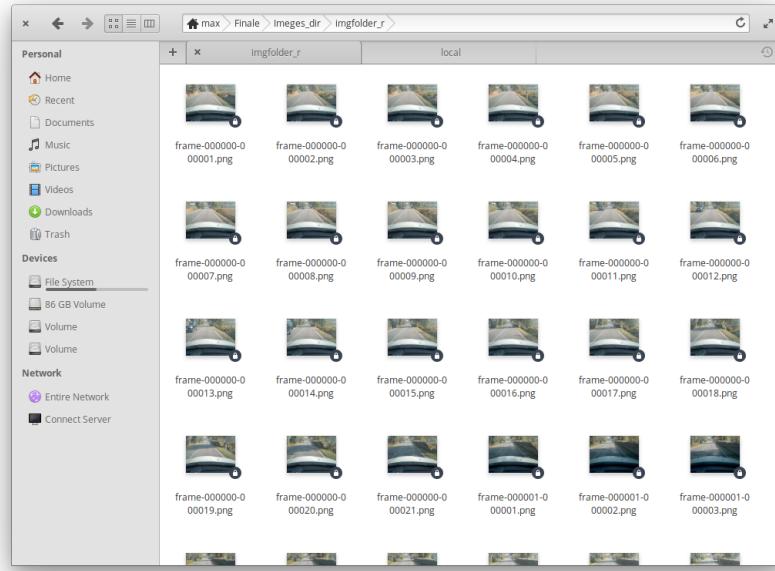


Figura 4.2: Questa cartella contiene i frames creati dal receiver.

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.

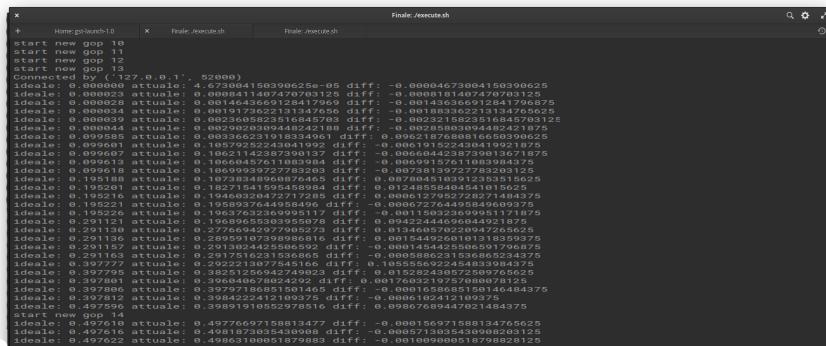


Figura 5.1: Il Sender ha appena avviato la trasmissione dati.

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:

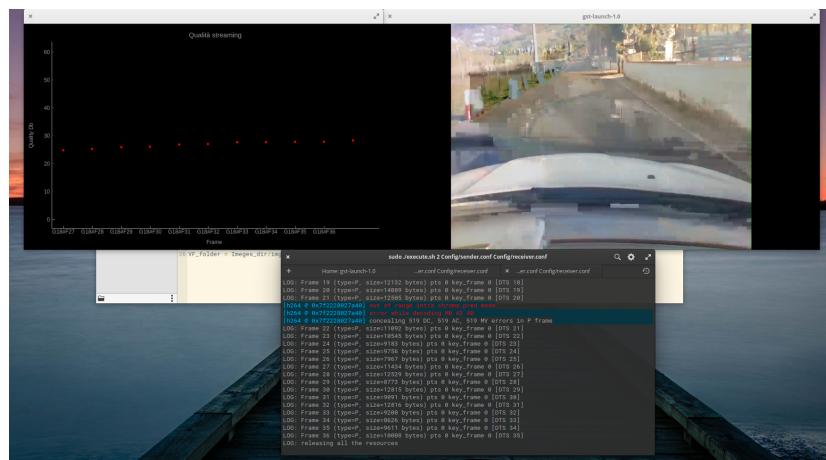


Figura 5.2: Esecuzione del Receiver con una distorsione dell'immagine.

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.

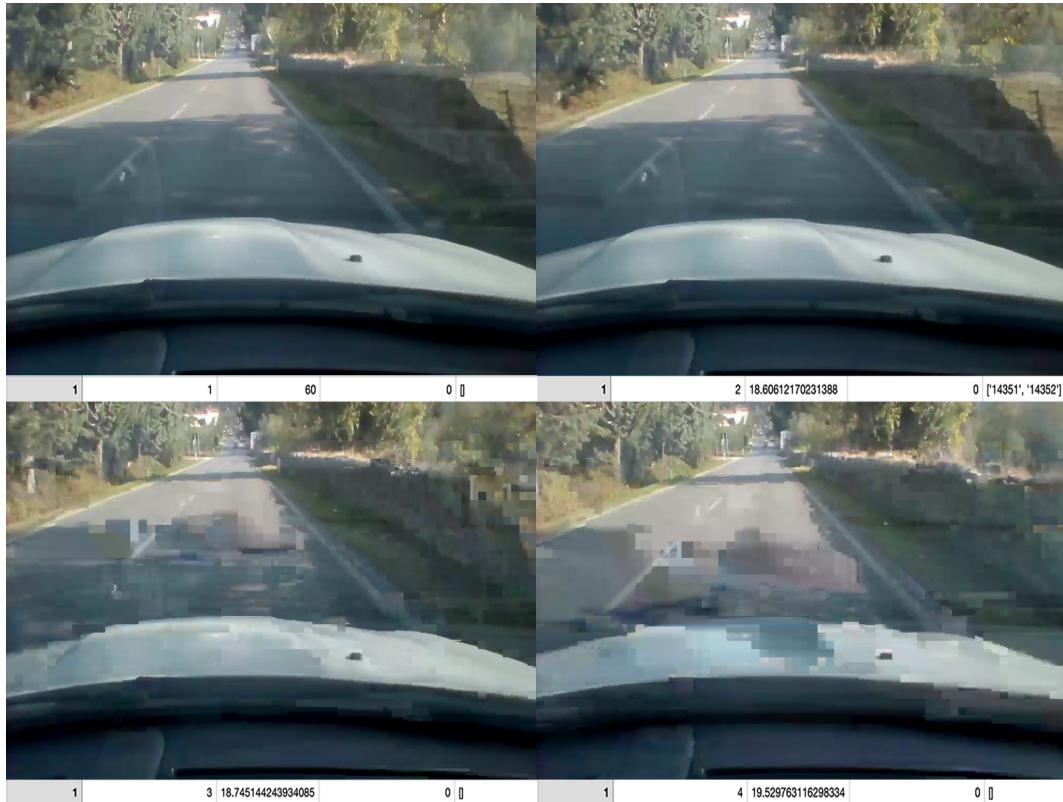


Figura 5.3: Quattro frame dello stesso GOP, viene mostrata la progressiva perdita di qualità dell'immagine.

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 leggere 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.

Appendice A

Codice

A.1 Codice Receiver

A.1.1 utility.h

```
1 #ifndef utility_h
2 #define utility_h
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #include <time.h>
8 #include <pcap.h>
9 #include <string.h>
10 #include <errno.h>
11 #include <pthread.h>
12 #include <signal.h>
13 #include <net/ethernet.h>
14 #include <arpa/inet.h>
15 #include <netinet/ip.h>
16 #include <netinet/udp.h>
17 #include <libavcodec/avcodec.h>
18 #include <libavformat/avformat.h>
19 #include <libavutil/imgutils.h>
20 #include <libswscale/swscale.h>
21 #include <libavformat/avio.h>
22 #include "packet_structs.h"
23 #include "define.h"
24 #include "struct.h"
25 #include "utility_fun.h"
26 #include "struct_fun.h"
27 #include "receiver.h"
```

```

28 #include "threads.h"
29 #include "h264toPng.h"
30 #include "lista.h"
31 #include "icl_hash.h"
32
33 extern FILE* pipe_plot; //pipe per comunicare con il grafico
    delle statistiche
34 extern struct sockaddr_in servaddr; /* indirizzi per inoltrare i
    pacchetti udp*/
35 extern int fd; /* file descriptor del socket */
36 extern pthread_t *listener, stat_thr, segnal; //thread listener,
    statistiche e segnali
37 extern icl_hash_t* hash_packet; /* hash table contentente i
    pacchetti da ordinare */
38 extern pcap_t** handle; /* packet capture handles */
39 extern pcap_t* loopback; /* loopback interface */
40 extern int from_loopback; /* booleano, indica se loopback e
    handle coincidono */
41 extern int datalink_loopback; /* intero che indica il tipo del
    livello 2 del pacchetto */
42 extern string payload; /* contiene la codifica di un h264 */
43 extern string metadata; //dovr contenere SPS e PPS
44 extern gop_info* info; /* utilizzato per creare le immagini */
45 extern char* path_file; /* path del file su cui viene salvato lo
    streaming*/
46 extern char* path_image; /* path su cui saranno salvate le
    immagini*/
47 extern char* path_image_sender; /* path delle immagini create dal
    sender */
48 extern int esci; /* indica l'uscita dal programma*/
49 extern int num_pkt; //numero dei pacchetti arrivati
50 extern pthread_mutex_t mtx_gop; /* mutex per fare produttore
    consumatore con il decodificatore */
51 extern pthread_cond_t cond_gop; /* variabile di condizione per
    produttore consumatore*/
52 extern pthread_mutex_t mtx_ord; /* mutex per ordinare i pacchetti
    */
53 extern pthread_cond_t cond_ord; /* variabile di condizione per
    ordinare i pacchetti*/
54 extern pthread_mutex_t mtx_dec; /* mutex per fare produttore
    consumatore con il decodificatore */
55 extern pthread_cond_t cond_dec; /* variabile di condizione per
    produttore consumatore*/
56 extern pthread_mutex_t mtx_info; /* mutex per poter utilizzare la
    variabile info */

```

```

57 extern pthread_mutex_t mtxhash[HSIZE/DIV]; /* mutex della tabella
   hash*/
58 extern pthread_mutex_t* mtxstat; /* mutex per modificare le
   statistiche */
59 extern pthread_mutex_t mtxplot; /* mutex per scrivere nella pipe
   del plot */
60 extern list* testa_gop;
61 extern list* coda_gop; /* coda della lista sopra */
62 extern list* testa_ord; /* lista dei pacchetti */
63 extern list* coda_ord;
64 extern list* testa_dec;
65 extern list* coda_dec; /* coda della lista sopra */
66 extern sigset_t sigset_usr;
67 extern stat_t* statistiche;
68 extern int num_list, from_port, stat_port, video_port, stat_interv
   , num_decoder; /* variabili per l'input */
69 extern char ip_sender[DIM_IP];
70
71 #endif /* utility_h */

```

A.1.2 utility_fun.c

```

1 #ifndef define_h
2 #define define_h
3
4 #define ERRORSYSHEANDLER(r,c,d,e) if((r==c)==d) { perror(e); exit(
   errno); }
5 #define SYSFREE(r,c,d,e) if((r==c)!=d) { perror(e); exit(errno); }
6 #define MAXBUF 1442 /* dimensione per ogni pacchetto di libpcap */
7 #define DPORT 9998 /* destination port del lettore video */
8 #define LFINESTRA 200000 /* indica la lunghezza della finestra,
   cio il tempo che deve trascorre per inviare le statistiche */
9 //##define DIM 500 //dimensione media per 2 di sps e pps
10 #define DIMARRSTAT 1024 //dimensione array statistiche
11 #define GOPM 2000000 //dimensioni di un GOP medio moltiplicato per
   2
12 #define HSIZE 4000 /* dimesione della tebella hash contenente i
   pacchetti. dimensione basata sul gop medio */
13 #define DIV 50 //grandezza partizione della tebella hash
14 #define NUMLISTTHR 2 /* numero di thread listener */
15 #define NUMDECODERTHR 1 //numero di thread per decodificare le
   immagini
16 #define MACLENGTH 6 //lunghezza mac address
17 #define MACADDR 0x00 //mac address da settare su un pacchetto
   ethernet da inviare a loopback

```

```

18 #define HOSTNAME "127.0.0.1" //host name su cui inoltrare i
    pacchetti rtp
19 #define DIM_IP 15 // dimensione di un ip in formato stringa
20 #define DIM_TIMESTAMP 8 // dimensione del timestamp aggiunto nei
    pacchetti
21 #define JITTER_DIV 16
22 typedef int make_iso_compilers_happy;
23
24 #endif /* define_h */

```

A.1.3 main.c

```

1 //
2 //  main.c
3 //  ImagePicker
4 //
5 //  Created by Massimo Puddu on 30/03/2020.
6 //  Copyright 2020 Massimo Puddu. All rights reserved.
7 //
8
9 #include "utility.h"
10
11 icl_hash_t* hash_packet = NULL; /* definita tutto in utility.h */
12 char* path_file = NULL;
13 char* path_image = NULL;
14 char* path_image_sender = NULL; //path delle immagini create dal
    sender
15 list* testa_gop = NULL;
16 list* coda_gop = NULL;
17 list* testa_ord = NULL;
18 list* coda_ord = NULL;
19 list* testa_dec = NULL;
20 list* coda_dec = NULL;
21 string metadata; //dovr contenere SPS e PPS
22 string payload; //dovr contenere un intero GOP
23 gop_info* info; //usato per capire quali pacchetti devono essere
    decodificati
24 pthread_mutex_t mtx_gop = PTHREAD_MUTEX_INITIALIZER;
25 pthread_cond_t cond_gop = PTHREAD_COND_INITIALIZER;
26 pthread_mutex_t mtx_dec = PTHREAD_MUTEX_INITIALIZER;
27 pthread_cond_t cond_dec = PTHREAD_COND_INITIALIZER;
28 pthread_mutex_t mtx_ord = PTHREAD_MUTEX_INITIALIZER;
29 pthread_mutex_t mtx_info = PTHREAD_MUTEX_INITIALIZER;
30 pthread_mutex_t mtxplot = PTHREAD_MUTEX_INITIALIZER;
31 pthread_cond_t cond_ord = PTHREAD_COND_INITIALIZER;

```

```

32 pthread_mutex_t mtxhash[HSIZE/DIV];
33 pthread_mutex_t* mtxstat;
34 sigset(SIGUSR1, sigset_usr); /* maschera globale dei segnali */
35 pcap_t** handle; /* packet capture handle */
36 pcap_t* loopback; /* loopback interface */
37 pthread_t *listener, stat_thr, segnal; //thread listener,
   statistiche e segnali
38 stat_t* statistiche; //statistiche dei thread listener
39 int esce = 0;
40 int from_loopback = 0;
41 int datalink_loopback = 0;
42 struct sockaddr_in servaddr;
43 int fd; //file descriptor del socket per inoltrare nuovamente i
   pacchetti
44 FILE* pipe_plot;
45 int num_list = 1, from_port = 5000, stat_port = DPORT, video_port
   = DPORT, stat_interv = LFINESTRA, num_decoder = NUMDECODERTHR;
46 char ip_sender[DIM_IP];
47
48 void set_pipe(){
49     pipe_plot = popen("python3 ImagePicker1/plot.py", "w");
50     if (pipe_plot == NULL) {
51         printf("popen error\n");
52         exit(1);
53     }
54 }
55
56 void get_loopback(pcap_if_t* alldevs, char name[], char errbuf[]){
57     pcap_if_t *d=alldevs;
58     while(d!=NULL) {
59         //printf("%s %d\n", d->name, d->flags); per controllare il
         tipo di interfaccia
60         if(d->flags == PCAP_IF_LOOPBACK || d->flags == 55 || d->
         flags == 7){
61             if (!strcmp(d->name, name)){
62                 from_loopback = 1;
63                 loopback = handle[0];
64             }
65             else{
66                 loopback = pcap_open_live(d->name, BUFSIZ, 0,
9500, errbuf); //ottengo uno sniffer
67                 if(loopback == NULL){
68                     printf("pcap_open() loopback failed due to [%s
] \n", errbuf);
69                     exit(1);

```

```

70         }
71     }
72     datalink_loopback = pcap_datalink(loopback);
73     return;
74 }
75 d=d->next;
76 }
77 printf("Cannot find loopback interface, other than loopback
78 interface you need a loopback with DLT_NULL link type header,
79 otherwise if you know what you are doing change the control on
80 the flag above in code\n");
81 exit(1);
82 }
83
84 void set_handler(char device_name[], int index, struct bpf_program
85 * fp, char filter[], char errbuf[]){
86     bpf_u_int32 pMask;           /* subnet mask */
87     bpf_u_int32 pNet;           /* ip address*/
88     handle[index] = pcap_open_live(device_name, MAXBUF, 0, 100,
89     errbuf); //ottengo uno sniffer
90     if(handle[index] == NULL){
91         handle[index] = pcap_open_offline(device_name, errbuf);
92         if(handle == NULL){
93             printf("pcap_open() failed due to [%s]\n", errbuf);
94             exit(1);
95         }
96     }
97     pcap_lookupnet(device_name, &pNet, &pMask, errbuf);
98     if(pcap_compile(handle[index], fp, filter, 0, pNet) == -1){ //usato per compilare la stringa str per il bpf filter
99         printf("\npcap_compile() failed\n");
100        exit(1);
101    }
102 }
103
104 void set_signal(){
105     int notused;
106     ERRORSYSHEANDLER(notused, sigemptyset(&sigset_usr), -1, "NO
107     SIGEMPTY 1")
108     ERRORSYSHEANDLER(notused, sigaddset( &sigset_usr , SIGUSR1), -1, "

```

```

    NO ADDSET" )
108     ERRORSYSHEANDLER( notused , sigaddset( &sigset_usr , SIGINT) ,-1,"
NO ADDSET" )
109     ERRORSYSHEANDLER( notused , sigaddset( &sigset_usr , SIGTERM) ,-1,"
NO ADDSET" )
110     ERRORSYSHEANDLER( notused , sigaddset( &sigset_usr , SIGQUIT) ,-1,"
NO ADDSET" )
111     ERRORSYSHEANDLER( notused , sigaddset( &sigset_usr , SIGPIPE) ,-1,"
NO ADDSET" )
112     ERRORSYSHEANDLER( notused , pthread_sigmask(SIG_SETMASK, &
113         sigset_usr , NULL) ,-1,"NO SIGMAS" )
114 }

115 void initialize_mtx(){
116     int notused , size = HSIZE/DIV;
117     mtxstat = malloc( sizeof(pthread_mutex_t)*num_list );
118     for( int i=0; i < size; i++){
119         if((notused(pthread_mutex_init(&mtxhash[ i ] , NULL)<0)){
120             perror("impossibile inizializzare mutex");
121             exit(errno);
122         }
123     }
124     for( int i=0; i < num_list; i++){
125         if((notused(pthread_mutex_init(&mtxstat[ i ] , NULL)<0)){
126             perror("impossibile inizializzare mutex");
127             exit(errno);
128         }
129     }
130 }

131
132 void set_socket(){
133     fd = socket(PF_INET, SOCK_DGRAM, 0);
134     if(fd<0){
135         perror("cannot open socket");
136         exit(-1);
137     }
138
139     bzero(&servaddr , sizeof(servaddr));
140     servaddr.sin_family = AF_INET;
141     servaddr.sin_addr.s_addr = inet_addr(HOSTNAME);
142     servaddr.sin_port = htons(video_port);
143 }

144
145 int main(int argc , const char * argv[] ) {
146     clock_t begin = clock();

```

```

147     int notused; //variabile usata per memorizzare valori di
148     ritorno di alcune chiamate di sistema
149     char *dev_name = NULL; /* capture device name */
150     char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
151     char* stringfilter = malloc(64);
152     memset(stringfilter, '0', 64);
153     pcap_if_t *alldevs = NULL;
154     pthread_t order;
155     struct bpf_program* fp = NULL; /* to hold compiled
156     program */

157     if (argc != 12) exit(EXIT_FAILURE);
158     dev_name = (char*) argv[1];
159     memcpy(ip_sender, argv[2], strlen(argv[2]));
160     num_list = atoi(argv[3]);
161     from_port = atoi(argv[4]);
162     stat_port = atoi(argv[5]);
163     video_port = atoi(argv[6]);
164     path_file = (char*) argv[7];
165     path_image = (char*) argv[8];
166     path_image_sender = (char*) argv[9];
167     stat_interv = atoi(argv[10]);
168     num_decoder = atoi(argv[11]);

169     statistiche = malloc(sizeof(stat_t)*num_list);
170     listener = malloc(sizeof(pthread_t)*num_list);
171     handle = malloc(sizeof(pcap_t)*num_list);
172     fp = malloc(sizeof(struct bpf_program)*num_list);
173     for (int i = 0; i<num_list; i++){
174         memset(&statistiche[i], 0, sizeof(stat_t));
175         statistiche[i].pkt_information = malloc(sizeof(pkt_info)*
176 DIMARRSTAT);
176         memset(statistiche[i].pkt_information, 0, sizeof(pkt_info)*
177 DIMARRSTAT);
177         statistiche[i].min = -1;
178         memset(&fp[i], 0, sizeof(struct bpf_program));
179     }

180     if (pcap.findalldevs(&alldevs, errbuf)==-1) exit(EXIT_FAILURE);
181     set_pipe();
182     set_socket();
183     set_signal();
184     initialize_mtx();
185     info = setElGOP(-1, -1); //primo gop, parte da 0
186     hash_packet = icl_hash_create(HSIZE, uint16_hash_function,
187

```

```

        uint_16t_key_compare);
188 //avvio thread che gestisce i segnali
189 SYSFREE(notused, pthread_create(&segnal, NULL, &Segnali, NULL), 0, "thread")
190 SYSFREE(notused, pthread_create(&order, NULL, &ReaderPacket, NULL),
191 , 0, "thread") // ctrl-c to stop sniffing
192 //device = find_device(alldevs, dev_name); //se null provoca leggere offline
193 printf("Sniffing on device: %s\n", dev_name);
194 for (int i=0; i<num_list; i++){
195     // fetch the network address and network mask
196     int* id = malloc(sizeof(int));
197     *id = i;
198     sprintf(stringfilter, "not icmp and udp and dst port %d",
199 (from_port + i));
200     set_handler(dev_name, i, &fp[i], stringfilter, errbuf);
201     if (!loopback)
202         get_loopback(alldevs, dev_name, errbuf);
203     SYSFREE(notused, pthread_create(&listener[i], NULL,
204 listenerThread, id), 0, "thread")
205 }
206 SYSFREE(notused, pthread_create(&stat_thr, NULL, &statThread, NULL),
207 , 0, "thread statistiche") //ora sono pronto per le statistiche
208 printf("Aspetta i thread\n");
209 for (int i=0; i<num_list; i++)
210     SYSFREE(notused, pthread_join(listener[i], NULL), 0, "listener
211 1")
212 /* libera lista e invia segnale di chiusura a order */
213 pthread_mutex_lock(&mtx_ord);
214 pushList(&testa_ord, &coda_ord, setElGOP(-1, -1));
215 pthread_cond_signal(&cond_ord);
216 pthread_mutex_unlock(&mtx_ord);
217 SYSFREE(notused, pthread_join(order, NULL), 0, "join order")
218 pthread_kill(segnal, SIGINT); //manda un segnale al thread
219 SYSFREE(notused, pthread_join(stat_thr, NULL), 0, "join
220 statistiche")
221 SYSFREE(notused, pthread_join(segnal, NULL), 0, "join 1")
222 printf("Uscita dal programma\n");
223 for (int i = 0; i < num_list; i++) {
224     pcap_freecode(&fp[i]);
225     pcap_close(handle[i]);
226 }
227 if (!from_loopback)
228     pcap_close(loopback);
229 pcap_freealldevs(alldevs);

```

```

224 // freeList(&testa_gop , &coda_gop , &freeRTP);
225 // freeList(&testa_dec , &coda_dec , &freeGOP);
226 // freeList(&testa_ord , &coda_ord , &freeGOP);
227 icl_hash_destroy (hash_packet , &freeKeyHash , &freeElHash );
228 free (fp);
229 free (handle);
230 for ( int i=0; i<num_list ; i++)
231     free ( statistiche [ i ]. pkt_information );
232 free ( statistiche );
233 free ( listener );
234 free ( stringfilter );
235 free ( mtxstat );
236 clock_t end = clock ();
237 close (fd);
238 //chiudi plot
239 fprintf (pipe_plot , "Esci\n");
240 fclose (pipe_plot );
241 printf ("Tempo di esecuzione %f\n" , (double)(end - begin) /
CLOCKS_PER_SEC );
242 return 0;
243 }
```

A.1.4 utility_fun.c

```

1 #include "utility_fun.h"
2
3 //inizializzazione socket , ritorna socket fd
4 int set_stat_sock(){
5     int sockfd , tentativi=0;
6     struct sockaddr_in servaddr;
7     // socket create and varification
8     sockfd = socket (AF_INET , SOCK_STREAM, 0);
9     if (sockfd == -1) {
10         printf("socket creation failed...\n");
11         exit(0);
12     }
13     else
14         printf("Socket successfully created..\n");
15     bzero (&servaddr , sizeof(servaddr));
16     // assign IP , PORT
17     servaddr.sin_family = AF_INET;
18     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
19     servaddr.sin_port = htons(stat_port);
20     // connect the client socket to server socket
21     while (tentativi < 5 && !esci) {
```

```

22     if (connect(sockfd , ( struct sockaddr *)&servaddr , sizeof(
23         servaddr)) != 0) {
24         printf("connection with the server failed...\\n");
25         sleep(2);
26         printf("Altri %d tentativi , prima della chiusura\\n" , 4
27             - tentativi);
28         tentativi++;
29     }
30     else
31         break;
32 }
33 if (tentativi == 5 && !esci){
34     printf("Impossibile connettersi al server , chiudo\\n");
35     fprintf(pipe_plot , "Esci\\n");
36     fclose(pipe_plot);
37     exit(-1);
38 } else
39     printf("connected to the server..\\n");
40 return sockfd;
41 //funzione di comparazione per il qsort
42 int cmpfunc (const void * a, const void * b){
43     return ( (*(pkt_info*)a).ids - (*(pkt_info*)b).ids );
44 }
45
46 uint16_t stat_lunghezza(pkt_info arr[] , int index){
47     int num_lost , num_buchi = 0, tot = 0, j;
48     for (j = 1; j < index; j++) {
49         num_lost = arr[j].ids - arr[j-1].ids - 1;
50         if (num_lost > 0)
51             num_buchi++;
52         tot += num_lost;
53     }
54     if (num_buchi > 0)
55         return tot/num_buchi;
56     return 0;
57 }
58
59 uint16_t stat_out_of_order(pkt_info ord[] , uint16_t not_ord[] , int
60     dim){
61     int i = 0, j = 0;
62     uint16_t out = 0, prec , curr; //prec contiene l'ids corrente
63     for (i = 0; i < dim; i++) {
64         if (ord[i].ids != not_ord[i]) {

```

```

64         out++;
65         prec = not_ord[i];
66         j = i + 1;
67         while (not_ord[j] != ord[i].ids) { // shift a sinistra
68             finch non si trova l'elemento corrispondente
69             curr = not_ord[j];
70             not_ord[j] = prec;
71             j++;
72             prec = curr;
73         }
74         not_ord[i] = ord[i].ids;
75         not_ord[j] = prec;
76     }
77     return out;
78 }
79
80 double jitter_calculator(pkt_info el[], int dim){
81     pkt_info infj, infi; // variabili di lavoro per rendere più
82     leggibile il calcolo del jitter
83     double d_value = 0, jitter = 0;
84     for(int j = 1; j < dim; j++){
85         infj = el[j];
86         infi = el[j-1];
87         d_value = fabs((infj.r_timestamp - infj.pkt_timestamp) - (infi
88         .r_timestamp - infi.pkt_timestamp));
89         jitter = jitter + (d_value - jitter)/JITTER_DIV;
90     }
91     return jitter;
92 }
93
94 void send_to_server(int sockfd, send_stat spedisci[]){
95     writen(sockfd, spedisci, sizeof(send_stat)*num_list);
96 }
97
98 rtp* delete_and_get_next(rtp* el, uint16_t* from, int end_seq){
99     while (!el->sent); // aspetto che il pacchetto stato spedito
100    delete_hash(from);
101    el = NULL;
102    while (end_seq > *from && !el) {
103        (*from)++;
104        el = find_hash(from);
105    }
106    return el;
107 }
```

```

106
107 void fill_lost_packet(uint16_t* prec, uint16_t* from, rtp* el,
108 gop_info* info){
109     if (!el)
110         return;
111
112     while ((*prec)+1 != *from) {
113         info->lost_packet[info->index_losted] = (*prec)+1;
114         info->index_losted++;
115         info->len_losted[info->index_len]++;
116         (*prec)++;
117     }
118     *prec = *from;
119     //Attenzione, assumo che mi arrivi almeno una fine o un
120     //inizio di ogni frame
121     if (el->state == 128) { // state vale 128 se un nuovo frame
122         // 64 se il frame finisce, 0 altrimenti
123         if (info->incremented) // gi stato incrementato
124             precedentemente
125             info->incremented = 0;
126         else
127             info->index_len++;
128     }
129     else if (el->state == 64){
130         info->incremented = 1;
131         info->index_len++;
132     }
133     int writen(int fd, void *buf, size_t size) {
134         size_t left = size;
135         int r=0;
136         char *bufptr = (char*)buf;
137         while(left >0) {
138             r=(int)write(fd ,bufptr ,left );
139             if (r == -1) {
140                 if (errno == EINTR) continue;
141                 if (errno == EPIPE) pthread_kill(segнал, SIGINT);
142                 return -1;
143             }
144             if (r == 0) return 0;
145             left -= r;
146             bufptr += r;

```

```

147     }
148     return 1;
149 }
150
151 // algoritmo preso da https://it.mathworks.com/matlabcentral/
152 // fileexchange/37691-psnr-for-rgb-images
153 //src e dst sono le immagini da confrontare, w e h sono dimensioni
154 // delle due immagini che devono coincidere (non viene controllato
155 //)
156 /*float calculate_PSNR(u_char* src, u_char* dst, int w, int h){
157     return (float)(10.0*log10( pow(255,2) / calculate_MSE(src, dst
158     , w, h))); //255 per le immagini 8 bit
159 }
160
161 float calculate_MSE(u_char* src, u_char* dst, int w, int h){
162     float mseR = sum(sum(mseRImage)) / (w * h);
163     float mseG = sum(sum(mseGImage)) / (w * h);
164     float mseB = sum(sum(mseBImage)) / (w * h);
165
166     return (mseR + mseG + mseB)/3;
167 }*/
168
169 /* funzioni di utilit per interagire con la tabella hash */
170 void* insert_hash(uint16_t primarykey, void* insert){
171     uint16_t* key = setKeyHash(primarykey);
172     int buck = hash_packet->hash_function(key) % hash_packet->
173     nbuckets;
174     int mtxi = buck / DIV;
175     pthread_mutex_lock(&mtxhash[mtx]);
176     void* ris = icl_hash_insert(hash_packet, key, insert);
177     pthread_mutex_unlock(&mtxhash[mtx]);
178     if (!ris)
179         freeKeyHash(key); // si libera key
180     return ris;
181 }
182
183 void* find_hash(uint16_t* primarykey){
184     int buck = hash_packet->hash_function(primarykey) %
185     hash_packet->nbuckets;
186     int mtxi = buck / DIV;
187     pthread_mutex_lock(&mtxhash[mtx]);
188     void* ris = icl_hash_find(hash_packet, primarykey);
189     pthread_mutex_unlock(&mtxhash[mtx]);
190     return ris;
191 }

```

```

186
187 int delete_hash(uint16_t* primarykey){
188     int buck = hash_packet->hash_function(primarykey) %
189     hash_packet->nbuckets;
190     int mtxi = buck / DIV;
191     pthread_mutex_lock(&mtxhash[ mtxi ] );
192     int ris = icl_hash_delete(hash_packet, primarykey, &
193     freeKeyHash, &freeElHash);
194     pthread_mutex_unlock(&mtxhash[ mtxi ] );
195     return ris;
196 }
```

A.1.5 struct_fun.c

```

1 #include "struct_fun.h"
2
3 uint16_t* setkeyHash(uint16_t key){
4     uint16_t* num = malloc(sizeof(uint16_t));
5     *num = key;
6     return num;
7 }
8
9 stat_t* setStat(){
10    stat_t* new = malloc(sizeof(stat_t));
11    memset(new, 0, sizeof(stat_t));
12    return new;
13 }
14
15 gop_info* setElGOP(int gop_num, int start){
16    gop_info* new = malloc(sizeof(gop_info));
17    memset(new, '\0', sizeof(gop_info));
18    new->gop_num = gop_num;
19    new->start_seq = start;
20    new->num_frame = 0;
21    new->end_seq = 0;
22    new->gop_over = -1;
23    new->metadata_start = INT_MAX;
24    new->next_metadata = INT_MAX;
25    return new;
26 }
27
28
29 gop_info* setNextElGOP(int gop_num, int start, int last_frame_gop,
30                         int start_new_gop, int new_metadata, int accept){
31     gop_info* new = malloc(sizeof(gop_info));
```

```

31     memset(new, '\0', sizeof(gop_info));
32     new->gop_over = -1;
33     new->gop_num = gop_num;
34     new->start_seq = start;
35     new->end_last_gop = last_frame_gop;
36     new->start_seq = start_new_gop;
37     new->accepted28 = start_new_gop;
38     new->metadata_start = new_metadata;
39     new->next_metadata = INT_MAX;
40     new->incremented = 1;
41     return new;
42 }
43
44 rtp* setElRTP(u_char* buf, int size, double timestamp, int npkt,
45   int from_thr){
46     rtp* new = malloc(sizeof(rtp));
47     memset(new, '\0', sizeof(rtp));
48     new->packet = buf;
49     new->size = size;
50     new->r_timestamp = timestamp;
51     new->n_pkt = npkt;
52     new->from_thr = from_thr;
53     new->slice_type = 0;
54     new->state = 0;
55     new->decoder = 0;
56     new->sent = 0;
57     return new;
58 }
59 void freeGOP(void** el){
60     gop_info* info = (gop_info*) (*el);
61     free(info);
62     info = NULL;
63 }
64
65 void freeORD(void** el){
66     //rtp* todestroy = (rtp*) (*el);
67     //free(todestroy->packet);
68     //free(todestroy);
69     //todestroy = NULL;
70 }
71
72 void freeRTP(void** el){
73     rtp* todestroy = (rtp*) (*el);
74     free(todestroy->packet);

```

```

75     free(todestroy);
76     todestroy = NULL;
77 }
78
79 void freeStat(void** el){
80     stat_t* stat = (stat_t*) (*el);
81     free(stat);
82     stat = NULL;
83 }
84
85 void freeKeyHash(void* el){
86     int* todestroy = (int*) el;
87     free(todestroy);
88     todestroy = NULL;
89 }
90
91 void freeElHash(void* el){
92     freeRTP(&el);
93 }
```

A.1.6 receiver.c

```

1 #include "receiver.h"
2
3 int initialized = 0; //ci dice se stato trovato SPS e PPS
4 int num_pkt = 0; //numero dei pacchetti arrivati
5 int num_gop = 0; //numero di GOP trovati
6 int current_seq = 0; /* ultimo numero di sequenza letto pi grande */
7 int start_gop = 0; /* inizio di un nuovo gop */
8 int end_new_gop = 0; /* aggiorna quando finisce il nuovo gop */
9 int to_iphdr = sizeof(struct ether_header);
10 int to_udphdr = sizeof(struct ether_header) + sizeof(sniff_ip_t);
11 int to_rtphdr = sizeof(struct ether_header) + sizeof(sniff_ip_t) +
    sizeof(struct udphdr);
12 int to_rtpdata = sizeof(struct ether_header) + sizeof(sniff_ip_t)
    + sizeof(struct udphdr) + sizeof(rtphdr);
13
14 void loopbackstarter(u_char* buff){
15     int zero = 0, due = 2;
16     memcpy(buff, &due, 1);
17     memcpy(buff+1, &zero, 1);
18     memcpy(buff+2, &zero, 1);
19     memcpy(buff+3, &zero, 1);
20 }
```

```

21
22 void starter(string* stringa){
23     int zero = 0, uno = 1;
24     memcpy(stringa->value+stringa->size , &zero , 1);
25     memcpy(stringa->value+stringa->size+1, &zero , 1);
26     memcpy(stringa->value+stringa->size+2, &uno , 1);
27     stringa->size += 3;
28 }
29
30 void add(string* dst, u_char src[], int src_len){
31     memcpy(dst->value+dst->size , src , src_len);
32     dst->size += src_len;
33 }
34
35 void send_packet(rtp* el){
36     struct udphdr* udp = NULL;
37     u_char* send = NULL;
38     int fix = 0, new_size = 0;
39     if (from_loopback && datalink_loopback == DLT_NULL)
40         fix = sizeof(struct ether_header) - 4;
41     udp = (struct udphdr*) (el->packet + to_udphdr - fix);
42     udp->uh_dport = htons(video_port);
43     udp->uh_sum = 0;
44     send = (el->packet + to_rtphdr - fix);
45     new_size = el->size - to_rtphdr + fix - DIM_TIMESTAMP; // 
DIM_TIMESTAMP va a togliere il timestamp aggiunto per calcolare
il delay
46     if (sendto(fd, send, new_size, 0, (struct sockaddr*)&servaddr,
47               sizeof(servaddr)) < 0)
48         perror("cannot send packet");
49     el->sent = 1;
50 }
51
52 void stat_calc(rtp* el, int index, uint16_t rtp_id, double
pkt_timestamp){
53     stat_t* stat = &statistiche[index];
54     if (rtp_id > stat->id_accepted) {
55         stat->pkt_information[stat->index].ids = rtp_id;
56         stat->pkt_information[stat->index].pkt_timestamp =
pkt_timestamp;
57         stat->pkt_information[stat->index].r_timestamp = el->
r_timestamp;
58         stat->index++;
59         if (stat->max < rtp_id)
                stat->max = rtp_id;

```

```

60         if (stat->min > rtp_id)
61             stat->min = rtp_id;
62     }
63 }
64
65 int addPacketToGOP(u_char* rtpdata, int rtpdata_len, uint16_t
seq_num, rtp* el){
66     int return_value = 0;
67     //Nal header
68     unsigned int start_bit = rtpdata[1] & 0x80; // 128 se e' il
primo pacchetto del frame 0 altrimenti
69     unsigned int end_bit = rtpdata[1] & 0x40; // 64 se e' l
ultimo pacchetto 0 altrimenti
70     unsigned int reserved = rtpdata[1] & 0x20 >> 5;
71     unsigned int nal_type = rtpdata[1] & 0x1F;
72     el->slice_type = nal_type;
73     el->state = start_bit == 128 ? start_bit : end_bit;
74     if (nal_type == 5){ //nuovo gop, devo essere sicuro che si
riferisce al gop corrente
75         pthread_mutex_lock(&mtx_info);
76         if (info) {
77             if (info->gop_over == -1) { //se questa condizione
falsa qualche altro thread ha già inserito tutte le
informazioni utili per questo gop, quindi pronto all'
utilizzo
78                 if (seq_num < info->start_seq && seq_num > info->
end_last_gop) {
79                     info->start_seq = seq_num;
80                 }
81                 else if (info->after_5 && seq_num > info->startP){
82                     info->gop_over = info->end_seq;
83                     info->end_seq = seq_num;
84                     return_value = seq_num;
85                 }
86             }
87             else { //ultimi aggiornamenti dati prima di spedire al
decodificatore
88                 if (info->start_seq > seq_num && info->
end_last_gop < seq_num) {
89                     info->start_seq = seq_num;
90                 }
91                 else if (seq_num > info->gop_over && seq_num <
info->end_seq){
92                     info->end_seq = seq_num;
93                 }

```

```

94         }
95     }
96     pthread_mutex_unlock(&mtx_info);
97 }
98 else{ //nal type == 1
99     pthread_mutex_lock(&mtx_info);
100    if (info) {
101        if (!info->after_5 && seq_num > info->accepted28) {
102            info->after_5 = 1;
103            info->startP = seq_num;
104            info->end_seq = seq_num;
105        }
106        if (info->gop_over == -1 && info->end_seq < seq_num) // se gop over != 0 inutile aggiornare end seq, tanto il
decoder controlla gop over per fermarsi
107            info->end_seq = seq_num;
108        pthread_mutex_unlock(&mtx_info);
109    }
110 }
111 if (start_bit == 128){
112     /* viene fatto dopo ogni marker */
113     unsigned int idr_nal = rtpdata[0] & 0xE0; // 3 NAL UNIT
BITS
114     unsigned int nal = idr_nal | nal_type; // [ 3 NAL UNIT
BITS | 5 FRAGMENT TYPE BITS] 8 bits
115     el->decoder = nal;
116 }
117 //}
118 //info->num_frame++; // solo per dati statistici
119 //if (return_value == 0){
120 //    pthread_mutex_lock(&mtx_info);
121 //    info->end_seq = info->end_seq > seq_num ? info->end_seq
: seq_num;
122 //    pthread_mutex_unlock(&mtx_info);
123 //}
124 return return_value;
125 }

126
127 int workOnPacket(rtp* el, int stat_index){
128     int ris = 0;
129     int fix = 0;
130     if (from_loopback && datalink_loopback == DLT_NULL) {
131         fix = (int) sizeof(struct ether_header) - 4;
132     }
133     int rtpdata_len = el->size - to_rtpdata + fix;

```

```

134     rtphdr* rtpHeader = (rtphdr*)(el->packet + to_rtpadr - fix);
135     uint16_t seq = ntohs(rtpHeader->seq_num);
136     double timestamp = *(double*) (el->packet + el->size -
137     DIM_TIMESTAMP);
138     //modifica statistiche
139     pthread_mutex_lock(&mtxstat[el->from_thr]);
140     stat_calc(el, el->from_thr, seq, timestamp);
141     pthread_mutex_unlock(&mtxstat[el->from_thr]);
142     //printf("RTP number [%d], timestamp of this packet is: %d\n",
143     ntohs(rtpHeader->seq_num), ntohs(rtpHeader->TS)); //function
144     //converts the unsigned short integer netshort from network byte
145     //order to host byte order.
146     u_char* rtpdata = (u_char*) (el->packet + to_rtpdata - fix);
147     //FU - A - HEADER
148     //unsigned int forbidden = rtpdata[0] & 0x80 >> 7;
149     //unsigned int nri = rtpdata[0] & 0x60 >> 5;
150     unsigned int fragment_type = rtpdata[0] & 0x1F; // il valore
151     28
152
153     pthread_mutex_lock(&mtx_info);
154     if (info && seq <= info->accept_packet)
155         ris = -1;
156     pthread_mutex_unlock(&mtx_info);
157
158     if (!ris && insert_hash(seq, el)){ //elemento già inserito
159         ritorna
160         if (fragment_type == 28) //dati per il GOP
161             ris = addPacketToGOP(rtpdata, rtpdata_len, seq, el);
162         else{
163             el->slice_type = fragment_type; //in questo caso sto
164             memorizzando il type del NAL header, occhio!
165             pthread_mutex_lock(&mtx_info);
166             if (info->metadata_start > seq && seq < info->
167             start_seq)
168                 info->metadata_start = seq;
169             else if (seq > info->start_seq && seq < info->
170             next_metadata)
171                 info->next_metadata = seq;
172             pthread_mutex_unlock(&mtx_info);
173         }
174         send_packet(el);
175     } else ris = -1;
176     return ris;
177 }
178
179

```

```

170 rtp* set_header(rtp* el, uint16_t* from, int fix, int end_seq){
171     int rtpdata_len = el->size - to_rtpdata + fix;
172     u_char* rtpdata = (u_char*) (el->packet + to_rtpdata - fix);
173
174     starter(&payload);
175     add(&payload, rtpdata, rtpdata_len-DIM_TIMESTAMP);
176
177     while (!el->sent); //aspetto che il pacchetto stato spedito
178     return delete_and_get_next(el, from, end_seq); //definita in
179     utility_fun.h
180 }
181
182 rtp* save_packet(FILE* f, rtp* el, uint16_t* from, int fix, int
183 end_seq){
184     int rtpdata_len = el->size - to_rtpdata + fix;
185     u_char* rtpdata = (u_char*) (el->packet + to_rtpdata - fix);
186     if (el->state == 128) {
187         starter(&payload);
188         memcpy(payload.value+payload.size, &(el->decoder), 1); //header per il frame
189         payload.size += 1;
190     }
191     add(&payload, rtpdata+2, rtpdata_len-2-DIM_TIMESTAMP);
192     fwrite(payload.value, 1, payload.size, f);
193     payload.size = 0;
194     return delete_and_get_next(el, from, end_seq);
195 }
196
197 void save_GOP(uint16_t *from, gop_info* info){
198     FILE* f = NULL;
199     char GOPName[64];
200     int num_hdr = 0;
201     int fix = 0;
202     uint16_t prec = 0; //prec mi serve per sapere quali pacchetti
203     non sono arrivati
204     if (from_loopback && datalink_loopback == DLT_NULL)
205         fix = (int) sizeof(struct ether_header) - 4;
206     sprintf(GOPName, "%sgop-%06d", path_file, info->gop_num);
207     if (!(f = fopen(GOPName, "w")))
208         printf("Error: %s\n", strerror(errno));
209     rtp* el = find_hash(from);
210     //add(&payload, metadata.value, metadata.size); //se un giorno
211     si decidesse di usare questa soluzione, solo un fwrite alla
212     fine
213     while (el && el->slice_type != 5){ // cicla finch i

```

```

pacchetti di header non finiscono
    el = set_header(el, from, fix, info->start_seq);
    num_hdr++;
}
if (num_hdr==2) { // sps e pps presenti
    add(&metadata, payload.value, payload.size);
    payload.size = 0;
}
fwrite(metadata.value, 1, metadata.size, f);
prec = *from - 1;
fill_lost_packet(&prec, from, el, info);
while(el && el->slice_type == 5){ // copia tutto il pacchetto
header
    el = save_packet(f, el, from, fix, info->end_seq);
    fill_lost_packet(&prec, from, el, info);
}
while (el && el->slice_type < 5) { //copia finch non trova
un nuovo GOP
    el = save_packet(f, el, from, fix, info->end_seq);
    fill_lost_packet(&prec, from, el, info);
}
//fwrite(payload.value, 1, payload.size, f);
//payload.size = 0;
fclose(f);
}

```

A.1.7 utility_fun.c

```

1 #include "h264toPng.h"
2
3 //AVCodecContext *pngContext = NULL;
4 //struct SwsContext* img_convert_ctx = NULL;
5
6 void logging(const char *fmt, ...){
7     va_list args;
8     fprintf(stderr, "LOG: ");
9     va_start(args, fmt);
10    vfprintf(stderr, fmt, args);
11    va_end(args);
12    fprintf(stderr, "\n");
13 }
14
15 void plot_value(char path_r[], int gop_num, int FrameNo, int op,
gop_info* info){
16     int sum = 0, i = 0, j = 0;

```

```

17     char PNGNameSender[128];
18     sprintf(PNGNameSender, "%sframe-%06d-%06d.png",
19             path_image_sender, gop_num, FrameNo);
20     if (op)
21         path_r = PNGNameSender;
22     pthread_mutex_lock(&mtxplot);
23     fprintf(pipe_plot, "%s %s %d %d", path_r, PNGNameSender,
24             gop_num, FrameNo);
25     //vado all'indice da utilizzare di losted
26     for (i=0; i<FrameNo-1; i++)
27         sum+=info->len_losted[i];
28     for (j=0; j<info->len_losted[FrameNo-1]; j++)
29         fprintf(pipe_plot, " %d", info->losted_packet[sum+j]);
30     fprintf(pipe_plot, "\n");
31     fflush(pipe_plot);
32     pthread_mutex_unlock(&mtxplot);
33 }
34
35 void savePNG(AVPacket* packet, char PNGName[]) {
36     FILE *PNGFile;
37     if ((PNGFile = fopen(PNGName, "wb"))){
38         fwrite(packet->data, 1, packet->size, PNGFile);
39         fclose(PNGFile);
40     } else printf("Error: %s\n", strerror(errno));
41 }
42
43 int decode_to_png(AVFrame *pFrame, int FrameNo, int gop_num,
44 gop_info* info) {
45     int result = 0;
46     char PNGName[128];
47     // codec per png
48     AVCodecContext *pngContext = NULL;
49     AVCodec *pngCodec = avcodec_find_encoder(AV_CODEC_ID_PNG);
50     if (!pngCodec) {
51         exit(-1);
52     }
53     pngContext = avcodec_alloc_context3(pngCodec);
54     if (!pngContext) {
55         exit(-1);
56     }
57     pngContext->pix_fmt = AV_PIX_FMT_RGB24;
58     pngContext->time_base = (AVRational) { .num = 25, .den = 1};
59     pngContext->framerate = (AVRational) { .num = 0, .den = 0};
60     pngContext->height = pFrame->height;

```

```

59     pngContext->width = pFrame->width;
60     //pngContext->thread_count = 0;
61     //pngContext->thread_type = FF_THREAD_FRAME;
62     if ((result = avcodec_open2(pngContext, pngCodec, NULL)) < 0)
63     {
64         printf("%s\n", av_err2str(result));
65         exit(-1);
66     }
67     int response = avcodec_send_frame(pngContext, pFrame);
68     if (response < 0) {
69         logging("Error while sending a packet to the decoder: %s",
70                av_err2str(response));
71         return response;
72     }
73     AVPacket* packet = av_packet_alloc();
74     if (!packet){
75         logging("failed to allocated memory for AVPacket");
76         return -1;
77     }
78     if (response >= 0) {
79         response = avcodec_receive_packet(pngContext, packet);
80         if (response == AERROR(EAGAIN) || response == AERROR_EOF)
81     }{
82         return (response==AERROR(EAGAIN)) ? 0:1;
83     }
84     else if (response < 0) {
85         fprintf(stderr, "Error during encoding\n");
86         exit(1);
87     }
88     sprintf.PNGName, "%sframe-%06d-%06d.png", path_image,
89     gop_num, FrameNo);
90     savePNG(packet, PNGName);
91     plot_value.PNGName, gop_num, FrameNo, 0, info);
92     av_packet_unref(packet);
93 }
94
95 int pixel_to_rgb24(AVFrame *pFrame, int pix_fmt ,int FrameNo, int
96 gop_num, gop_info* info) {
97     //pFrame=avcodec_alloc_frame();
98     // Allocate an AVFrame structure
99     struct SwsContext* img_convert_ctx = sws_getContext(pFrame->

```

```

width , pFrame->height , pix_fmt , pFrame->width , pFrame->height ,
AV_PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);
99   if (!img_convert_ctx) {
100     fprintf(stderr , "Cannot initialize the conversion context
101           !\n");
102     exit(1);
103   }
104   AVFrame* pFrameRGB=av_frame_alloc();
105   if (pFrameRGB==NULL)
106     return -1;
107
108   int numBytes=avpicture_get_size(AV_PIX_FMT_RGB24, pFrame->
109 width , pFrame->height );
110
111   // Assign appropriate parts of buffer to image planes in
112   // pFrameRGB
113   avpicture_fill((AVPicture *)pFrameRGB, buffer ,
114 AV_PIX_FMT_RGB24, pFrame->width , pFrame->height );
115
116   int ret = sws_scale(img_convert_ctx , pFrame->data , pFrame->
117 linesize , 0, pFrame->height , pFrameRGB->data , pFrameRGB->
118 linesize );
119   if (ret != pFrame->height ) {
120     fprintf(stderr , "SWS_Scale failed [%d]!\n" , ret );
121     exit(-1);
122   }
123   // Save the frame to disk
124   pFrameRGB->format = pFrame->format ;
125   pFrameRGB->height = pFrame->height ;
126   pFrameRGB->width = pFrame->width ;
127   pFrameRGB->pts = FrameNo;
128   //SaveFrame(pFrameRGB, gop_num, FrameNo); // se si volesse
129   //salvarli in formato ppm
130   decode_to_png(pFrameRGB, FrameNo, gop_num, info );
131   av_frame_free(&pFrameRGB);
132   free(buffer );
133   sws_freeContext(img_convert_ctx );
134   return 0;
135 }
136
137 int decode_packet(AVPacket *pPacket , AVCodecContext *pCodecContext
138 , int gop_num, gop_info* info){
139   // https://ffmpeg.org/doxygen/trunk/structAVFrame.html

```

```

134 AVFrame *pFrame = av_frame_alloc();
135 if (!pFrame){
136     logging(" failed to allocated memory for AVFrame");
137     return -1;
138 }
139 pCodecContext->time_base = (AVRational) {1, 50};
140 // Supply raw packet data as input to a decoder
141 // https://ffmpeg.org/doxygen/trunk/group__lavc__decoding.html
#ga58bc4bf1e0ac59e27362597e467efff3
142 int response = avcodec_send_packet(pCodecContext, pPacket);
143 //output(" nome");
144 if (response < 0) {
145     logging("Error while sending a packet to the decoder: %s",
av_err2str(response));
146     //send data to plot
147     plot_value("", gop_num, pCodecContext->frame_number, 1,
NULL);
148     return response;
149 }
150 while (avcodec_receive_frame(pCodecContext, pFrame) >= 0 ){
151     // Return decoded output data (into a frame) from a
decoder
152     // https://ffmpeg.org/doxygen/trunk/group__lavc__decoding.html#ga11e6542c4e66d3028668788a1a74217c
153     logging(
154         "Frame %d (type=%c, size=%d bytes) pts %d
key_frame %d [DTS %d]" ,
155         pCodecContext->frame_number,
156         av_get_picture_type_char(pFrame->pict_type),
157         pFrame->pkt_size,
158         pFrame->pts,
159         pFrame->key_frame,
160         pFrame->coded_picture_number);
161     //decode_to_png(pCodecContext, pFrame, pCodecContext->
frame_number);
162     pixel_to_rgb24(pFrame, pCodecContext->pix_fmt,
pCodecContext->frame_number, gop_num, info);
163     //pthread_mutex_lock(&mtx);
164     //pushList(&testa, &coda, pFrame, pCodecContext->
frame_number);
165     //pthread_cond_signal(&cond);
166     //pthread_mutex_unlock(&mtx);
167     av_frame_unref(pFrame);
168 }
169 av_frame_free(&pFrame);

```

```

170     return 0;
171 }
172
173 int create_image(gop_info* info){
174     int result; //variabile per gli errori
175     //pthread_t decode[NUMDECODERTHR];
176     //printf("creando i thread\n");
177     //for (int i = 0; i < NUMDECODERTHR; i++)
178     //    SYSFREE(result ,pthread_create(&decode[ i ],NULL,&decoder ,
179     //NULL) ,0,"thread")
180     //printf("thread creati...\n");
181     AVFormatContext *c = NULL;
182     char GOPName[64];
183     sprintf(GOPName, "%sgop-%06d", path_file, info->gop_num);
184     logging("decodificando il file %s\n", GOPName);
185     result = avformat_open_input(&c, GOPName, NULL, NULL);
186     if ( result != 0){
187         logging("Cannot open file");
188         printf("%s\n", av_err2str(result));
189         return -1;
190     }
191
192     printf("format %s, duration %lld us, bit_rate %lld\n",
193           c->iformat->name, c->duration, c->bit_rate);
194
195     if ( avformat_find_stream_info(c, NULL) < 0) {
196         logging("ERROR could not get the stream info");
197         return -1;
198     }
199
200     // the component that knows how to enCode and DECode the
201     // stream
202     // it's the codec (audio or video)
203     // http://ffmpeg.org/doxygen/trunk/structAVCodec.html
204     AVCodec *pCodec = NULL;
205     // this component describes the properties of a codec used by
206     // the stream i
207     // https://ffmpeg.org/doxygen/trunk/structAVCodecParameters.html
208     AVCodecParameters *pCodecParameters = NULL;
209     int video_stream_index = -1;
210
211     for (int i = 0; i < c->nb_streams; i++){
212         AVCodecParameters *pLocalCodecParameters = NULL;
213         pLocalCodecParameters = c->streams[ i ]->codecpar;

```

```

210     logging(”AVStream->time_base before open coded %d/%d”, c->
211 streams[ i]->time_base.num, c->streams[ i]->time_base.den);
212     logging(”AVStream->r_frame_rate before open coded %d/%d”,
213 c->streams[ i]->r_frame_rate.num, c->streams[ i]->r_frame_rate.
214 den);
215     logging(”AVStream->start_time %” PRId64, c->streams[ i]->
216 start_time);
217     logging(”AVStream->duration %” PRId64, c->streams[ i]->
218 duration);

219     logging(”finding the proper decoder (CODEC)”);

220     AVCodec *pLocalCodec = NULL;

221     // finds the registered decoder for a codec ID
222     // https://ffmpeg.org/doxygen/trunk/group__lavc__decoding.
223     html#ga19a0ca553277f019dd5b0fec6e1f9dca
224     pLocalCodec = avcodec_find_decoder(pLocalCodecParameters->
225 codec_id);

226     if (pLocalCodec==NULL) {
227         logging(”ERROR unsupported codec!”);
228         return -1;
229     }

230     // when the stream is a video we store its index , codec
231     // parameters and codec
232     if (pLocalCodecParameters->codec_type ==
233 AVMEDIA_TYPE_VIDEO) {
234         if (video_stream_index == -1) {
235             video_stream_index = i;
236             pCodec = pLocalCodec;
237             pCodecParameters = pLocalCodecParameters;
238         }
239
240         logging(”Video Codec: resolution %d x %d”,
241 pLocalCodecParameters->width, pLocalCodecParameters->height);
242     } else if (pLocalCodecParameters->codec_type ==
243 AVMEDIA_TYPE_AUDIO) {
244         logging(”Audio Codec: %d channels , sample rate %d”,
245 pLocalCodecParameters->channels, pLocalCodecParameters->
246 sample_rate);
247     }
248
249     // print its name, id and bitrate

```

```

242     logging("\tCodec %s ID %d bit_rate %lld", pLocalCodec->
243         name, pLocalCodec->id, pLocalCodecParameters->bit_rate);
244     }
245     // https://ffmpeg.org/doxygen/trunk/structAVCodecContext.html
246     AVCodecContext *pCodecContext = avcodec_alloc_context3(pCodec)
247     ;
248     if (!pCodecContext){
249         logging("failed to allocated memory for AVCodecContext");
250         return -1;
251     }
252     // Fill the codec context based on the values from the
253     // supplied codec parameters
254     // https://ffmpeg.org/doxygen/trunk/group__lavc__core.html#gac7b282f51540ca7a99416a3ba6ee0d16
255     if (avcodec_parameters_to_context(pCodecContext,
256         pCodecParameters) < 0)
257     {
258         logging("failed to copy codec params to codec context");
259         return -1;
260     }
261     // Initialize the AVCodecContext to use the given AVCCodec.
262     // https://ffmpeg.org/doxygen/trunk/group__lavc__core.html#ga11f785a188d7d9df71621001465b0f1d
263     if (avcodec_open2(pCodecContext, pCodec, NULL) < 0){
264         logging("failed to open codec through avcodec_open2");
265         return -1;
266     }
267     //AVPacket **pPacket = NULL;
268     //pPacket = malloc(sizeof(APacket*)*gop_info.num_frame);
269     // https://ffmpeg.org/doxygen/trunk/structAVPacket.html
270     //for (int i = 0; i < gop_info.num_frame; i++) {
271     //    pPacket[i] = av_packet_alloc();
272     //    if (!pPacket){
273     //        logging("failed to allocated memory for AVPacket");
274     //        return -1;
275     //    }
276     //}
277     AVPacket* pPacket = av_packet_alloc();
278     if (!pPacket){
279         logging("failed to allocated memory for AVPacket");
280         return -1;
281     }

```

```

281
282     int ret = 0;
283     // fill the Packet with data from the Stream
284     // https://ffmpeg.org/doxygen/trunk/group__lavf__decoding.html
285     #ga4fdb3084415a82e3810de6ee60e46a61
286     while ((ret = av_read_frame(c, pPacket)) >= 0) {
287         if (pPacket->stream_index == video_stream_index)
288             decode_packet(pPacket, pCodecContext, info->gop_num,
289             info);
290             // https://ffmpeg.org/doxygen/trunk/group__lavc__packet.
291             html#ga63d5a489b419bd5d45cf09091cbc2
292             if (pPacket)
293                 av_packet_unref(pPacket);
294
295         }
296         pthread_mutex_lock(&mtxplot);
297         fprintf(pipe_plot, "Fine %d\n", info->gop_num);
298         fflush(pipe_plot);
299         pthread_mutex_unlock(&mtxplot);
300
301         logging("releasing all the resources");
302
303         avformat_close_input(&c);
304         //for (int i = 0; i < gop_info.num_frame; i++)
305         //av_packet_free(&pPacket[i]);
306         av_packet_free(&pPacket);
307         //free(pPacket);
308         avcodec_free_context(&pCodecContext);
309
310     return 0;
311 }
```

A.1.8 lista.c

```

1 #include "lista.h"
2
3 void freeList(list** head, list** coda, void(* del)(void** el)){
4     while (*head) {
5         void* el = popList(head, coda);
6         del(&el);
7     }
8 }
9
10 void pushList(list** head, list** coda, void* el){
11     list* new = malloc(sizeof(list));
12     new->next = NULL;
```

```

13     new->el = el;
14     if (*head == NULL){
15         *head = new;
16         *coda = *head;
17     }
18     else{
19         (*coda)->next = new;
20         *coda = new;
21     }
22 }
23
24 void* popList(list** head, list** coda){
25     list* ret = *head;
26     void* el = ret->el;
27     *head = (*head)->next; // se GOPM      troppo piccolo qua ci
28     sar un segfault
29     free(ret);
30     if (*head == NULL)
31         *coda = NULL;
32     return el;
33 }
```

A.1.9 icl_hash.c

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <assert.h>
5
6 #include "icl_hash.h"
7
8 #include <limits.h>
9
10 #define BITS_IN_int ( sizeof(int) * CHAR_BIT )
11 #define THREEQUARTERS ((int)((BITS_IN_int * 3) / 4))
12 #define ONEEIGHTH ((int)(BITS_IN_int / 8))
13 #define HIGH_BITS (~((unsigned int)(~0) >> ONEEIGHTH))
14 /**
15  * A simple string hash.
16  *
17  * An adaptation of Peter Weinberger's (PJW) generic hashing
18  * algorithm based on Allen Holub's version. Accepts a pointer
19  * to a datum to be hashed and returns an unsigned integer.
20  * From: Keith Seymour's proxy library code
21  *
```

```

22 * @param[in] key — the string to be hashed
23 *
24 * @returns the hash index
25 */
26 static unsigned int hash_pjw(void *key) {
27     char *datum = (char *)key;
28     unsigned int hash_value, i;
29
30     if (!datum) return 0;
31
32     for (hash_value = 0; *datum; ++datum) {
33         hash_value = (hash_value << ONE_EIGHTH) + *datum;
34         if (((i = hash_value & HIGH_BITS) != 0)
35             hash_value = (hash_value ^ (i >> THREE_QUARTERS)) & ~
36             HIGH_BITS;
37     }
38     return (hash_value);
39 }
40 //static int string_compare(void *a, void *b) {
41 //    return (strcmp((char *)a, (char *)b) == 0);
42 //}
43 /**
44 * Create a new hash table.
45 *
46 * @param[in] nbuckets — number of buckets to create
47 * @param[in] hash_function — pointer to the hashing function to
48 * be used
49 * @param[in] hash_key_compare — pointer to the hash key
50 * comparison function to
51 * be used
52 *
53 * @returns pointer to new hash table.
54 */
55
56 icl_hash_t *icl_hash_create(int nbuckets, unsigned int (*
57     hash_function)(void *),
58                             int (*hash_key_compare)(void *, void
59     *)) {
60     icl_hash_t *ht;
61     int i;
62
63     ht = (icl_hash_t *)malloc(sizeof(icl_hash_t));
64     if (!ht) return NULL;
65 }
```

```

62     ht->nentries = 0;
63     ht->buckets = (icl_entry_t **)malloc(nbuckets * sizeof(
64         icl_entry_t *));
65     if (!ht->buckets) return NULL;
66
67     ht->nbuckets = nbuckets;
68     for (i = 0; i < ht->nbuckets; i++) ht->buckets[i] = NULL;
69
70     ht->hash_function = hash_function ? hash_function : hash_pjw;
71     ht->hash_key_compare = hash_key_compare ? hash_key_compare :
72         ulong_key_compare;
73
74
75 /**
76 * Search for an entry in a hash table.
77 *
78 * @param ht — the hash table to be searched
79 * @param key — the key of the item to search for
80 *
81 * @returns pointer to the data corresponding to the key.
82 * If the key was not found, returns NULL.
83 */
84
85 void *icl_hash_find(icl_hash_t *ht, void *key) {
86     icl_entry_t *curr;
87     unsigned int hash_val;
88
89     if (!ht || !key) return NULL;
90
91     hash_val = (*ht->hash_function)(key) % ht->nbuckets;
92
93     for (curr = ht->buckets[hash_val]; curr != NULL; curr = curr->
94         next)
95         if (ht->hash_key_compare(curr->key, key)) return (curr->data);
96
97     return NULL;
98 }
99 /**
100 * Insert an item into the hash table.
101 *
102 * @param ht — the hash table
103 * @param key — the key of the new item

```

```

104 * @param data — pointer to the new item's data
105 *
106 * @returns pointer to the new item. Returns NULL on error.
107 */
108
109 icl_entry_t *icl_hash_insert(icl_hash_t *ht, void *key, void *data
110 ) {
111     icl_entry_t *curr;
112     unsigned int hash_val;
113
114     if (!ht || !key) return NULL;
115
116     hash_val = (*ht->hash_function)(key) % ht->nbuckets;
117
118     for (curr = ht->buckets[hash_val]; curr != NULL; curr = curr->
119          next)
120         if (ht->hash_key_compare(curr->key, key)) {
121             return (NULL); /* key already exists */
122         }
123         /* if key was not found */
124         curr = (icl_entry_t *)malloc(sizeof(icl_entry_t));
125         if (!curr) return NULL;
126
127         curr->key = key;
128         curr->data = data;
129         curr->next = ht->buckets[hash_val]; /* add at start */
130
131         ht->buckets[hash_val] = curr;
132         ht->nentries++;
133
134         return curr;
135     }
136 /**
137 * Free one hash table entry located by key (key and data are
138 * freed using
139 * functions).
140 *
141 * @param ht — the hash table to be freed
142 * @param key — the key of the new item
143 * @param free_key — pointer to function that frees the key
144 * @param free_data — pointer to function that frees the data
145 *
146 * @returns 0 on success, -1 on failure.
147 */

```

```

146 int icl_hash_delete(icl_hash_t *ht, void *key, void (*free_key)(
147     void *),
148             void (*free_data)(void *)) {
149     icl_entry_t *curr, *prev;
150     unsigned int hash_val;
151
152     if (!ht || !key) return -1;
153     hash_val = (*ht->hash_function)(key) % ht->nbuckets;
154
155     prev = NULL;
156     for (curr = ht->buckets[hash_val]; curr != NULL;) {
157         if (ht->hash_key_compare(curr->key, key)) {
158             if (prev == NULL) {
159                 ht->buckets[hash_val] = curr->next;
160             } else {
161                 prev->next = curr->next;
162             }
163             if (*free_key && curr->key) (*free_key)(curr->key);
164             if (*free_data && curr->data) (*free_data)(curr->data);
165             ht->nentries++;
166             free(curr);
167             return 0;
168         }
169         prev = curr;
170         curr = curr->next;
171     }
172 }
173
174 /**
175 * Free hash table structures (key and data are freed using
176 * functions).
177 *
178 * @param ht -- the hash table to be freed
179 * @param free_key -- pointer to function that frees the key
180 * @param free_data -- pointer to function that frees the data
181 *
182 * @returns 0 on success, -1 on failure.
183 */
184 int icl_hash_destroy(icl_hash_t *ht, void (*free_key)(void *),
185                     void (*free_data)(void *)) {
186     icl_entry_t *bucket, *curr, *next;
187     int i;
188
189     if (!ht) return -1;

```

```

189
190     for ( i = 0; i < ht->nbuckets; i++) {
191         bucket = ht->buckets[ i ];
192         for (curr = bucket; curr != NULL;) {
193             next = curr->next;
194             if (*free_key && curr->key) (*free_key)(curr->key);
195             if (*free_data && curr->data) (*free_data)(curr->data);
196             free( curr );
197             curr = next;
198         }
199     }
200
201     if (ht->buckets) free( ht->buckets );
202     if (ht) free( ht );
203
204     return 0;
205 }
206
207 /**
208 * Dump the hash table's contents to the given file pointer .
209 *
210 * @param stream -- the file to which the hash table should be
211 * dumped
212 * @param ht -- the hash table to be dumped
213 * @returns 0 on success , -1 on failure .
214 */
215
216 int icl_hash_dump(FILE *stream , icl_hash_t *ht) {
217     icl_entry_t *bucket , *curr ;
218     int i;
219
220     if (!ht) return -1;
221
222     for ( i = 0; i < ht->nbuckets; i++) {
223         bucket = ht->buckets[ i ];
224         for (curr = bucket; curr != NULL;) {
225             if (curr->key)
226                 fprintf(stream , "icl_hash_dump: %s: %p\n" , (char *)curr->
227                         key ,
228                         curr->data );
229             curr = curr->next;
230         }
231     }

```

```
232     return 0;  
233 }
```

A.1.10 threads.c

```
1 #include "threads.h"  
2  
3 /* funzione handler dei segnali che vengono inviati al thread  
   listener per chiudere*/  
4 void termination_loopback(int signum){  
5     int volatile i = 0;  
6     pthread_t self = pthread_self();  
7     while(i<num_list){  
8         if (self == listener[i]) {  
9             pcap_breakloop(handle[i]);  
10            break;  
11        }  
12        i++;  
13    }  
14 }  
15  
16 //manda la chiusura su una lista  
17 void send_close(list** testa , list** coda , pthread_mutex_t* mtx,  
18                 pthread_cond_t* cond , void(* del)(void** el) , void* el){  
19     pthread_mutex_lock(mtx);  
20     if (del)  
21         freeList(testa , coda , del); // pulisci la liste cos si  
22         sicuri dell' uscita  
23     pushList(testa , coda , el);  
24     pthread_cond_signal(cond);  
25     pthread_mutex_unlock(mtx);  
26 }  
27  
28 void* Segnali(void *arg){  
29     int ris;  
30     printf("Avviato\n");  
31     if(sigwait(&sigset_usr , &ris)!=0){  
32         perror("SIGWAIT");  
33         errno=0;  
34         exit(0);  
35     }  
36     printf("In uscita..\n");  
37     esce = 1;  
38  
39     pthread_kill(stat_thr , SIGALRM); //sveglia thread statistiche
```

```

    per un uscita rapida
38     /* inviare pacchetti fittizzi per essere sicuri di uscire */
39     /* invia chiusura gop thread */
40     for (int i = 0; i < num_list; i++) {
41         pthread_kill(listener[i], SIGALRM); // invia segnali a
        listener per sbloccarlo da pcap loop
42         usleep(3000000); // per dare al tempo ad ogni Thread di
        svegliare il thread
43     }
44     printf("Thread segnali chiuso\n");
45     return (void*) 0;
46 }

47

48 void* statThread(void* arg){
49     int i = 0, j = 0, size, lenght;
50     double delay = 0;
51     struct timespec ts = { 0, stat_interv*1000};
52     stat_t temp; // usato per fare lo swap con statistiche e
        statistiche temp
53     statistiche_ref = malloc(sizeof(stat_t)*num_list); // contiene il riferimento alle vecchie statistiche
54     uint16_t* ids_copy = malloc(sizeof(uint16_t)*DIMARRSTAT); // array temporaneo di lavoro che copia di nuovo gli ids
55     send_stat* spedisci = malloc(sizeof(send_stat)*num_list);
56     for(int i = 0; i<num_list;i++){
57         memset(&statistiche_ref[i], 0, sizeof(stat_t));
58         statistiche_ref[i].pkt_information = malloc(sizeof(
            pkt_info)*DIMARRSTAT);
59         memset(statistiche_ref[i].pkt_information, 0, sizeof(
            pkt_info)*DIMARRSTAT);
60         statistiche_ref[i].min = -1;
61         memset(&spedisci[i], 0, sizeof(send_stat));
62     }
63     memset(ids_copy, 0, sizeof(uint16_t)*DIMARRSTAT);
64     memset(spedisci, 0, sizeof(send_stat)*num_list);
65     int sockfd = set_stat_sock(); //in utility.c
66     while (!esci) {
67         nanosleep(&ts, NULL);
68         // cambia l'array
69         for (i = 0; i < num_list; i++){
70             pthread_mutex_lock(&mtxstat[i]);
71             statistiche_ref[i].id_accepted = statistiche[i].max;
72             temp = statistiche[i];
73             statistiche[i] = statistiche_ref[i];
74             statistiche_ref[i] = temp;

```

```

75     pthread_mutex_unlock(&mtxstat[i]);
76 }
77 //inizio vero calcolo stastiche
78 for (i = 0; i < num_list; i++) {
79     if(statistiche_ref[i].index > 0){
80         temp = statistiche_ref[i];
81         for(j = 0; j < temp.index; j++){
82             ids_copy[j] = temp.pkt_information[j].ids; // copia array
83             delay += (double) temp.pkt_information[j].r_timestamp - temp.pkt_information[j].pkt_timestamp;
84         }
85         //calcolo delay
86         spedisci[i].delay = (double) delay/statistiche_ref[i].index;
87         qsort(statistiche_ref[i].pkt_information,
88               statistiche_ref[i].index, sizeof(pkt_info), cmpfunc); // ordinato
89         //calcolo jitter
90         spedisci[i].jitter = jitter_calculator(temp.
91               pkt_information, temp.index);
92         //calcolo out of order
93         spedisci[i].ordine = stat_out_of_order(temp.
94               pkt_information, ids_copy, statistiche_ref[i].index);
95         //calcolo lunghezza
96         spedisci[i].lunghezza = stat_lunghezza(temp.
97               pkt_information, statistiche_ref[i].index);
98         //calcolo numero di perdite
99         size = statistiche_ref[i].index - 1; // numero di elementi nell'array - 1(l'estremo)
100        lenght = temp.pkt_information[size].ids - temp.
101            pkt_information[0].ids; // max - min
102        spedisci[i].perdita = size > lenght ? size -
103            lenght : lenght - size;
104        spedisci[i].number_of_pkt = statistiche_ref[i].
105            index;
106    }
107    else {
108        spedisci[i].perdita = -1;
109        spedisci[i].delay = -1;
110        spedisci[i].jitter = -1;
111        spedisci[i].lunghezza = -1;
112        spedisci[i].ordine = -1;
113        spedisci[i].number_of_pkt = -1;
114    }
115}

```

```

108         statistiche_ref[i].index = 0;
109         statistiche_ref[i].min = -1;
110         delay = 0;
111     }
112     // spedisci al server
113     send_to_server(sockfd, spedisci);
114 }
115 // close the socket
116 if (sockfd != -1)
117     close(sockfd);
118 for (i=0; i<num_list; i++)
119     free(statistiche_ref[i].pkt_information);
120 free(statistiche_ref);
121 free(ids_copy);
122 free(spedisci);
123 printf("Thread statistiche chiude\n");
124 return (void*) 0;
125 }
126
127 void* DecoderThread( void* arg){
128     printf("Thread decodifica creato\n");
129     int esci = 0;
130     gop_info* el = NULL;
131     while (!escli) {
132         pthread_mutex_lock(&mtx_dec);
133         while (testa_dec == NULL)
134             pthread_cond_wait(&cond_dec, &mtx_dec);
135         el = popList(&testa_dec, &coda_dec);
136         pthread_mutex_unlock(&mtx_dec);
137         if (el->gop_num >= 0)
138             create_image(el);
139         else escli = 1;
140         freeGOP((void**) &el);
141     }
142     printf("Thread decodifica chiude\n");
143     return (void*) 0;
144 }
145
146 void* ReaderPacket( void* arg){
147     printf("Thread per salvare un GOP\n");
148     int esci = 0, ret;
149     gop_info* el = NULL;
150     uint16_t* from = malloc(sizeof(uint16_t));
151     pthread_t decodehandler[num_decoder];
152     for (int i = 0; i < num_decoder; i++)

```

```

153     SYSFREE( ret , pthread_create(&decodehandler[ i ] ,NULL,&
154 DecoderThread ,NULL) ,0 ,”thread” )
155     while ( ! esci ) {
156         pthread_mutex_lock(&mtx_ord );
157         while ( testa_ord == NULL)
158             pthread_cond_wait(&cond_ord , &mtx_ord );
159         el = popList(&testa_ord , &coda_ord );
160         pthread_mutex_unlock(&mtx_ord );
161         if ( el->start_seq != -1 ) {
162             usleep(LFINESTRA); //aspetta altri pacchetti per 200ms
163             pthread_mutex_lock(&mtx_info );
164             if( info )
165                 info->accept_packet = info->start_seq; //aspetta e
166                 aggiorna il last gop, da questo momento non saranno pi
167                 accettati numeri di sequenza del gop precedente
168                 pthread_mutex_unlock(&mtx_info );
169                 *from = el->metadata_start;
170                 save_GOP( from , el );
171                 pthread_mutex_lock(&mtx_dec );
172                 pushList(&testa_dec , &coda_dec , el );
173                 pthread_cond_signal(&cond_dec );
174                 pthread_mutex_unlock(&mtx_dec );
175             }
176             else{
177                 if ( el->gop_over == -1) // esci
178                     esci = 1;
179                 else{ // pacchetti da scartare, inviati senza pps e
180                     sps
181                     *from = el->metadata_start;
182                     rtp* pkt = find_hash( from );
183                     while ( pkt && pkt->slice_type != 5 )
184                         pkt = delete_and_get_next( pkt , from , el->
185 next_metadata-1 );
186                     freeGOP(( void **)&el );
187                 }
188             }
189         }
190         free( from );
191         freeGOP(( void **)&el );
192         //svuota la lista di decodifica
193         pthread_mutex_lock(&mtx_dec );
194         freeList(&testa_dec , &coda_dec , &freeGOP );
195         for ( int i = 0; i < num_decoder; i++ ) {
196             pushList(&testa_dec , &coda_dec , setElGOP(-1, -1));
197             pthread_cond_signal(&cond_dec );

```

```

193     }
194     pthread_mutex_unlock(&mtx_dec);
195     for (int i = 0; i < num_decoder; i++)
196         SYSFREE(ret, pthread_join(decodehandler[i], NULL), 0, "decode
197         1");
198     printf("Thread per salvare il GOP chiude\n");
199     return (void*) 0;
200 }
201 void* GOPThread(void* arg){
202     int num_thr = *(int*) arg, chiudi = 0, count = 0, ret;
203     rtp* el = NULL;
204     gop_info *copy = NULL;
205     printf("Thread che crea GOP creato %d\n", num_thr);
206     while (!chiudi) {
207         pthread_mutex_lock(&mtx_gop);
208         while (testa_gop == NULL)
209             pthread_cond_wait(&cond_gop, &mtx_gop);
210         el = popList(&testa_gop, &coda_gop);
211         pthread_mutex_unlock(&mtx_gop);
212         if (el->packet) {
213             if ((ret = workOnPacket(el, num_thr)) == -1) // attualmente copiato attraverso una alloc, potrei togliere la
214             memcpy
215             freeRTP((void**)&el);
216             else if (ret > 0){ //ret ha il numero di sequenza del
217                 nuovo pacchetto
218                 pthread_mutex_lock(&mtx_info);
219                 if (info) {
220                     copy = info;
221                     info = setNextElGOP(info->gop_num+1, ret, info
222                     ->gop_over, info->end_seq, info->next_metadata, info->start_seq
223                     );
224                     printf("start new gop [%d]\n", info->start_seq
225                     );
226                 }
227                 pthread_mutex_unlock(&mtx_info);
228
229             }
230         }

```

```

231     else {
232         chiudi = 1;
233         freeRTP(( void **) &el);
234     }
235     count++;
236 }
237 /* manda l'ultimo GOP */ //probabilmente andrà tolto
238 pthread_mutex_lock(&mtx_info);
239 copy = info;
240 info = NULL;
241 pthread_mutex_unlock(&mtx_info);

242 if (copy)
243     send_close(&testa_ord, &coda_ord, &mtx_ord, &cond_ord, &
244     freeGOP, copy);
245 printf("Thread che crea GOP esce %d\n", num_thr);
246 return (void *) 0;
247 }

248
249 void* listenerThread(void* arg){
250     //creazione thread che gestisce
251     int ret, *num_thr = (int *) arg;
252     /* registro l'handler per sigusr1*/
253     struct sigaction new_action;
254     new_action.sa_handler = termination_loopback;
255     new_action.sa_flags = 0;
256     ERRORSYSHEANDLER(ret, sigemptyset(&new_action.sa_mask), -1, "NO
257     SIGEMPTY LIST")
258     ERRORSYSHEANDLER(ret, sigaction(SIGALRM, &new_action, NULL), -1,
259     "NO ALARM")
260     /* */
261     pthread_t gophandler;
262     SYSFREE(ret, pthread_create(&gophandler, NULL, &GOPThread, num_thr)
263     , 0, "thread")
264     pcap_loop(handle[*num_thr], -1, sniff, (u_char *) num_thr);
265     printf("Thread listener %d stopped sniffing\n", *num_thr);
266     send_close(&testa_gop, &coda_gop, &mtx_gop, &cond_gop, NULL,
267     setElRTP(NULL, -1, -1, -1, -1));
268     SYSFREE(ret, pthread_join(gophandler, NULL), 0, "gop 1")
269     printf("Thread listener esce %d\n", *num_thr);
270     free(num_thr);
271     return (void *) 0;
272 }

273 void sniff(u_char *user, const struct pcap_pkthdr* pkthdr, const

```

```

271     u_char* packet){
272     num_pkt += 1;
273     //setto gli elementi per poter passare i dati a un altro
274     //thread
275     u_char* buf = malloc( sizeof(u_char)*pkthdr->len );
276     memcpy(buf, packet, pkthdr->len );
277     double time = ((double) pkthdr->ts.tv_sec) + ((double) pkthdr
278     ->ts.tv_usec/1000000);
279     //rtpHeader = (rtpHeader*) (packet + 4 + sizeof(struct ip)
280     //)+ sizeof(struct udphdr));
281     rtp* el= setElRTP(buf, pkthdr->len , time , num_pkt , *(int*)
282     user );
283     pthread_mutex_lock(&mtx_gop);
284     pushList(&testa_gop , &coda_gop , el);
285     pthread_cond_signal(&cond_gop);
286     pthread_mutex_unlock(&mtx_gop);
287 }
```

A.2 Sender

A.2.1 Scheduler.py

```

from scapy.layers.inet import UDP, IP
from scapy.layers.rtp import RTP
from Sender import Sender
from Operatore import Operatore
import Image_Handler
import Statistiche
from scapy.all import *
from Timing import *
from multiprocessing import Process, Queue
import queue
import time
import signal
signal.signal(signal.SIGINT, signal.default_int_handler
)

"""
argv[1] deve contenere il path del file pcap che dovrà
essere letto

```

```

argv[2] Serve per specificare l'ip in cui spedire i
       pacchetti
attualmente specifica l'ip
argv[3] numero di canali
argv[4] la prima porta di un canale, se ci sono pi
       porta i canali saranno numerati come (argv[5], argv
       [5]+1, etc...)
argv[5] Specifica la porta in cui saranno ricevute le
       statistiche
argv[6] Specifica il path in cui saranno salvati i GOP
argv[7] Specifica il path in cui saranno salvati i
       frame
argv[8] Specifica se usare i valori gamma, 1 per usarli
       , 0 altrimenti
... si prosegue come segue per ogni canale
argv[9] Specifica il parametro gamma per decidere se si
       entra nell'evento perdita pacchetti
argv[10] Specifica il parametro beta per decidere se si
       entra nell'evento perdita pacchetti
argv[11] Specifica il parametro gamma per decidere
       quanti pacchetti scartare nell'evento perdita
       pacchetti
argv[12] Specifica il parametro beta per decidere
       quanti pacchetti scartare nell'evento perdita
       pacchetto
argv[13] Specifica il parametro gamma per decidere la
       durata del delay
argv[14] Specifica il parametro beta per decidere la
       durata del delay
"""

```

```

conf.use_pcap = True # permette di usare subito un
                     nuovo socket appena lo creo
nomi_operatori = ['Vodafone', "Tim", "Wind"]
pcap_path, interface, ip_receiver, gop_dir, img_dir =
", "", "", "", ""

```

```

num_porte, porta_inoltro, port_stat, simula = 0, 0, 0,
    0
alpha_evento, scale_evento, alpha_perdita,
    scale_perdita, alpha_delay, scale_delay = [], [],
    [], [], [], []
    
```



```

def inizializza():
    global pcap_path, interface, ip_receiver, gop_dir,
        img_dir, num_porte, porta_inoltro, port_stat,
        simula
    global alpha_evento, scale_evento, alpha_perdita,
        scale_perdita, alpha_delay, scale_delay
    pcap_path = sys.argv[1]
    ip_receiver = sys.argv[2]
    num_porte = int(sys.argv[3])
    porta_inoltro = int(sys.argv[4])
    port_stat = int(sys.argv[5])
    gop_dir = sys.argv[6]
    img_dir = sys.argv[7]
    simula = int(sys.argv[8])
    for j in range(num_porte):
        num_var = (j * 6)
        alpha_evento.append(float(sys.argv[9 + num_var
            ]))
        scale_evento.append(float(sys.argv[10 + num_var
            ]))
        alpha_perdita.append(float(sys.argv[11 +
            num_var]))
        scale_perdita.append(float(sys.argv[12 +
            num_var]))
        alpha_delay.append(float(sys.argv[13 + num_var
            ]))
        scale_delay.append(float(sys.argv[14 + num_var
            ]))
    
```

```

def invio_canali(num_canali, queue_canali, p_num_canali
, p_canale, pkt, index):
    val = random.random()
    if num_canali == 3:
        if val > p_num_canali[0]: # usa un canale
            if val <= p_canale[0]: # canale 1
                queue_canali[0].put((pkt, index))
            elif val <= (p_canale[1]+p_canale[2]): #
                canale 2
                queue_canali[1].put((pkt, index))
            else: # canale 3
                queue_canali[2].put((pkt, index))
        elif val > (p_num_canali[1] - p_num_canali[0]):
            if val <= p_canale[3]: # canale 12
                queue_canali[0].put((pkt, index))
                queue_canali[1].put((pkt, index))
            elif val <= (p_canale[4] + p_canale[5]): #
                canale 13
                queue_canali[0].put((pkt, index))
                queue_canali[2].put((pkt, index))
            else: # canale 23
                queue_canali[1].put((pkt, index))
                queue_canali[2].put((pkt, index))
        else:
            for i in queue_canali:
                i.put((pkt, index))
    else:
        for i in queue_canali:
            i.put((pkt, index))

```

```

# si ottiene una lista che contiene tutti i pacchetti
# che non sono stati spediti da nessun Thread
def not_sended_paket(process_list, queue_list):
    arr = []
    for i in process_list:
        i.join()

```

```

for i in queue_list:
    arr.append(i.get())
new = []
for i in arr[0]: # el per elemento del primo array
    conta = 0
    for x in range(1, 3): # scorri le altre liste
        for val in arr[x]:
            if i >= val:
                if i == val:
                    conta += 1
            else:
                break
    if conta == 2:
        new.append(i)
print(new)

def canale(queue, nome, ip, porta, simulato, alpha_e,
           scale_e, alpha_p, scale_p, alpha_d, scale_d):
    sender = Sender(simulate=simulato, alpha_e=alpha_e,
                     scale_e=scale_e, alpha_p=alpha_p,
                     scale_p=scale_p, alpha_d=alpha_d,
                     scale_d=scale_d)
    operatore = Operatore(sender, ip, porta, nome)
    while True:
        try:
            pkt, index = queue.get()
            if pkt is None:
                break
            operatore.send(pkt, index)
        except KeyboardInterrupt:
            break
    if simulato:
        sender.getQueue().put((False, None))
        sender.getThr().join()
queue.put(operatore.getSender().getNotSent())

```

```

if __name__ == "__main__":
    process_list = []
    queue_list = []
    inizializza()
    simulate = False
    if simula > 0:
        simulate = True
    operatore = Operatore(None, ip_receiver, 0, None)
        # non si spedir mai con questo sender, usato
        # solo per settare un pacchetto
    for i in range(num_porte):
        if i >= len(nomi_operatori): # si potrebbe
            anche eliminare l'array nomi operatori
            name = nomi_operatori[2]
        else:
            name = nomi_operatori[i]
        queue_list.append(Queue())
        process_list.append(Process(target=canale, args
            =(queue_list[i], name, ip_receiver,
            porta_inoltro,
            simulate
            ,
            alpha_ev
            [
            i
            ],
            scale_ev
            [
            i
            ],
            alpha_pe
            [

```

```

        i
    ] , scale_perd
        [
        i
    ] , alpha_de
        [
        i
    ] , scale_de
        [
        i
    ]) )
    )

process_list[i].start()
porta_inoltro += 1

queue_stat = Queue()
queue_gop = Queue()
esci = False
stat_process = Process(target=Statistiche.stat,
    args=((queue_stat, ip_receiver, port_stat,
    num_porte), ))
image_process = Process(target=Image_Handler.
    analyzer, args=((queue_gop, pcap_path, gop_dir,
    img_dir), )) # da usare
image_process.start() # da usare
stat_process.start()
try:
    num_canali, quali = queue_stat.get()
except KeyboardInterrupt:

```

```

        exit(-1)
packets = rdpcap(sys.argv[1], 1)
timer = Timing()
start = time.time()
start_time = 0
pkt_start_time = packets[0].time
#numero_totale_pkt = 0
bind_layers(UDP, RTP)
try:
    with PcapReader(pcap_path) as pcap_reader:
        for index, pkt in enumerate(pcap_reader):
            if IP in pkt and RTP in pkt:
                #numero_totale_pkt = index
                try:
                    num_canali, quali = queue_stat.
                        get(block=False)
                except queue.Empty:
                    pass
                if start_time == 0: # per avere
                    uno start time pi fe de le
                    possibile
                    start_time = time.time()
                send = operatore.set_pkt(pkt)
                timer.nsleep(timer.delay_calculator
                    (pkt.time, pkt_start_time,
                     start_time)*0.85) # in modo da
                     svegliarsi un 15% prima e poter
                     eseguire pi invii senza
                     problemi
                invio_canali(num_porte, queue_list,
                             num_canali, quali, send, index)
except KeyboardInterrupt:
    pass

esci = True
for i in queue_list:
    i.put((None, -1))

```

```

        time.sleep(1)
queue_gop.put(None) # da usare
image_process.join() # da usare
#print(arr)
#print('Numero di elementi non inviati ' + str(len(
#    arr)) + ' su ' + str(numero_totale_pkt) + '
pacchetti')
print("---- %s seconds ----" % (time.time() - start))

```

A.2.2 Timing.py

```

import time
from ctypes import *
import sys
from decimal import Decimal


class Timespec(Structure):
    """ timespec struct for nanosleep , see:
        http://linux.die.net/man/2/nanosleep """
    _fields_ = [ ('tv_sec', c_long),
                ('tv_nsec', c_long) ]

usec = 1000000
nanosec = 1000000000
timespec = Timespec()
timespec.tv_sec = c_long(0)

class Timing:

    windows = False

    def __init__(self):
        if sys.platform != "win32":
            if sys.platform == 'darwin':

```

```

        self.libc = self.libc = CDLL('/usr/lib/
            libc.dylib')
    elif sys.platform == 'linux':
        self.libc = self.libc = CDLL('libc.so.6
            ')
    self.libc.usleep.argtypes = [c_uint32]
    self.libc.nanosleep.argtypes = [POINTER(
        Timespec),
            POINTER(
                Timespec
            )]
else:
    self.windows = True

def delay_calculator(self, current_pkt_time,
    first_pkt_time, start_calculation_time):
    ideale = current_pkt_time - first_pkt_time
    attuale = time.time() - start_calculation_time
    diff = ideale - Decimal(attuale)
    print('ideale: ' + str(ideale) + ' attuale: ' +
        str(attuale) + ' diff: ' + str(diff))
    return float(diff)

"""
il Thread dorme per time millisecondi
@param time: deve essere compreso tra 0 e 10000000
"""

def usleep(self, temp):
    if not self.windows:
        self.libc.usleep(c_uint32(temp*usec))
    else:
        self.__winsleep(time)

"""
il Thread dorme per time nanosecondi
@param nsec1: nanosecondi compresi tra 0 e
    999999999 specifica per quanto tempo il thread
"""

```

```

    deve dormire
"""

def nsleep(self, temp):
    #if not self.windows: # too slow :(
    #    target_time = time.time() + float(temp)
    #    while (target_time - time.time()) > 0.25:
    #        # entra sse ho un grosso vantaggio
    #        temp = target_time - time.time()
    #        timespec.tv_nsec = c_long(int(temp *
    #nanosec)))
    #        self.libc.nanosleep(pointer(timespec),
    #None)
    #else:
    self.__winsleep(temp)

"""

Windows non ha sleep , simulo solo una nsleep in
maniera brutale
"""

def __winsleep(self, time_to_wait):
    target_time = time.time() + float(time_to_wait)
    while time.time() < target_time:
        pass

```

A.2.3 Operatore.py

```

from scapy.all import *
import socket
from scapy.layers.inet import IP, UDP
from scapy.layers.l2 import Ether, Loopback


class Operatore:

"""

@param sender: canale su cui spedire il pacchetto

```

```

@param ip: ip di destinazione su cui spedire i
pacchetti
@param port: porta su cui vengono spediti i
pacchetti
"""

def __init__(self, sender, ip, port, nome):
    if sender is not None:
        self._address = (ip, port)
        self._socket = socket.socket(family=socket.
                                     AF_INET, type=socket.SOCK_DGRAM)
        self._sender = sender
        self._ip = ip
        self._port = port
        self._name = nome

def send(self, pkt, indice):
    self._sender.send(self._socket, self._address,
                      pkt, indice)

def set_pkt(self, pkt):
    return bytes(pkt[UDP].payload)

def setIP(self, dstip):
    self._ip = dstip

def setPort(self, port):
    self._port = port

def getAddress(self):
    return self._address

def getSocket(self):
    return self._socket

def getIP(self):
    return self._ip

```

```

def getPort(self):
    return self._port

def getSender(self):
    return self._sender

def getName(self):
    return self._name

def setName(self, name):
    self._name = name

def __repr__ (self):
    return "<Sender con ip: %s, porta %s>" % (self._ip, self._port)

def __str__ (self):
    return "Sender con ip: %s, porta %s" % (self._ip, self._port)

```

A.2.4 Sender.py

```

import math
import random
import struct
import queue
import time
from threading import Thread
from queue import Empty
from scipy.stats import gamma

class Sender:

    #contiene gli indici dei pacchetti non inviati
    _pkt_losted = []

```

```

"""
@param simulate: True se si simula l'invio su una
    rete reale, False altimenti
@param alpha_e: valore alpha per decidere se si
    deve entrare nell'evento
@param scale_e: valore scale per decidere se si
    deve entrare nell'evento
@param alpha_p: valore alpha per decidere quanti
    pacchetti perdere
@param scale_p: valore scale per decidere quanti
    pacchetti perdere
@param alpha_d: valore alpha per decidere di quanto
    far ritardare un pacchetto
@param scale_d: valore scal per decidere di quanto
    far ritardare un pacchetto
"""

def __init__(self, simulate, alpha_e, scale_e,
            alpha_p, scale_p, alpha_d, scale_d):
    self._simulate = simulate
    if simulate:
        self._alpha_e = alpha_e
        self._scale_e = scale_e
        self._alpha_p = alpha_p
        self._scale_p = scale_p
        self._alpha_d = alpha_d
        self._scale_d = scale_d
        self._delay = gamma.rvs(alpha_d, scale=
                               scale_d, size=1)
        self._evento = gamma.rvs(alpha_e, scale=
                               scale_e, size=1)
        self._perdita = gamma.rvs(alpha_p, scale=
                               scale_p, size=1)
        self._delay_prec = 0
        self._counter = 0 # numero di pacchetti da
                          scartare

```

```

        self._indice = 0 # numero di pacchetti
                         inviati
        self._delay_list = [] # lista in cui
                              verranno inseriti i pacchetti da non
                              spedire subito
        self._queue = queue.Queue() # coda per
                                    inoltrare pacchetti al listThread
        self._listThread = Thread(target=self.
                                   send_delayed, args=(self._delay_list,
                                   self._queue))
        self._listThread.start()

def send(self, socket, address, pkt, indice):
    if self._simulate:
        ret = self.send_simulate(socket, address,
                                 pkt, indice)
        if ret == 0:
            self._delay = gamma.rvs(self._alpha_d,
                                     scale=self._scale_d, size=1000)
            self._evento = gamma.rvs(self._alpha_e,
                                     scale=self._scale_e, size=1000)
            self._perdita = gamma.rvs(self._alpha_p,
                                       scale=self._scale_p, size=1000)
    else: # ci sar un ritardo e una perdita
        reale
        pkt = pkt + (bytarray(struct.pack("d",
                                         time.time())))
    self.send_pkt(socket, address, pkt)

def send_simulate(self, socket, address, pkt,
                  indice):
    index = self._indice % 1000
    if self._counter <= 0: # mi sto chiedendo se
                           non ho pacchetti da scartare
                           rng = random.random()
                           if rng > self._evento[index]:

```

```

        val = self._delay[index] # calcola il
                               delay
        delay = val - self._delay_prec
        if delay < 0:
            delay = 0
        self._delay_prec = delay
        pkt = pkt + (bytearray(struct.pack("d",
                                           time.time())))
        self._queue.put((True, socket, address,
                         pkt, (time.time() + delay))) # il
                                         primo parametro indica al Thread di
                                         continuare a ciclare
    else: # entra nell'evento perdita
        self._counter = math.ceil(self._perdita
                                   [index]) - 1
        self._pkt_lost.append(indice)
        self._indice += 1
    else: # se ho pacchetti da scartare a causa di
          un evento perdita
        self._counter -= 1
        self._pkt_lost.append(indice)
    return index

def send_delayed(self, delay_list, queue):
    temp_min = float('inf') # rappresenta il
                           timestamp del pacchetto con meno delay,
                           inizializzato con un grande va
    cicla = True
    while cicla:
        if not delay_list: # La lista vuota, si
                           attende un pacchetto.
            cicla, temp_min = self.insertList(
                delay_list, queue, True, temp_min)
        if not cicla:
            continue
        while time.time() <= temp_min: # si
            attende fin ch non c' almeno un

```

```

pacchetto da spedire
cicla, temp_min = self.insertList(
    delay_list, queue, False, temp_min)
if not cicla:
    return
temp_min = float('inf')
for i in reversed(self._delay_list): # c'
    almeno un pacchetto da spedire
    socket, addr, pkt, times = i
    if time.time() > times:
        self.send_pkt(socket, addr, pkt)
        self._delay_list.remove(i)
    else: # pu essere aggiornato il
        tempo minimio
        if temp_min > times:
            temp_min = times

def insertList(self, list, queue, blocking,
min_time):
    cicla = True
    try:
        cicla, *i = queue.get(block=blocking)
        if cicla:
            socket, addr, pkt, time = i
            if min_time > time:
                min_time = time
            list.append((socket, addr, pkt, time))
    except Empty:
        pass
    return cicla, min_time

def send_pkt(self, socket, addr, pkt):
    try:
        socket.sendto(pkt, addr)
    except KeyboardInterrupt:
        pass

```

```

def setDelay(self, delay):
    self._delay = delay

def getThr(self):
    return self._listThread

def getQueue(self):
    return self._queue

def getDelay(self):
    return self._delay

def getNotSent(self):
    return self._pkt_losted

def __repr__(self):
    return "<Operatore con nome: %s, ha un delay di %s>" % (self._delay)

def __str__(self):
    return "Nome: %s, ha un delay di %s" % (self._delay)

```

A.2.5 Statistiche.py

```

import csv
import socket
from Scheduler_controller import SchedulerController
from ctypes import *
from datetime import datetime

class Stat(Structure): # Struttura che deve essere identica alla struttura send_stat definita in struct.h
    _fields_ = [("perdita", c_uint16),
               ("lunghezza", c_uint16),

```

```

        ("delay", c_double),
        ("jitter", c_double),
        ("ordine", c_uint16),
        ("num_of_pkt", c_int)]


def stat(args):
    data = datetime.now()
    path = 'statistics/stat/' + str(data) + '.csv'
    queue = args[0]
    ip = args[1]
    port = args[2]
    num_lst = args[3] # per sapere il numero di sender
    statistiche = []
    esci = False
    dim = sizeof(Stat)
    controller = SchedulerController()
    with open(path, 'w+') as f:
        writer = csv.writer(f)
        writer.writerow(["#Perdite", "Lunghezza perdite",
                        "Delay", "Jitter", "Fuori ordine",
                        "Number of packets"])
        with socket.socket(socket.AF_INET, socket.
                           SOCK_STREAM) as s:
            s.bind((ip, port))
            s.listen()
            conn, addr = s.accept()
            with conn:
                print('Connected by', addr)
                queue.put((controller.
                           probabilitaSulNumeroLink(),
                           controller.probabilitaSulLinkInvio()
                           )))
            while not esci:
                try:
                    data = conn.recv(dim*num_lst,
                                     socket.MSG_WAITALL) #

```

aspetto che vengano ricevuti tutti i byte dichiarati non solo il massimo

```

if not data:
    break
for i in range(num_lst):
    test = data[i*dim:(i+1)*dim]
    stat_ = Stat.
        from_buffer_copy(test)
    statistiche.append((stat_.
        perdita/stat_.num_of_pkt
        , stat_.delay))
    writer.writerow([stat_.
        perdita, stat_.lunghezza
        , stat_.delay, stat_.
        jitter, stat_.ordine,
        stat_.num_of_pkt])
    #print("Received perd=%d,
    lungh=%d, delay=%d, ord
    =%d" % (stat_.perdita ,
    stat_.lunghezza , stat_.
    delay , stat_.ordine))
if num_lst == 3:
    loss1, delay1 = statistiche
    [0]
    loss2, delay2 = statistiche
    [1]
    loss3, delay3 = statistiche
    [2]
    controller.setChannels(
        loss1, loss2, loss3,
        delay1, delay2, delay3)
    queue.put((controller.
        probabilitaSulNumeroLink
        (), controller.
        probabilitaSulLinkInvio

```

```

        (()))
writer.writerow(["", "", "", "", ""
                , "", ""])
except KeyboardInterrupt:
    break
print("Esce dalle statistiche")

```

A.2.6 Scheduler_controller.py

```

import numpy as np
from scipy.optimize import minimize as Min
from random import choices
from collections import Counter

class SchedulerController:

    def __init__(self):
        # delay link 1
        self.d1 = 0.11
        # loss link 1
        self.pl1 = 2*10**-3
        # delay link 2
        self.d2 = 0.14
        # loss link 2
        self.pl2 = 3*10**-2
        # delay link 3
        self.d3 = 0.07
        # loss link 3
        self.pl3 = 9*10**-2
        self.calcolaPesi() # calcolo intermedio
        self.probabilitaPerdita1Link() # prob di
            scegliere un canale
        self.second_ch() # calcolo intermedio
#self.optimizedProb()
        self.probabilitaPerdita2Link() # probabilit
            di scegliere 2 canali

```

```

self.probabilitaPerdita3Link() # probabilit
di scegliere 3 canali

def calcolaPesi(self):
    d1, d2, d3 = self.d1, self.d2, self.d3
    # calculate weight 1
    self.p1 = (d1**-1/(d1**-1+d2**-1+d3**-1))
    # calculate weight 2
    self.p2 = (d2**-1/(d1**-1+d2**-1+d3**-1))
    # calculate weight 3
    self.p3 = (d3**-1/(d1**-1+d2**-1+d3**-1))

def second_ch(self):
    # 2 PATH
    p1, p2, p3 = self.p1, self.p2, self.p3
    self.p12 = p2*p1/(1 - p2) + p1*p2/(1 - p1)
    self.p13 = p3*p1/(1 - p3) + p1*p3/(1 - p1)
    self.p23 = p3*p2/(1 - p3) + p2*p3/(1 - p2)

def optimizedProb(self):
    p12, p13, p23 = self.p12, self.p12, self.p23
    self.p1d = (p12 + p13)/((p12 + p13) + (p12 +
        p23) + (p13 + p23))
    self.p2d = (p12 + p23)/((p12 + p13) + (p12 +
        p23) + (p13 + p23))
    self.p3d = (p13 + p23)/((p12 + p13) + (p12 +
        p23) + (p13 + p23))

def probabilitaPerdita1Link(self):
    # 1 PATH
    # resulting packet loss with monopath
    self.plP1 = self.p1 * self.p11 + self.p2 * self
        .p12 + self.p3 * self.p13

def probabilitaPerdita2Link(self):
    # resulting packet loss with doublepath

```

```

    self.plP2 = self.p12 * self.p11 * self.p12 +
                self.p13 * self.p11 * self.p13 + self.p23 *
                self.p13 * self.p12

def probabilitaPerdita3Link(self):
    #1 PATH
    # resulting packet loss with triplepath
    self.plP3 = self.p11 * self.p12 * self.p13

def probabilitaSulNumeroLink(self):
    # multipath loss vector
    return np.array([self.plP1, self.plP2, self.
                    plP3])

def probabilitaSulLinkInvio(self):
    # multipath loss vector
    return np.array([self.p1, self.p2, self.p3,
                    self.p12, self.p13, self.p23])

def setChannels(self, p11, p12, p13, d1, d2, d3):
    self.d1 = d1
    self.p11 = p11
    # delay link 2
    self.d2 = d2
    # loss link 2
    self.p12 = p12
    # delay link 3
    self.d3 = d3
    # loss link 3
    self.p13 = p13
    self.calcolaPesi() # calcolo intermedio
    self.probabilitaPerdita1Link() # prob di
        scegliere un canale
    self.second_ch() # calcolo intermedio
    #self.optimizedProb()
    self.probabilitaPerdita2Link() # probabilit
        di scegliere 2 canali

```

```
self.probabilitaPerdita3Link() # probabilit  
di scegliere 3 canali
```

A.2.7 Image_Handler.py

```
from queue import Empty
from multiprocessing import Process
from scapy.all import *
from scapy.layers.inet import UDP, IP
from scapy.layers.rtp import RTP

to_null = "> /dev/null"

def starter(to_add):
    zero = (0).to_bytes(1, byteorder=sys.byteorder)
    uno = (1).to_bytes(1, byteorder=sys.byteorder)
    to_add += zero + zero + uno

def to_image(args):
    gop_path = args[0]
    img_path = args[1]
    with open(os.devnull, 'w') as fp:
        try:
            subprocess.Popen(["ffmpeg", "-i", gop_path,
                            img_path], stdin=None, stdout=fp,
                            stderr=fp)
        except KeyboardInterrupt:
            pass

def analyzer(args):
    queue = args[0]
    pcap_path = args[1]
    path_gop = args[2]
    path_img = args[3]
    num_frame = 0
```

```

num_gop = 0
payload = bytearray()
metadata = bytearray()
inizialized = False
print(pcap_path)
bind_layers(UDP, RTP)
,,

Devo iniziare a unificare quando ricevo pacchetti
Header(nal_type == 5)
i pacchetti P sono con nal_type == 1
Gli header sono preceduti da pacchetti SPS e PPS
rispettivamente fragment_type == 7 fragment_type
== 8
il formato deve essere
00 00 01 [SPS] 00 00 01 [PPS] 00 00 01 [DATA] in
data bisogna togliere il nal header

Inizio il mio algoritmo (venendo incontro in modo
furbo al pcap di prova)
scarto tutti i pacchetti finche non trovo un sps e
pps
cerco header e attacco idr nal e nal_type, per i
pacchetti header questo valore sempre lo
stesso
inserisco poi il payload togliendo i primi due byte
(contengono gli header)
se incontro poi altri sps e pps li scarto (per lo
stream in analisi sono sempre uguali)
,,

try:
    for pkt in PcapReader(pcap_path):
        try:
            val = queue.get(block=False)
        except Empty:
            pass
        else:
            if val == 'Esci':

```

```

        break

if IP in pkt and RTP in pkt:
    # https://stackoverflow.com/questions/
    #7665217/how-to-process-raw-udp-
    #packets-so-that-they-can-be-decoded-
    #by-a-decoder-filter-i
    #prendo H264 FRAGMENT
    data = pkt[RTP].load

    #FU - A --HEADER
    forbidden = data[0] & 0x80 >> 7
    nri = data[0] & 0x60 >> 5
    fragment_type = data[0] & 0x1F  # spero
        che il valore sia 28

    if not initialized:
        if fragment_type == 7:
            starter(metadata)
            metadata += pkt[Raw].load  #
                SPS informaition
        if fragment_type == 8:
            starter(metadata)
            metadata += pkt[Raw].load  #
                PPS information
            initialized = True
        continue

    #informazioni per lo streaming ottenuto
    , inizio concatenazione
    if fragment_type == 28:
        #Nal header
        start_bit = data[1] & 0x80  # 128
            se e' il primo pacchetto del
            frame 0 altrimenti
        end_bit = data[1] & 0x40  # 64 se e
            ' l ultimo pacchetto 0
            altrimenti

```

```

reserved = data[1] & 0x20 >> 5
nal_type = data[1] & 0x1F

if start_bit == 128:
    if nal_type == 5:
        if len(payload) != 0:
            path = path_gop + 'gop-
                ' + str(num_gop).
            zfill(6)
            img_path = path_img + "
                frame-" + str(
            num_gop).zfill(6) +
            "-%06d.png"
            f = open(path, "wb")
            f.write(payload)
            f.close()
            num_gop += 1
            payload = bytearray()
            Process(target=to_image
                , args=((path,
            img_path),)).start()
            print('start new gop ' +
            str(num_gop))
            payload += metadata
            idr_nal = data[0] & 0xE0 # 3
            NAL UNIT BITS
            nal = idr_nal | nal_type # [ 3
            NAL UNIT BITS | 5 FRAGMENT
            TYPE BITS] 8 bits
            starter(payload)
            payload += nal.to_bytes(1,
            byteorder=sys.byteorder) #
            header per il frame
elif end_bit == 64:
    num_frame += 1 # solo per dati
    statistic

```

```

        payload += data[2:] # rimuove gli
        header del payload e contiene
        solo i dati successivi

except KeyboardInterrupt:
    pass

    , ,
    cap = cv2.VideoCapture(path)
    success, image = cap.read()
    count = 0
    while success:
        cv2.imwrite("/Users/maxuel/Desktop/files/frame%"
                   "d.jpg" % count, image) # save frame as JPEG
        file
        success, image = cap.read()
        print('Read a new frame: ', success)
        count += 1
    , ,
    , ,
    print('gosh')
    jpg_original = base64.b64decode(payload)
    #invImg = cv2.bitwise_not(jpg_original)
    jpg_as_np = np.frombuffer(jpg_original, dtype=np.
                             uint8)
    #invImg = cv2.bitwise_not(jpg_as_np)
    img_str = cv2.imencode('.jpg', jpg_as_np)[1]
    #print(img_str)
    #img = cv2.imdecode(jpg_as_np, flags=1)
    img2 = cv2.imdecode(img_str, cv2.IMREAD_UNCHANGED)
    cv2_img = cv2.cvtColor(img2, cv2.COLOR_RGB2BGR)
    print(img2)
    cv2.imwrite('/Users/maxuel/Desktop/img.jpg', img2)
    #cv2.imshow('jpg', img2)
    #cv2.waitKey(0)
    print('rip')
    #print(i)

```

```

b64_bytes = base64.b64encode(payload)
b64_string = b64_bytes.decode()

# reconstruct image as an numpy array
img = imread(io.BytesIO(base64.b64decode(payload)))

# show image
plt.figure()
plt.imshow(img, cmap="gray")

# finally convert RGB image to BGR for opencv
# and save result
cv2_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
cv2.imwrite("/Users/maxuel/Desktop/img.jpg",
            cv2_img)
plt.show()
,,

Come modificare il pacchetto https://stackoverflow.com/
questions/52111663/python-scapy-rtp-header-
manipulation-how-can-i-decode-rtp-in-scapy
##### un-comment and change lines below to manipulate
headers

```

```
# packet[RTP].version = 0
# packet[RTP].padding = 0
# packet[RTP].extension = 0
# packet[RTP].numsync = 0
# packet[RTP].marker = 0
# packet[RTP].payload_type = 0
# packet[RTP].sequence = 0
# packet[RTP].timestamp = 0
;;;
```

A.3 Bash Script

A.3.1 execute.sh

```
#!/bin/bash
helper="NAME
          Drones Streaming - Esecuzione del programma
          del tirocinio
SYNOPSIS
          helper WHO_EXECUTE SENDER_CONFIGURATION_FILE
          RECEIVER_CONFIGURATION_FILE
DESCRIPTION
          WHO_EXECUTE Intero che identifica che script
          eseguire, in questo modo,
          solo il file di configurazione corrispondente
          verr utilizzato. 0 Per
          utilizzare entrambi, 1 per utilizzare solo il
          sender e 2 per il receiver.
          SENDER_CONFIGURATION_FILE File di
          configurazione che gestisce il sender
          RECEIVER_CONFIGURATION_FILE di configurazione
          che gestisce il receiver
AUTHOR
          Written by Massimo Puddu
          "
if [[ $UID != 0 ]]; then
    echo "Please run this script with sudo"
    echo "sudo $0 $*"
    exit -1
fi
if [ "$#" -ne 3 ]; then
    echo "$helper"
    exit -1
fi

chmod +x Script/sender.sh
if [ ! $? -eq 0 ]; then
```

```

        echo "Failed to give script permission"
        exit -1
    fi
    chmod +x Script/receiver.sh
    if [ ! $? -eq 0 ]; then
        echo "Failed to give script permission"
        exit -1
    fi
    chmod +x Script/all_in_one.sh
    if [ ! $? -eq 0 ]; then
        echo "Failed to give script permission"
        exit -1
    fi

# Attenzione qui ci si aspetta che entrambi i config
# file siano corretti
# e che tutto vada a buon fine senn si dovranno
# arrestare antrambi gli script(o uno dei due se l'
# altro fallisce)
# manualmente inviando un SIGINT, consigliabile
# usarli prima separatamente
if [[ "$1" -eq 0 ]]; then
    Script/all_in_one.sh "$2" "$3"
elif [[ "$1" -eq 1 ]]; then
    Script)sender.sh "$2"
elif [[ "$1" -eq 2 ]]; then
    Script/receiver.sh "$3"
else
    echo "Inserire il primo argomento con un
          valore tra 0 e 2 assicurandosi di
          selezionare il valore opportuno"
fi

```

A.3.2 receiver.sh

```

#!/bin/bash
helper="NAME
          Drones Streaming - Esecuzione del programma

```

```

        del tirocinio

SYNOPSIS

        helper CONFIGURATION_FILE

DESCRIPTION

        CONFIGURATION_FILE File di configurazione che
        contiene i dati in input del receiver

AUTHOR

        Written by Massimo Puddu
        "
if [[ $UID != 0 ]]; then
    echo "Please run this script with sudo"
    echo "sudo $0 $*"
    exit -1
fi
if [ "$#" -ne 1 ]; then
    echo "$helper"
    exit -1
fi
removed_comment=$(grep -o '^[^#]*' "$1")
input[0]=$(echo "$removed_comment" | awk '/^nic/{
    print $3}') #interfaccia di rete in cui inserire e
                sniffare pacchetti
input[1]=$(echo "$removed_comment" | awk '/^sender_IP
/{print $3}') #ip per spedire le statistiche al
                sender
input[2]=$(echo "$removed_comment" | awk '/^
                number_of_links/{print $3}') #numero di canali
input[3]=$(echo "$removed_comment" | awk '/^port/{
    print $3}') #sniffa pacchetti dalla porta x fino
                alla porta x+numero canali
input[4]=$(echo "$removed_comment" | awk '/^
                feedback_port_at_sender/{print $3}') #numero della
                porta delle statistiche
input[5]=$(echo "$removed_comment" | awk '/^
                video_port/{print $3}') #numero della porta per
                inoltrare i pacchetti al player
input[6]=$(echo "$removed_comment" | awk '/^

```

```

GOP_folder/{print $3}') #cartella dei gop lato
receiver
input[7]=$(echo "$removed_comment" | awk '/^VF_folder
/{print $3}') #cartella delle immagini lato
receiver
input[8]=$(echo "$removed_comment" | awk '/^
original_VF_folder/{print $3}') #cartella immagini
sender
input[9]=$(echo "$removed_comment" | awk '/^
window_length/{print $3}') #lunghezza della
finestra delle statistiche
input[10]=$(echo "$removed_comment" | awk '/^
decoding_threads/{print $3}') #numero dei thread
del decoder

for i in "${input[@]}"
do
    if [[ -z "$i" ]]; then #controlla che non ci
        siano elementi vuoti
        echo "config file non corretto"
        exit -1
    fi
done

for i in {6..8}
do
    if [ ! -d "${input[$i]}" ]; then # controlla
        che la directory non esista, se non
        esistono le crea
        if ! mkdir -p "${input[$i]}"; then
            echo "Impossibile creare la
            cartella gop lato sender"
            exit -1
        fi
    fi
    length=${#input[i]}
    last_char=${input[i]:length-1:1}

```

```

[[ $last_char != "/" ]] && input[i]="${input[i]}/";
done

cd ImagePicker
make
if [ ! $? -eq 0 ]; then
    cd ..
    echo "Makefail failed try to run it from its
          folder and check if ffmpeg is linked with
          the path in makefile"
    exit -1
fi
cd ..

sleep 2
ImagePicker/start.sh "${input[0]}" "${input[1]}" "${input[2]}"
"${input[3]}" "${input[4]}" "${input[5]}"
"${input[6]}" "${input[7]}" "${input[8]}" "${input[9]}"
"${input[10]}"&
clientpid+="! "
wait $clientpid

```

A.3.3 sender.sh

```

#!/bin/bash
helper="NAME
          Drones Streaming - Esecuzione del programma
          del tirocinio
SYNOPSIS
          helper CONFIGURATION_FILE
DESCRIPTION
          CONFIGURATION_FILE File di configurazione che
          contiene l'input per il sender
AUTHOR
          Written by Massimo Puddu
          "

```

```

if [[ $UID != 0 ]]; then
    echo "Please run this script with sudo"
    echo "sudo $0 $*"
    exit 1
fi
if [ "$#" -ne 1 ]; then
    echo "$helper"
    exit 1
fi
removed_comment=$(grep -o '^[^#]*' "$1")
input[0]=$(echo "$removed_comment" | awk '/^
    pcap_src_file/{print $3}') #posizione dei file pcap
input[1]=$(echo "$removed_comment" | awk '/^
    receiver_IP/{print $3}') #ip su cui inviare i
    pacchetti dal sender
input[2]=$(echo "$removed_comment" | awk '/^
    number_of_links/{print $3}') #numero di canali
input[3]=$(echo "$removed_comment" | awk '/^port/{
    print $3}') #numero della porta
input[4]=$(echo "$removed_comment" | awk '/^
    incoming_feedback_port/{print $3}') #numero della
    porta delle statistiche
input[5]=$(echo "$removed_comment" | awk '/^
    GOP_folder/{print $3}') #cartella dei gop lato
    sender per i GOP
input[6]=$(echo "$removed_comment" | awk '/^VF_folder
    /{print $3}') #cartella immagini sender per le
    immagini
input[7]=$(echo "$removed_comment" | awk '/^simulator
    /{print $3}') #cartella immagini sender
var=$(echo "$removed_comment" | awk '/^channels_stats
    /{$1=$2=""; print $0}') #stampa tutta la riga per
    le statistiche
var=$(echo $var | tr -d "[]") # rimuove le parentesi
    quadre di channels stats
var=$(echo "$var" | sed 's/.;./g') # Si elimina lo
    spazio successivo a ;

```

```

IFS=';' read -a arr <<< "$var" # crea un array per
    ogni ;
i=0
j=0
insert=8
if [[ ${input[7]} -eq 1 ]]; then
    for el in "${arr[@]}"
    do
        i=0
        for val in $el
        do
            if [[ "$val" =~
                ^[+-]?[0-9]+\.[0-9]*$ ]];
            then #controlla se      un
                  numero
                input[$insert]=$val
                ((insert++))
                ((i++))
            fi
        done
        if [[ !($i -eq 6) ]]; then
            echo "Numero di argomenti
                  delle statistiche delle
                  porte sbagliato"
            exit 1
        fi
        ((j++))
    done
    if [[ !($j -eq ${input[2]}) ]]; then
        echo "Il numero di canali non
              coincide con il numero di
              parametri delle statistiche
              "
        exit 1
    fi
else
    fino=$((insert+6*${input[2]}))

```

```

        for i in $(seq $insert $fino)
        do
                input[$i]=0
        done
fi

for i in "${input[@]}"
do
        if [[ -z "$i" ]]; then #controlla che non ci
                siano elementi vuoti
                echo "config file non corretto"
                exit -1
        fi
done

for i in {5..6}
do
        if [ ! -d "${input[$i]}" ]; then # controlla
                che la directory non esista, se non
                esistono le crea
                if ! mkdir -p "${input[$i]}"; then
                        echo "Impossibile creare la
                                cartella gop lato sender"
                        exit 0
                fi
        fi
        length=${#input[$i]}
        last_char=${input[$i]:length-1:1}
        [[ $last_char != "/" ]] && input[i]="${input[
        i]}/"; #aggiunge uno slash finale nei path
done

python3 Sender/Scheduler.py ${input[@]}&
clientpid+="! "
wait $clientpid

```

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.