UNIVERSITÀ DEGLI
STUDI DI PADOVA

FACOLTÀ DI
INGEGNERIA

TESI DI LAUREA SPEIALISTICA

SOFTWARE PER LA GESTIONE REMOTA DI UN SISTEMA DI TELERIABILITAZIONE

RELATORE: Ch. mo Proff. S. Pupolin

LAUREANDO: *Christian Tobaldo*

Padova, 12 luglio 2010

CORSO DI LAUREA IN
INGEGNERIA DELLE
TELECOMUNICAZIONI

DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

Indice

Introduzione

Questa tesi nasce dopo una collaborazione di tirocinio nata con l'ospedale San Camillo del Lido di Venezia. Il settore di ricerca dell'ospedale San Camillo è entrato a far parte di un progetto europeo di Integrated Homecare, che ha lo scopo di studiare il beneficio della continuità domiciliare delle cure cliniche iniziate in ospedale per pazienti che sono stati colpiti da ictus, broncopneumopatia cronica ostruttiva e insufficienza cardiaca, e che necessitano quindi di riabilitazione. La parte del progetto che interessa a noi sarà quella relativa alla tele-riabilitazione, cioè la possibilità di eseguire sedute di riabilitazione beneficiando dell'assistenza domiciliare e quindi con tutti i privilegi che ne comporta.

Questo significa che è necessario predisporre un apparato che permetta al paziente di eseguire delle sedute di riabilitazione a casa propria, ma mantenendo la stessa qualità di quelle "classiche", sia per il paziente che per il medico curante. Per far ciò si è sfruttato il VRRS, un sistema di realtà virtuale creato appositamente per la riabilitazione. Per la fornitura di questo apparecchio ci si è appoggiati alla società Khymeia, con la quale ho collaborato per lo sviluppo del software.

Esiste già in commercio la versione tele del sistema VRRS, è nata però la volontà di riscrivere la parte del software che gestisce la connessione e il trasferimento dei dati e la parte di videoconferenza. Il problema che si è dovuti andare a gestire nasce dalla necessità di trasportare una grande quantità di dati da casa del paziente all'ospedale e viceversa. Si è dovuti andare quindi a cercare di snellire il software e ottimizzare il più possibile i dati da trasmettere con codec adeguati.

Lo scopo finale è quello di riuscire a creare un software abbastanza robusto che riesca resistere ai problemi che possono nascere da una trasmissione real-time in rete e alla grande quantità di dati trasportati.

La tesi sarà strutturata andando prima ad approfondire i vari argomenti con cui ci si è scontrati durante lo sviluppo del software e infine andando a vedere com'è strutturato il software.

In particolare la suddivisione dei capitoli verrà fatta nel seguente modo:

- Nel primo capitolo si andrà a esporre il progetto europeo da cui è nata la necessità di sviluppare il software per il sistema di teleriabilitazione.

- Nel secondo capitolo si cercherà di spiegare innanzi tutto cos'è e a cosa serve il VRRS. Si andrà poi a osservare il funzionamento del sistema nella configurazione locale e valutare quindi il sistema in versione "tele".

- Nel terzo capito si andrà a parlare di DirectShow, un framework e una collezione di API multimediali sviluppato da Microsoft nell'ambito della gestione di file multimediali e degli stream audio e video. DirectShow risulta molto adatto per la cattura d'immagini d'alta qualità e ci baseremo su di esso per lo sviluppo del software.

- Nel quarto capitolo si andrà a valutare il protocollo che a livello trasporto, osservando il modello ISO/OSI, può risultare più adatto al nostro scopo. Andremo quindi ad analizzare i protocolli TCP, UDP e RTP.

- Nel quinto capitolo si andrà a parlare della parte audio del sistema, cioè della codifica utilizzata e dei vari problemi che sono sorti, come l'eco, e che si è cercato di sistemare.

- Nel sesto capitolo si andrà a descrivere cos'è un segnale video e quali sono le sue principali caratteristiche, perché si usa la compressione e le principali tecniche di codifica di un codec video. Sarà poi spiegata la scelta fatta per il codec.

- Nel settimo capitolo si andrà ad analizzare la struttura del software che si è andati a sviluppare.

1 Integrated Homecare Project

In questo capitolo si andrà a esporre il progetto europeo da cui è nata la necessità di sviluppare il software per il sistema di teleriabilitazione.

1.1 Il progetto europeo

Nell'aprile del 2009 è partito il progetto su *Integrated Homecare (IHC)* che ha lo scopo di studiare il modo migliore per poter garantire la continuita delle cure cliniche per i pazienti che sono stati colpiti da ictus, broncopneumopatia cronica ostruttiva (COPD) e insufficienza cardiaca, tre frequenti condizioni che colpiscono almeno un milione di pazienti all'anno in Europa e che necessitano quindi di riabilitazione. Lo scopo del progetto è quello di documentare i benefici dell'assistenza domiciliare integrata con l'uso della riabilitazione e che questa può risultare un notevole risparmio economico per la società.

Ecco cos'è IHC:

- IHC si svolge nella casa del paziente come parte di un percorso integrato tra i servizi di cure ospedaliere, cure primarie e di servizi sociali per pazienti che necessitano di cure specialistiche e riabilitazione
- IHC dovrebbe essere finanziato e coordinato amministrativamente con l'obbiettivo di avere una maggiore efficacia e risparmio delle risorse fisse nel settore dell'assistenza sanitaria e/o dei servizi sociali
- IHC è eseguito da un team multidisciplinare in collaborazione con il paziente in casa sua, nonché in ospedale dove può ricevere le cure generali di cui ha bisogno
- il team di IHC si concentra sull'efficacia, qualità, accesso e soddisfazione degli utenti utilizzando le apparecchiature nella versione "tele" in modo da soddisfare questi obbiettivi

È già stato constatato l'efficacia di IHC per pazienti affetti da ictus, insufficienza cardiaca e COPD. Il progetto mira a completare la base di conoscenze attraverso una serie di studi e indagini parallele per ciascuna delle tre condizioni selezionate.

Questa fase empirica del progetto comprende:

- RCT (randomized controlled trial) su IHC per pazienti colpiti da ictus in Danimarca e Portogallo
- Studi di fattibilità di IHC per insufficienza cardiaca in Svezia, Spagna e Paesi Bassi
- Studi pilotati di formazione terapeutico-domiciliare per pazienti con COPD in Danimarca e in Spagna
- Sviluppo e test pilotati di Tele-riabilitazione come supplemento alla diretta terapeutica IHC
- Indagine europea sulle barriere finanziarie e di organizzazione per IHC in ogni stato membro, nonché uno studio polacco sulla necessità di IHC per i pazienti colpiti da ictus nei paesi a medio reddito.

L'obbiettivo del progetto non è solo quello di completare la base di prove per IHC, ma anche di facilitare la diffusione dei risultati della ricerca IHC. A tal fine si vogliono sviluppare delle guide pratiche basate sui migliori standard internazionali per ognuna delle tre condizioni destinate ai professionisti del campo. Infine, un importante obbiettivo del progetto è quello di stabilire un sistema internazionale, una rete IHC multidisciplinare al fine di organizzare nuove implementazioni regionali in tutta Europa.

Il progetto è stato finanziato dalla Commissione Europea con 2.2 milioni di euro e prevede un gruppo di 12 partner di ricerca provenienti da 8 diversi stati membri dell'UE che dovranno collaborare alla buona riuscita del progetto per una durata di 3 anni.

1.2 Composizione del progetto

Il progetto si articola di 5 pacchetti che dovranno essere sviluppati indipendentemente l'uno dall'altro e che verranno riuniti alla fine del progetto.

1.2.1 Assistenza domiciliare integrata per pazienti affetti da ictus

L'obbiettivo di questo pacchetto è quello di valutare gli effetti di *Early Home Supported Discharge (EHSD)* sulla qualità della riabilitazione in termini di risultati sanitari, costi e soddisfazione, nonché identificare e convalidare i fattori che determinano l'esito dell'integrazione in contesti diversi dalle cure cliniche presso l'ospedale.

Descrizione dei lavori

Segnalazione danese degli RCT su EHSD

Un RCT condotto in 4 centri di riabilitazione in Danimarca, sta per essere riportato come parte di questo progetto, compresa la valutazione degli aspetti quantitativi

e qualitativi. I risultati dovrebbero chiarire le migliori pratiche internazionali nel settore.

Realizzazione di un RCT su EHSD ad Aveiro, in Portogallo, con 140 pazienti

Il modello EHSD danese sta per essere realizzato in un RCT portoghese, al fine di testare il modello in uno stato membro a medio reddito come il Portogallo. In questa fase si valuterà anche se non sarebbe utile l'uso di PDA nel normale funzionamento del EHSD per la cattura diretta dei dati durante le sessioni casalinghe.

Indagine sullo stato di neuroriabilitazione in Polonia

Uno studio approfondito sullo stato di neuroriabilitazione in Polonia e la necessità di EHSD sarà effettuato presso l'Accademia di Educazione Fisica a Katowice, in Polonia, con l'assistenza dell'Università della Danimarca meridionale. Questo progetto serve sia per lo sviluppo della guida al EHSD per gli stati membri a basso e medio reddito, come pure per l'HTA (Health Technology Assessment) finale per IHC

Sviluppo di una guida operativa per EHSD in EU

Come sintesi delle prove esistenti verrà fatta una guida sulla preparazione, l'organizzazione e il funzionamento delle EHSD.

1.2.2 Assistenza domiciliare integrata per pazienti con scompenso cardiaco

Esistono già dei criteri europei per la gestione domiciliare ottimale di pazienti con insufficienza cardiaca. Tuttavia c'è la necessità di un'indagine sistematica. L'obbiettivo di questo pacchetto su *Home Health for Heart Failure (HHHF)* è, anzitutto, descrivere i criteri europei per la gestione domiciliare ottimale di pazienti con scompenso cardiaco e, secondariamente, testare l'intervento domiciliare in 3 paesi con diversi sistemi sanitari.

Descrizione dei lavori

Descrizione dei criteri europei per la gestione domiciliare ottimale di pazienti con scompenso cardiaco

Esperti svolgeranno interventi in casa dei pazienti con insufficienza cardiaca descriveranno i criteri per l'assistenza domiciliare ottimale in base a tre punti:

- Ai centri che hanno risposto al sondaggio sulla gestione programmata di scompensi cardiaci nel 2005 e che hanno descritto il lavoro con programma basato sul domicilio, sarà chiesto di fornire il contenuto del programma e l'esperienza con il telemonitoraggio.
- Gli altri ricercatori del settore saranno invitati a valutare la prima bozza dei criteri ottimali per la gestione domiciliare.
- Il Consiglio delle cure primarie della European Society of cardiology (ESC), il Consiglio sulle cure cardiovascolari e l'Heart Failure Association dell'ESC

dovranno valutare, migliorare e approvare i criteri ottimali per la gestione domiciliare.

Prova di un intervento rispettando i criteri in 3 paesi con sistemi sanitari diversi

Uno studio di fattibilità su un intervento totale di 100 pazienti e un gruppo di controllo di 100 pazienti per paese (Svezia, Paesi Bassi e Spagna) sarà effettuato per verificare l'intervento domiciliare sulla base dei criteri precedentemente fissati. I risultati dello studio saranno la qualità delle cure, la soddisfazione del paziente, i giorni di ospedalizzazione, i tassi di ri-ospedalizzazione e i costi.

Adeguamento dei criteri per la gestione domiciliare per pazienti con insufficienza cardiaca e l'integrazione delle linee guida esistenti

1.2.3 Assistenza domiciliare integrata per pazienti con COPD

Questo pacchetto si concentra su una piattaforma ICT del corrente modello sanitario integrato spagnolo per pazienti con COPD con il duplice scopo di consolidare la piattaforma ICT nonché di valutare l'utilità della piattaforma per interventi di assistenza domiciliare per i pazienti con altre malattie croniche. Inoltre sarà esplorata la possibilità di arricchire l'attuale modello di assistenza integrata con un modulo di formazione terapeutica domiciliare. L'obbiettivo finale del pacchetto è quello di sviluppare una linea guida per l'assistenza domiciliare di pazienti affetti da COPD.

Descrizione dei lavori

Proroga della sperimentazione di piattaforme per COPD

La prima fase si concentra sul consolidamento di prove sulle prestazioni del modello attuale di assistenza integrata per COPD, applicato su altre condizioni croniche. L'idea sarebbe quella di rivalutare l'approccio e i supporti delle piattaforme ICT in pazienti affetti da ictus e scompenso cardiaco (HF). Gli indicatori di equivalenza dei risultati positivi saranno identificati per ictus e HF, in modo da avere una metodologia valida per riportare i risultati con l'obbiettivo di future applicazioni del modello in altri contesti. Da inoltre la possibilità di arricchire il pacchetto standard del modello attuale.

Linee guida per il servizio sanitario domiciliare di COPD

La seconda fase svilupperà una linea guida pratica ai Servizi di Assistenza Domiciliare per COPD basata sui risultati e sulle esperienze della prima fase.

1.2.4 Teleriabilitazione

Questo studio si concentra sulla tele-riabilitazione come un utile complemento per la riabilitazione a casa del paziente. L'obbiettivo è quello di creare delle guide per la tele-riabilitazione e di integrarle successivamente nelle guide per le malattie specifiche di ictus, scompensi cardiaci e COPD.

Descrizione dei lavori

Sviluppo di un piano operativo per TeleRehab in IHC

Ciò include una revisione sistematica degli studi sulla telemedicina in relazione con IHC, nonché la collaborazione col Comitato Direttivo per:

- identificazione delle attività suddivise in moduli per i vari sintomi per applicazioni di tele-riabilitazione,
- valutazione di programmi di formazione tecnologica per i pazienti, gli operatori sanitari e i professionisti,
- identificazione del personale che sarà responsabile della raccolta dei dati per un certo campione di pazienti per le tre tipologie di patologia
- valutazione comparativa delle connessioni web e dei provider internet per gli stati membri che ospitano le prove in questione. Ciò dovrebbe includere la modifica di VRRS.net all'interno dell'ospedale (intra-LAN) e altre piattaforme ICT rilevanti per IHC.

Pilot-testing del piano operativo per TeleRehab

La gestione tecnologica del personale che sarà coinvolto nelle applicazioni di telemedicina. La somministrazione di scale di soddisfazione per l'avvicinamento tecnologico per i pazienti e gli operatori sanitari.

Analisi dei dati e reporting

Analisi dei dati, Pubblicazioni di alcuni risultati su riviste scientifiche. Compilazione di "Guide su TeleRehab" come parte di guide su infarto, HF e COPD.

1.2.5 Healt Tachnology Assessment di Assistenza domiciliare integrata

L'obbiettivo generale di questa proposta è di fornire un *Health Technology Assessment(HTA)* relativa al IHC, che secondo la letteratura potrebbe rappresentare un miglioramento significativo nell'assistenza sanitaria di patologie croniche in generale e a riguardo di ictus, scompensi cardiaci e COPD in particolare.

Una revisione della letteratura ha individuato ictus, insufficienza cardiaca, COPD e malattia mentale come condizioni croniche in cui IHC sembra essere efficace ed economico.

Descrizione dei lavori

Partendo da una revisione della letteratura, una maggiore ricerca sistematica per la sperimentazione clinica su misure IHC si svolgerà con l'uso di cataloghi letterari come *Medline*. La ricerca della letteratura sarà limitata a individuare gli studi su IHC pubblicati dopo l'1 gennaio 2000. Un insieme di criteri di selezione saranno adottati per la ricerca nella letteratura, al fine di concentrarsi sugli studi con il più alto potenziale esplicativo che arrivano a conclusioni per quanto riguarda la sperimentazione clinica su IHC. L'obbiettivo della ricerca nella letteratura deve corrispondere

al significato delle prove IHC incluse nella presente proposta. I risultati della ricerca della letteratura verranno sintetizzati in una sintesi di ciò che è e non è noto per quanto riguarda gli effetti delle misure IHC, se ci sono aree di polemica, e verranno formulate domande per la ricerca futura.

Un punto di partenza dovrebbe essere previsto relativamente a una mappatura più sistematica del livello di integrazione nei sistemi europei di assistenza sanitaria e dell'effetto di macro-caratteristiche dei sistemi di assistenza sanitaria (ad esempio i sistemi di finanziamento e di assicurazione) sui processi di integrazione. La questione di fondo della ricerca è: ci sono le caratteristiche del finanziamento e del sistema normativo (livello macro), che possono ostacolare o favorire lo sviluppo di assistenza domiciliare integrata? L'obiettivo finale è di raggiungere delle regole comuni, che possono essere adottate, dagli stati membri e che possono essere tradotte in Raccomandazioni.

Sintesi di studi sulla efficacia, efficienza, soddisfazione del paziente e l'accesso di IHC per i pazienti con ictus, HF, COPD e le malattie mentali consegnato dai pacchetti di lavoro. A tal fine, i pacchetti di lavoro saranno completati con servizi-HTA, quali analisi costi-efficacia e interviste qualitative con i pazienti, assistenti e operatori sanitari.

Reportage HTA completo di IHC in un formato adatto per luoghi clinici, amministrativi e responsabili politici negli Stati membri dell'UE.

2 VRRS

In questo capitolo si cercherà di spiegare innanzi tutto cos'è e a cosa serve il VRRS. Si andrà poi a osservare il funzionamento del sistema nella configurazione locale e poi a valutare invece il sistema in versione "tele", quindi disposto con una parte in ospedale e l'altra a casa del paziente.

2.1 Cos'è il VRRS

Il VRRS (Virtual Reality Rehabilitation System) è un simulatore di realtà virtuale ideato per la riabilitazione di pazienti che hanno subito danni neurologici. Questo dispositivo è nato poiché in ambiente virtuale il soggetto può avere certe informazioni sul proprio movimento, utili a migliorarlo durante l'esecuzione stessa o nel tentativo successivo. Non di minor rilevanza sono anche i vantaggi di misurazione che offre il sistema di realtà virtuale. La possibilità di rendere obiettiva la valutazione cinematica di un gesto motorio, permette d'individuare l'entità di una performance insoddisfacente e dunque scegliere le vie più adatte per migliorarla; avendo sempre a disposizione dei dati obiettivi per rivalutarla e/o confrontarla.

Si è visto che le persone con disabilità possono apprendere gesti motori in ambiente virtuale e che i movimenti appresi in ambiente virtuale, vengono trasferiti sulla vita reale nella gran maggioranza dei casi. È una simulazione del mondo reale attraverso un software grafico e un computer dedicato, e l'utente ne usufruisce attraverso interfacce uomo-macchina dedicate. Per creare un ambiente virtuale esiste un'enorme varietà di applicazioni software e di sistemi hardware con diverse caratteristiche. Tanti sistemi possono essere utilizzati per creare un ambiente virtuale, tuttavia i componenti di base sono: un computer, una scheda grafica speciale che permetta l'analisi e rendering in tempo reale dei dati in tre dimensioni, dei dispositivi di proiezione per la visualizzazione dell'ambiente virtuale, periferiche hardware per monitorare i dati cinematici del movimento oppure fornire una forza o resistenza simulata e un software dedicato che permetta a tutti questi componenti di lavorare in sincronia. Un ambiente virtuale più immersivo dà al soggetto una percezione più reale dell'ambiente in tre dimensioni.

In aggiunta la realtà virtuale permette d'inserire nel display il *maestro virtuale*, che realizza lo stesso movimento corretto che il soggetto tenta di emulare. Il potere

dell'input visivo della performance del maestro, ripetuta più volte, fornirebbe un miglioramento dato dall'apprendimento per imitazione "learning by imitation". Questo tipo d'apprendimento permette non solo di aumentare il feedback visivo, ma anche di sviluppare la formazione di un corretto pattern di attività cellulare nel sistema nervoso centrale attraverso la ripetizione. Un ambiente virtuale ha la capacità di offrire al soggetto un feedback in tempo reale durante l'esecuzione del movimento, in maniera facile da capire e molto intuitiva. I pazienti possono vedere i risultati dei loro movimenti contemporaneamente a quello del maestro virtuale. Da un'altra parte l'ambiente virtuale offre anche la possibilità di semplificare l'apprendimento, riducendo gli stimoli distraenti che esistono nell'ambiente reale.

In relazione a queste considerazioni il VRRS (Virtual Reality Rehabilitation System), rappresenta un prodotto che soddisfa ampiamente le caratteristiche richieste per facilitare e/o velocizzare l'apprendimento motorio. Il sistema realizzato da Khymeia, permette un utilizzo piu ampio del sistema, per le maggiore duttilita della piattaforma di controllo, con applicazioni possibili anche nella riabilitazione neuropsicologica. Il VRRS crea un ambiente virtuale con il quale il paziente interagisce, secondo i principi del razionale illustrati in precedenza.

Le caratteristiche peculiari di questo sistema sono:

- creare un vero ambiente attorno al paziente;
- richiedere movimenti per l'esecuzione di un compito, sovrapponibili a quelli del mondo reale;
- permettere l'utilizzo di oggetti uguali o molto simili (magari fatti con materiali sicuri o piu leggeri) a quelli veri;
- facile utilizzo da parte dei fisioterapisti.

Dopo anni di utilizzo in neuroriabilitazione e soprattutto con una forte interazione tra gli operatori sanitari ed i tecnici, è stata creata l'attuale versione VRRS, con l'obiettivo di ottimizzare un sistema che soddisfi ancora di più le richieste dei fisioterapisti che lo utilizzano, mirate a facilitare e velocizzare l'apprendimento motorio, senza rendere troppo complicato l'impiego del sistema.

2.2 Scenario locale

Nello scenario locale ci troviamo in una classica situazione di riabilitazione, però con realtà virtuale. Tutte le apparecchiature sono allocate in un'unica stanza e in questo caso il fisioterapista è a contatto diretto col paziente. In questo situazione il paziente ha la possibilità di interagire in maniera istantanea col proprio medico. Il paziente, nonostante i suoi problemi, è in grado di percepire bene le informazioni che gli arrivano dal medico, e nel caso in questione, di eseguire gli esercizi potendo guardare direttamente il maestro che glieli può mostrare di fronte a lui e avere una correzione immediata dei movimenti nel caso ci sia qualche errore. Questo da al paziente una maggior capacità di reagire agli stimoli.

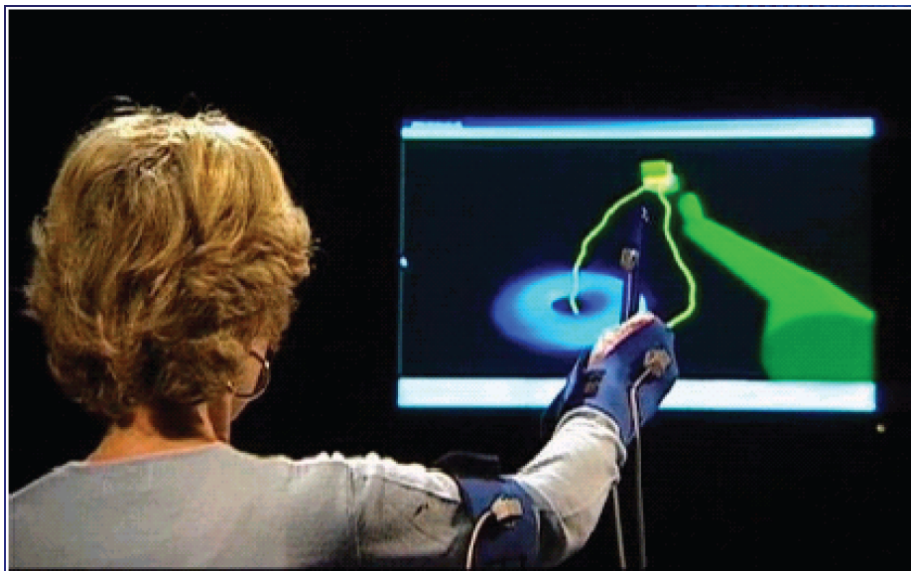


Figura 2.1: Esempio realtà virtuale

Dal lato del fisioterapista c'è una visione diretta degli esercizi che compie il paziente, con possibilità di fermarlo e/o correggerlo immediatamente in caso d'errore. Cosa da non sottovalutare è anche la possibilità di poter vedere e sentire ciò che accade nell'ambiente circostante al paziente, fattore molto importante perché può far capire quale sia la natura di certi errori durante lo svolgimento degli esercizi (causati per esempio da momenti di distrazione per rumori improvvisi).

Figura 2.2: Schema semplificativo in locale

Il sistema utilizzato, come mostrato in figura 2.2, è così composto: il paziente, seduto o in piedi, ha a disposizione *un monitor* dove osservare gli esercizi da fare e i movimenti da lui compiuti, *delle casse* poiché il sistema è in grado di comunicare col paziente per dargli degli stimoli, *un gruppo di sensori* da 1 a 16 collegati mediante cavi di lunghezza di circa 5.5 mt., *di un'antenna* che genera un debole campo magnetico coerente di diametro di circa 2.5 mt. per la lettura in tempo reale del movimento dei sensori.

In questa situazione il fisioterapista è in grado di gestire personalmente la riabilitazione mediante il sistema VRRS con ottimi risultati. Inoltre il sistema VRRS dispone in questo caso di una postazione adeguata che può essere considerata fissa con dimensioni maggiori e con la possibilità di sfruttare tutti i sensori che vengono messi a disposizione.

2.3 Scenario domiciliare

Ciò che si vuole fare è riuscire a trasformare il sistema di riabilitazione in un sistema di *teleriabilitazione* che sia stabile e funzionante. Nello scenario di teleriabilitazione però molte cose cambiano. Infatti in questo caso il paziente e il fisioterapista si trovano in due luoghi fisicamente diversi, il primo si trova a casa propria, o in un altro ambiente predisposto, mentre il secondo si trova in clinica. Questa soluzione è in grado di agevolare entrambe le parti. Il paziente è agevolato poiché riconosce il luogo in cui si trova come un ambiente casalingo, più famigliare, e quindi sentendosi più a suo agio scompare ogni scrupolo che poteva avere in clinica, senza contare che non nasce più la necessità di doversi presentare ogni volta in clinica per fare gli esercizi. Diventa quindi anche un vantaggio sotto l'aspetto della comodità. Come detto in precedenza però i vantaggi nascono anche per il medico che dallo stesso posto può andare a fare sedute di teleriabilitazione a più pazienti, allocati anche in zone diverse della città, senza doversi spostare e ottimizzando così i suoi tempi.

Da parte del medico però iniziano a nascere delle difficoltà poiché non ha più la possibilità di avere un il controllo visivo, e il solo controllo sonoro del paziente potrebbe non essere sufficiente per farsi capire. Nasce così la necessità di avere un feedback visivo che possa permettere, dal lato del medico di poter vedere se il paziente compie gli esercizi in maniera adeguata, mentre dal lato paziente di poter vedere il proprio medico che gli può corregge certi errori durante l'esercizio.

Il risultato è che nasce un sistema di videoconferenza su poi andrà ad appoggiarsi il software dedicato per la riabilitazione. Il nostro sistema di figura 2.2 si va quindi a modificare come mostrato in figura 2.3, o quanto meno si aggiunge una parte che prima non era necessaria. Dalla parte del paziente è necessario aggiungere *una telecamera IP motorizzata*, per poter inviare il proprio video; mentre dal lato del fisioterapista è necessario avere *un computer* per vedere il video e gestire le varie scene, *una webcam*, *delle casse o cuffie* e *un microfono* per poter interagire con i paziente. La scelta di utilizzare una telecamera motorizzata e non una semplice webcam nasce dal fatto che maggiori sono le informazioni che il fisioterapista ha dell'ambiente in cui avviene la riabilitazione, maggiore diventa per lui la possibilità di interagire con il paziente e di dargli dei punti di riferimento utili per il corretto svolgimento di certi esercizi.

Con questa nuova configurazione però iniziano a crearsi dei problemi che devono essere gestiti. Infatti non c'è più il contatto diretto ma ci si deve appoggiare alla rete internet. Questo va inevitabilmente a creare dei ritardi nelle comunicazioni che devono essere ridotti il più possibile per poter permettere una corretta comunicazione. Oltre al problema dei ritardi è necessario valutare anche la qualità dell'audio e del

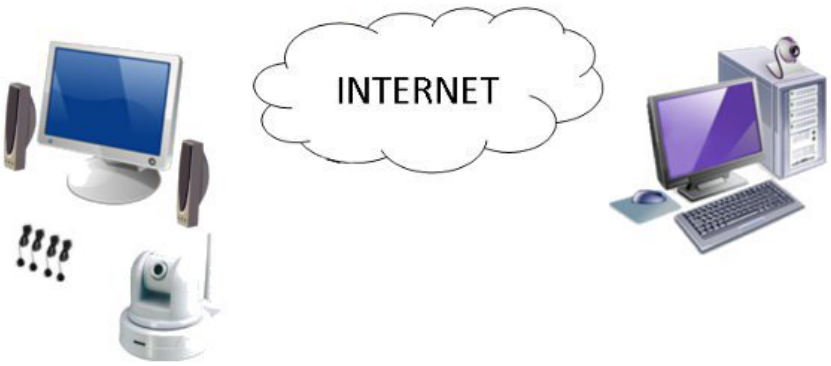


Figura 2.3: Schema semplificativo per il domicilio

video che si va a trasmettere per permettere che la comunicazione abbia una certa fluidità. È necessaria quindi un lavoro di analisi per la scelta di codec e protocolli adeguati a questo tipo di lavoro.

3 DirectShow

Il software che si è andato a sviluppare è scritto con linguaggio di programmazione *C#*. Il C# è un linguaggio di programmazione orientato agli oggetti, sviluppato da Microsoft e approvato come standard ECMA. La sintassi di C# nasce dall'unione dei linguaggi di programmazione Delphi, C++, Java e Visual Basic per poter sfruttare una maggiore semplicità di linguaggio.

In questo capito si andrà a parlare di DirectShow, un framework e una collezione di API multimediali sviluppato da Microsoft nell'ambito della gestione di file multimediali e degli stream audio e video. DirectShow risulta molto adatto per la cattura d'immagini d'alta qualità e supporta una grande quantità di formati.

DirectShow semplifica la riproduzione di contenuti multimediali, la conversione tra formati e la cattura delle immagini. Fornisce inoltre la possibilità di accedere all'architettura alla base del controllo per le applicazioni che richiedono soluzioni personalizzate.

DirectShow si basa su *Component Object Model (COM)*, cioè su un'interfaccia per componenti software che permette la comunicazione tra processi e la creazione dinamica di oggetti con qualsiasi linguaggio di programmazione. Non è necessario però doversi costruire tutti gli oggetti COM necessari per la propria applicazione poiché DirectShow ne mette a disposizione una grande quantità di già creati.

Verrà di seguito data una descrizioni degli elementi principali sfruttati da DirectShow

3.1 Programmazione di applicazioni per mezzo di DirectShow

L'elemento base per la costruzione di blocchi DirectShow è un componente software chiamato *Filtro*. Il filtro è un componente che può compiere varie operazioni, per esempio leggere file, acquisire video da sorgenti, decodificare vari formati di dati, passare i dati alla scheda video o audio.

I filtri sono quindi in grado di ricevere in input una serie di dati e di produrre un output adeguatamente elaborato.

Un'applicazione DirectShow connette diversi filtri tra loro creando una catena, in modo che l'output di uno sia l'input del successivo e così via. Un esempio di

interconnessione tra filtri è rappresentato in figura 3.1. Si può vedere come un file AVI letto da Hard drive possa essere riprodotto da un monitor e delle casse venendo elaborato dai diversi filtri intermedi.

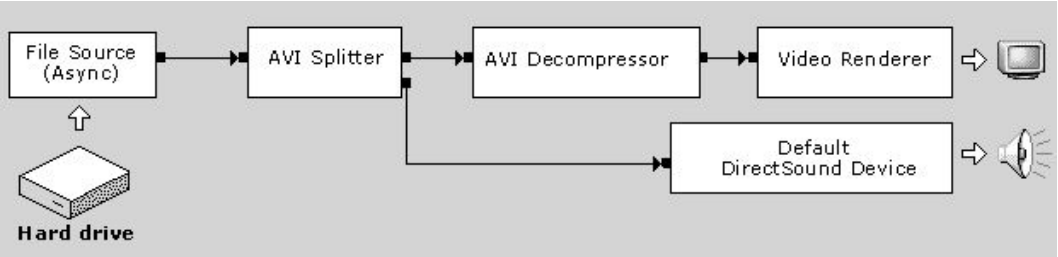


Figura 3.1: Modello di *Grafo*

L'insieme di filtri connessi viene chiamato *grafo*.

L'applicazione non deve preoccuparsi di gestire il funzionamento di tutti i filtri, poiché esiste un componente di più alto livello chiamato *filter graph manager* che controlla tutto. Per esempio dispone di metodi di alto livello come "Run()" e "Stop()" che sono in grado di gestire l'intero grafo, senza dover controllare il funzionamento di tutti i singoli filtri. Se invece è necessario operare direttamente sui filtri bisogna operare attraverso le interfacce COM dei filtri stessi.

Tipicamente un grafo di questo tipo è composto sostanzialmente dalle seguenti categorie di filtri:

- i *filtri sorgente* introducono i dati nel grafo da diversi tipi di sorgenti;
- i *filtri di trasformazione* capaci di elaborare i dati provenienti dai filtri sorgenti per poter essere usati dai filtri di rendering;
- i *filtri di rendering* in grado d'interfacciarsi con l'hardware esterno;
- i *filtri splitter* che dividono uno stream di input in più output;
- i *filtri mux* che combinano e multiplano diversi input per avere un unico stream.

La distinzione tra le categorie non è comunque assoluta, per esempio possiamo avere un filtro che fa sia da sorgente che da splitter.

3.2 Composizione di DirectShow

Lavorando con materiale multimediale ci si presentano davanti molti problemi: grandi quantità di dati da processare molto velocemente, audio e video devono essere sincronizzati, i dati possono arrivare da diverse sorgenti e in molti formati diversi. DirectShow è disegnato proprio per semplificare tutte queste problematiche.

Per raggiungere degli ottimi throughput per lo stream di video e audio, DirectShow sfrutta DirectDraw e DirectSound dov'è possibile. Queste tecnologie permettono di renderizzare in maniera ottimale i dati con l'hardware video e audio. Per gestire la varietà di sorgenti, formati e dispositivi hardware, DirectShow usa appunto un'architettura modulare di filtri.

La figura 3.2 mostra le relazioni tra un'applicazione, i componenti DirectShow e alcuni componenti hardware e software che è in grado di supportare. Come illustra-

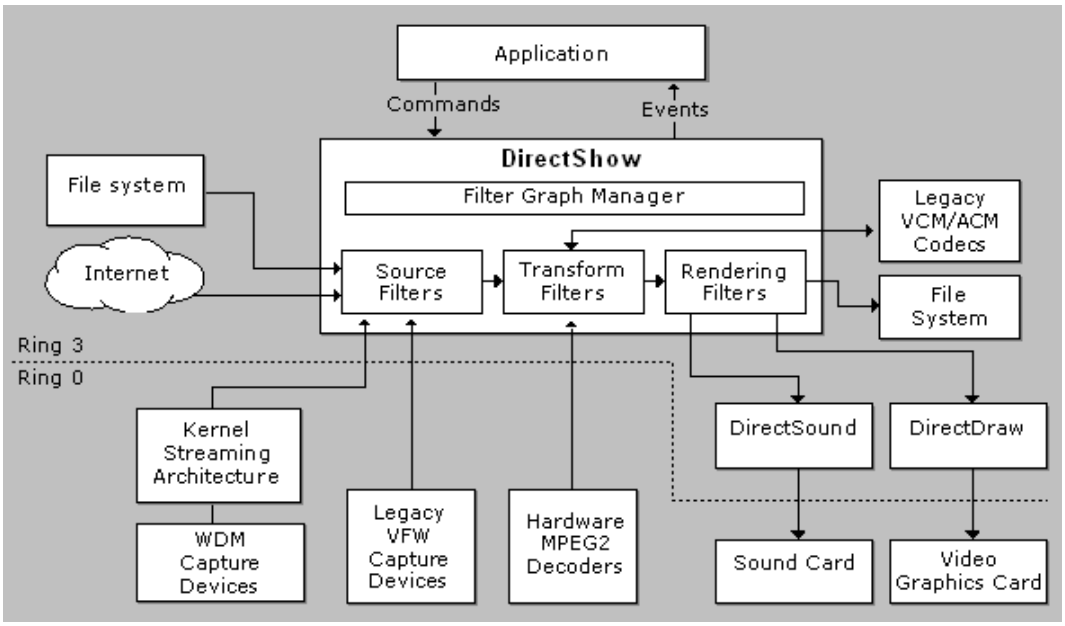


Figura 3.2: Relazioni tra un componente DirectShow e l'esterno

to, i filtri DirectShow sono in grado di comunicare e controllare una grande varietà di dispositivi, inclusi file system locali, sintonizzatori TV e schede di acquisizione video, schermi video e schede audio. Perciò DirectShow è in grado di isolare molti elementi critici di questi componenti. Inoltre è fornito di filtri per compressione e decompressione di certi formati di file.

Come detto anche in precedenza le classi principali di oggetti sono:

Filter Graph Data la natura dell'architettura modulare di DirectShow dove ogni stadio è processato da un oggetto COM chiamato filtro, questi ultimi devono essere collegati tra loro tramite degli altri oggetti COM chiamati *pins* che trasportano i dati da un filtro al successivo.

Filter Graph Manager È un oggetto COM che controlla i filtri in un filter graph. È in grado di coordinare i cambiamenti di stato tra i filtri, stabilire il clock di riferimento, comunicare con l'applicazione.

3.2.1 Filter Graph Manager

Il *Filter Graph Manager* solitamente è in grado di costruire l'intero grafo e gestisce:

Cambiamento di stato. Il cambiamento di stato deve avvenire con un certo ordine in base ai filtri utilizzati. Pertanto l'applicazione non da i comandi di stato direttamente ai filtri, ma da un singolo comando al Filter Graph Manager.

Clock di riferimento. Tutti i filtri del grafo utilizzano lo stesso clock. Il clock di riferimento permette che tutti gli stream siano sincronizzati.

Eventi del Grafo. Filter Graph Manager usa una coda di eventi per informare l'applicazione dell'ordine con cui devono essere gestiti.

Metodi di costruzione del Grafo. Filter Graph Manager fornisce metodi per l'applicazione per aggiungere, connettere e togliere filtri dal grafo.

Per costruire il grafo, Filter Graph Manager usa un processo iterativo, prende il *media type* dal pin d'uscita del filtro e ricerca nel suo registro i filtri che accettano quel tipo di dato in ingresso.

Esso usa vari criteri per la ricerca e la prioritarizzazione dei filtri:

- la categoria del filtro identifica le funzionalità generali del filtro
- media type descrive i tipi di dati che possono essere accettati in ingresso e in uscita dal filtro
- il valore determina l'ordine con cui i filtri verranno testati. Due filtri possono essere della stessa categoria e utilizzare gli stessi tipi di dati ma avere valori diversi a causa della specificità dei filtri stessi.

Il Filter Graph Manager usa l'oggetto *Filter Mapper* per cercare nel registro.

3.2.2 Formato dati Media Types

Poiché DirectShow è modulare, è necessario avere un modo per descrivere il formato dei dati in ogni punto del grafo.

Il *media type* è un modo universale per descrivere i formati digitali multimediali. Quando due filtri devono essere connessi, essi devono avere lo stesso media type poiché il media type identifica il tipo di dati che il filtro a monte consegnerà al filtro a valle e la disposizione fisica dei dati. Se i due filtri non riescono ad accordarsi sul tipo non si connetteranno.

I media types sono definiti dalla struttura **AM_MEDIA_TYPE** che contiene le seguenti informazioni:

- *Major type:* è un GUID che definisce la categoria dei dati. I major types includono video, audio, stream di byte, dati MIDI, e così via.
- *Subtype:* è un altro GUID che definisce il formato. Per esempio per un major type video possiamo avere come subtypes RGB-24, RGB-32, UYVY e altri ancora. Il subtype porta molte più informazioni del major type ma non definisce completamente il formato.
- *Format block:* è un blocco di dati che descrive il formato nei dettagli, per esempio nel caso del video definisce la dimensione dell'immagine e la frequenza dei frame. Il format block è allocato separatamente dalla struttura AM_MEDIA_TYPE, nel membro **pbFormat**, perché la sua struttura cambia in base al tipo di dato.

3.2.3 Media samples e allocator

Il passaggio dei dati tra i filtri avviene attraverso le connessioni dei pin. I dati si muovono dai pin di uscita di un filtro ai pin d'ingresso del successivo. In base al tipo di filtro utilizzato la memoria per i dati può essere allocata in vari modi: ammucchiati, su un livello di DirectDraw, usando la memoria condivisa GDI o con altri meccanismi di allocazione. L'oggetto responsabile per l'allocazione si chiama *allocator* ed è un oggetto COM contenuto nell'interfaccia IMenAllocator.

Quando due pins vengono connessi, uno dei due deve fornire un allocator. DirectShow definisce una sequenza di chiamate a metodi che sono usate per stabilire quale pin deve provvedere all'allocator.

Prima che inizi lo streaming, l'allocator crea un gruppo di buffer. Durante lo streaming il filtro a monte riempie i buffer con i dati e li consegna al filtro a valle. Però il filtro a monte non da al filtro a valle anche i puntatori al buffer, usa oggetti COM chiamati *media samples*, con cui viene creato l'allocator per gestire il buffer. I Media samples sono contenuti nell'interfaccia IMediaSample e contengono:

- il puntatore al buffer relativo
- il riferimento temporale
- vari flags
- opzionalmente, il media type

Il riferimento temporale serve per permettere al filtro di render di programmarsi la renderizzazione, I flags indicano se di sono delle interruzioni dei dati dai campioni precedenti, mentre il media type permette al filtro di cambiare il formato dello streaming.

Finche un filtro utilizza un buffer, esso tiene un contatore di riferimento per i campioni. L'allocator tiene conto del contatore per determinare quando è possibile riutilizzare il buffer. Questo permette di evitare che i filtri sovrascrivano un buffer mentre questo è in uso da un altro filtro.

Flusso di dati in Filter Graph

In questa parte si darà una spiegazione veloce di come si comporta il flusso di dati.

I dati sono memorizzati in un buffer, che è semplicemente un array di byte. Ogni buffer è gestito dall'oggetto *media sample*. I campioni sono creati da un altro oggetto l'*allocator*. Ad ogni pin è assegnato un allocator che può essere messo in comune in caso di due o più pin connessi assieme. In figura 3.3 è mostrato il processo

Ogni allocator crea un gruppo di campioni e alloca i buffer per ogni campione. Quando un filtro ha bisogno di usare un buffer di dati, questo fa la richiesta all'allocator che gli restituisce il puntatore che indica il campione a meno che non sia in uso da altri filtri. Se tutti i campioni sono in uso, l'allocator rimane in attesa finché un campione non si libera. Quando il metodo ritorna il campione, il filtro inserisce i dati nel buffer e setta i flags appropriati.

Nel momento in cui il filtro di render riceve il campione, verifica il riferimento temporale e blocca il campione finché il riferimento temporale del grafo indica che

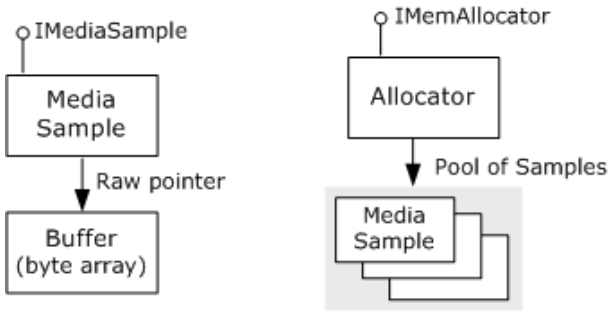
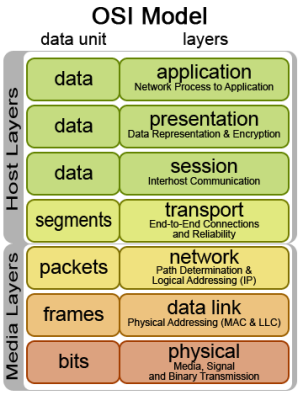


Figura 3.3: Gestione dei dati in un grafo

i dati possono essere renderizzati. Una volta effettuato il render il campione deve essere rilasciato da tutti i filtri che lo stanno utilizzando.

4 Transport Layer

In questo capitolo si andrà a valutare il protocollo che a livello trasporto, osservando il modello ISO/OSI, può risultare più adatto al nostro scopo. Il livello di trasporto può essere considerato il cuore dell'architettura della rete perché ha il compito di instaurare un collegamento logico tra le applicazioni residenti su host remoti. I protocolli applicativi si appoggiano direttamente sul livello di trasporto.



Andiamo ora ad analizzare come funziona il protocollo TCP per poi passare a quello UDP. Si vedrà infine il protocollo RTP che può essere considerato come componente aggiuntiva al protocollo UDP in certe applicazioni.

4.1 TCP - Trasmission Control Protocol

Il protocollo TCP è un protocollo del livello di trasporto orientato alla connessione. TCP nasce dalla necessità di rendere affidabile un canale di comunicazione basato sui servizi del protocollo IP.

4.1.1 Caratteristiche

Le caratteristiche principali del protocollo TCP sono:

- il servizio offerto da TCP è dato dal trasporto di un flusso di byte bidirezionale tra due applicazioni in esecuzione su host differenti;
- il flusso di dati proveniente dall'applicazione può essere frazionato da TCP;
- TCP è orientato alla connessione e quindi ha la funzionalità di creare, mantenere e chiudere una connessione;
- TCP fornisce ai livelli superiori un servizio equivalente a una connessione fisica, quindi garantisce che i dati trasmessi giungano a destinazione in ordine e una sola volta;

- TCP garantisce controllo del flusso e della congestione sulla connessione attraverso un meccanismo di finestra scorrevole. Questo permette di ottimizzare l'utilizzo della rete anche in caso di congestione;
- viene fornito un servizio di multiplazione delle connessioni attraverso il meccanismo delle porte.

TCP è quindi un servizio che garantisce la consegna dello stream di dati da un host a un altro, senza duplicati o perdita d'informazioni. Poiché il trasferimento di pacchetti di per se non è affidabile, si utilizza una tecnica conosciuta come *positive acknowledgment* con ritrasmissione per garantirne l'affidabilità. Il trasmettitore mantiene in memoria i pacchetti che invia e aspetta l'acknowledgment per poter trasmettere i successivi. Inoltre il trasmettitore possiede anche un timer per i pacchetti inviati che ritrasmette nel caso il timer scada.

4.1.2 Struttura segmento TCP

Vediamo ora com'è costituito un segmento TCP. Il segmento TCP si può dividere in due parti, una parte di header e una parte di dati. L'header TCP contiene 10 campi obbligatori e uno opzionale. In figura 4.1 è visualizzato l'header di un pacchetto TCP.

TCP Header

Bit offset	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	Source port																Destination port															
32	Sequence number																															
64	Acknowledgment number																															
96	Data offset				Reserved				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
128	Checksum																Urgent pointer															
160 ...	Options (if Data Offset > 5) ...																															

Figura 4.1: TCP Header

I campi dell'header sono i seguenti:

Source port (16 bit) Identifica il numero di porta sull'host mittente associato alla connessione TCP.

Destination port (16 bit) Identifica il numero di porta sull'host destinatario associato alla connessione TCP.

Sequence number (32 bit) Indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo a partire dall' *Initial Sequence Number*.

Acknowledgment number (32 bit) Ha significato solo se il flag ACK è settato a 1, e conferma la ricezione di una parte del flusso di dati nella direzione opposta indicando il valore del prossimo Sequence Number che l'host mittente si aspetta di ricevere.

Data offset (4 bit) Indica la lunghezza (in parole da 32 bit) dell'header del segmento TCP; tale lunghezza può variare da 5 a 15 parole.

Reserved (4 bit) Bit non utilizzati e predisposti per utilizzi futuri.

Flags (8 bit) Bit utilizzati per il controllo del protocollo:

- **CWR** (Congestion Window Reduced) se settato a 1 indica che l'host sorgente ha ricevuto un segmento TCP con flag ECE settato a 1
- **ECE** (ECN-Echo) se settato a 1 indica che l'host supporta ECN (Explicit Congestion Notification) durante il Three-way handshake
- **URG** se settato a 1 indica che nel flusso sono presenti dati urgenti alla posizione indicata dal campo Urgent pointer
- **ACK** se settato a 1 indica che il campo acknowledgment number è valido
- **PSH** se settato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione
- **RST** se settato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore
- **SYN** se settato a 1 indica che l'host mittente del segmento vuole aprire una connessione TCP con l'host destinatario; ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK
- **FIN** se settato a 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP. Il mittente attende la conferma dal ricevitore FIN/ACK.

Advertise Window (16 bit) Indica la dimensione della finestra di ricezione dell'host mittente (in byte).

Checksum (16 bit) Campo di controllo per la verifica della validità del segmento. È ottenuto facendo il complemento a 1 a 16 bit dell'intero header con l'aggiunta di un pseudo header.

Urgent pointer (16 bit) Puntatore a dato urgente, ha significato solo se il flag URG è settato a 1.

Options (16 bit) Opzioni per usi di protocolli avanzati.

4.1.3 Operazioni adibite al protocollo

Durante una connessione con protocollo TCP si possono sostanzialmente riscontrare tre fasi. Per prima cosa la connessione deve essere stabilita e convalidata grazie a un processo di handshake. Successivamente si passa alla trasmissione dei dati e, completato ciò, avviene il processo di disconnessione dove i circuiti virtuali vengono chiusi e le risorse allocate rilasciate.

In figura 4.2 viene mostrato, in maniera molto semplificata, la serie di cambiamenti di stato che possono avvenire durante una connessione TCP. Gli stati attraversati

sono: *Listen*, nel caso un server stia attendendo una richiesta di connessione da un client; *Syn-Sent*, quando si è in attesa per l'utente remoto di rispondere con il segmento avente i flag SYN e ACK settati; *Syn-Received*, quando si è in attesa per l'utente remoto di rispondere con un acknowledgment dopo aver rimandato l'acknowledgment per la connessione; *Established*, quando la porta è pronta per ricevere/inviare dati; *Fin-Wait*-1; *Fin-Wait*-2; *Close-Wait*; *Closing*; *Last-Ack*; *Time-Wait*, rappresenta il tempo per essere sicuri che il terminale remoto abbia ricevuto l'acknowledgment della richiesta di terminazione della connessione, *Closed*.

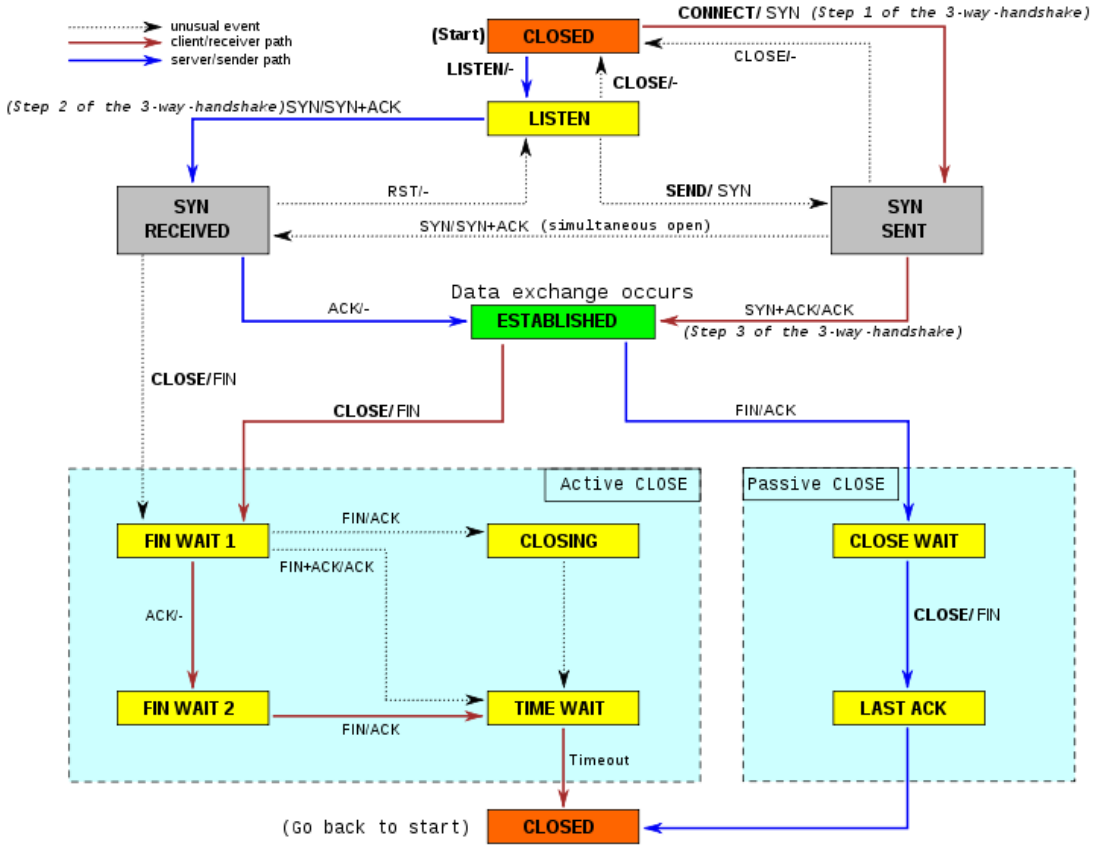


Figura 4.2: Schema semplificato delle operazioni che compie il protocollo TCP

I punti forti di questo protocollo sono quindi: la **stabilità di connessione** grazie al three-way handshake che sfrutta una connessione che si instaura in tre passi: SYN dal cliente, SYN-ACK dal server e ACK finale dal cliente; i **dati trasferiti** hanno la caratteristica di essere ordinati grazie a un numero di sequenza, di essere ritrasmessi nel caso di perdita di pacchetti, di essere eliminati se risultano doppi, di essere trasferiti liberi da errori, di avere un controllo del flusso e della congestione; la dichiarazione del **Maximum segment size (MSS)** che rappresenta la grandezza massima dei dati che TCP sta per inviare in un singolo segmento; gli **acknowledgments selettivi (SACK)** in grado di dare al ricevitore la conferma dell'arrivo di blocchi discontinui di pacchetti; il **window scaling**, sfruttato per massimizzare l'occupazione di banda, va a modificare la dimensione della finestra che può andare dai

2 ai 65, 535 bytes; il **TCP Timestamps** aiuta TCP a calcolate il *round-trip time* tra trasmettitore e ricevitore o nel caso i numeri di sequenza superino il limite di 2^{32}; la **forzatura della consegna dei dati** nel caso le informazioni da inviare siano inferiori al MSS per un lungo periodo di tempo; infine la **terminazione di connessione** gestita da un four-way handshake che si basa su 4 passi: FIN da uno dei due terminali con relativo ACK in risposta e viceversa.

4.2 UDP - User Datagram Protocol

Il protocollo UDP è un protocollo del livedo di trasporto non orientato alla connessione, significa che non è necessario che si instauri una vera e propria connessione tra il trasmettitore e il ricevitore. Offre quindi un servizio "best effort" che permette la possibilità di perdere e di far arrivare in modo non ordinato i pacchetti. La non connessione tra trasmettitore e ricevitore permette di non avere ulteriori ritardi, di semplificare lo stato di connessione del ricevitore e del trasmettitore, di avere header più piccoli e di non avere controllo della congestione, prevede quindi che le applicazioni possano comunicare tra di loro con il minimo delle informazioni possibili. Consente inoltre connessioni multipli, quindi trasmissioni broadcast e multicasting. Questo protocollo, non essendo sensibile ai ritardi, è molto adatto per applicazioni con streaming multimediali. Anche UDP si appoggia al protocollo IP.

4.2.1 Porte

Le applicazioni UDP usano i *datagram sockets* per stabilire le comunicazioni da utente a utente. I sockets utilizzano le porte per far si che ogni applicazione capisca quali siano i propri dati. Una porta è una struttura software che è identificata da un numero, 16 bit, che va da 0 a 65535. La porta 0 è riservata, ma può essere utilizzata come porta sorgente se non ci si aspettano messaggi di risposta.

Le porte sono state divise dall'*Internet Assigned Numbers Authority*:

- Porte 0 - 1023 sono le porte permanenti che sono assegnate e controllate da IANA. Sono usate come porte universali per i server.
- Porte 1024 - 49151 sono le porte registrate. Queste porte non sono assegnate o controllate da IANA, ma possono essere registrate da essa per prevenire duplicazioni.
- Porte 49152 - 65535 sono porte dinamiche che non sono né controllate né registrate. Queste porte possono essere usate da qualunque processo.

4.2.2 Struttura datagramma UDP

Il protocollo UDP non fornisce garanzie ai livelli superiori riguardanti la consegna dei messaggi, per cui non necessita di memorizzare gli stati di questi una volta inviati. Per questo motivo la struttura dell'header UDP risulta molto semplice rispetto a TCP, come si può osservare anche in figura 4.3.

bits	0 – 15	16 – 31
0	Source Port Number	Destination Port Number
32	Length	Checksum
64	Data	

Figura 4.3: UDP Header

L'header risulta quindi essere composto da 4 campi ognuno composto da 2 bytes. I campi sono:

Source port number identifica il numero di porta del mittente del datagramma;

Destination port number identifica il numero della porta sull'host del destinatario del datagramma;

Length contiene la lunghezza totale in bytes del datagramma UDP (header+dati);

Checksum contiene il codice di controllo del datagramma (header+dati+pseudo header);

Data contiene i dati del datagramma

Come detto in precedenza, poiché questo protocollo prevede che le applicazioni possano comunicare tra loro con il minimo delle informazioni possibili, si può constatare che nel datagramma UDP l'unica operazione sostanziale che viene effettuata è quella di aggiungere il campo con il numero di porta sorgente e destinazione rendendo i pacchetti da inviare molto snelli.

4.3 Confronto tra TCP e UDP

Andiamo quindi ora a valutare le principali differenze tra TCP e UDP.

La prima cosa che si nota è che UDP non offre nessuna garanzia dell'arrivo dei datagrammi né del loro ordine di arrivo ma li lascia gestire al livello applicazione, al contrario il TCP, considerato affidabile grazie ai meccanismi di acknowledgement e di ritrasmissione su timeout, riesce a garantire la consegna dei dati così come vengono inviati. Meccanismi necessari per questo tipo di protocollo poiché l'elaborazione da parte dell'applicazione non parte se tutti i pacchetti non arrivano in ordine. Questo però produce in TCP un notevole aumento dell'overhead e quindi va ad aumentare inutilmente, sotto questo aspetto, l'occupazione di banda.

Altra differenza caratteristica sta nella connessione. Infatti il protocollo TCP è orientato alla connessione, pertanto per stabilire, mantenere e chiudere una connessione, è necessario inviare pacchetti di servizio i quali costituiscono un ulteriore aumento dell'overhead di comunicazione. Inoltre per inizializzare la sessione in TCP è necessario il processo di three-way handshake che provoca una maggiore latenza anche in fase di partenza. Al contrario UDP è un protocollo non orientato alla

connessione e invia solo i datagrammi richiesti a livello applicativo con un notevole risparmio. Le informazioni inviate da TCP sono lette come stream di byte, mentre per UDP sono considerate come sequenze di datagrammi.

Inoltre, essendo il protocollo UDP non orientato alla connessione, ogni pacchetto viene trattato in maniera indipendente e non si fa alcun tentativo di controllo della congestione, al contrario il TCP assume che la perdita di pacchetti significhi congestione e ciò fa si che il protocollo TCP utilizzi un algoritmo per l'ottimizzazione del throughput andando a ridurre brutalmente la quantità di pacchetti inviati (cosa che per una comunicazione real-time non va per niente bene).

Dal punto di vista della sicurezza il protocollo TCP ha notevoli vantaggi e ha risolto molte minacce che invece preoccupavano il protocollo UDP e la gestione dei suoi pacchetti da parte dei firewall.

Si vede quindi, con questo breve confronto, che l'utilizzo del protocollo TCP rispetto a UDP è, in generale, preferito quando è necessario avere garanzie sulla consegna dei dati, sull'ordine di arrivo dei singoli segmenti o sui tempi di consegna. È infatti utilizzato da molte delle applicazioni più popolari di internet. Al contrario il protocollo UDP è usato principalmente quando l'interazione tra i due host è idempotente o nel caso si abbiano forti vincoli sulla velocità, l'economia di risorse della rete o la necessità di sostenere più utenti. Molto utile risulta questo protocollo per le applicazioni in tempo reale come nel nostro caso. A noi interessa infatti che questo protocollo riesca a trasportare una grande quantità d'informazioni senza eccessivi ritardi, poiché ci deve essere un'interazione in tempo reale tra i due host, inoltre non ci interessa se c'è la perdita di qualche pacchetto poiché l'applicazione è in grado di scartarlo senza conseguenze per l'utente.

Accertato quindi che dobbiamo scartare il protocollo TCP perché a noi interessano di più le caratteristiche del protocollo UDP, andiamo ora a vedere cos'è e com'è composto il protocollo RTP poiché è nato per gestire trasmissioni real-time.

4.4 RTP - Real Time Protocol

Il **Real Time Protocol**, detto **RTP**, è un protocollo che dovendo gestire applicazioni real-time si basa su UDP, infatti i processi di instaurazione di un collegamento stabile e della correzione degli errori sono perfettamente inutili per questo tipo di applicazioni. Poiché RTP fornisce alcuni servizi (sequence number,timestamp,ecc.) alle applicazioni multimediali, può essere considerato come un sottolivello del livello di trasporto, come si può osservare in figura 4.4.

Le applicazioni con streaming multimediale real-time necessitano di una consegna tempestiva delle informazioni, ma allo stesso tempo tollerano la perdita di alcuni pacchetti. È per questo motivo che RTP non si appoggia a TCP, che introdurrebbe una latenza troppo elevata a causa della sua natura, ma si basa su UDP.

Il protocollo RTP è inoltre in grado di supportare trasferimenti di dati a destinatari multipli e quindi con distribuzione multicast.

Le specifiche RTP descrivono due sotto-protocolli:

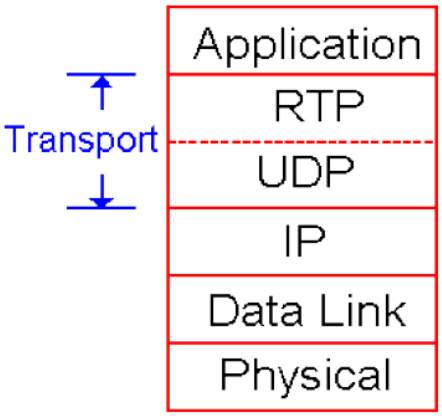


Figura 4.4: Rtp visto come un sottolivello del livello trasporto

- *Real-time Transport Protocol(RTP)*, usato per trasportare dati multimediali con proprietà real-time;
- *RTP Control Protocol(RTCP)*, usato per specificare la qualità del servizio e trasmettere le informazioni relative ai partecipanti della sessione. La banda occupata dal traffico RTCP rispetto a RTP si aggira sul 5%.

Scenario con Mixer and Traduttori

In questo paragrafo si illustra un esempio delle funzionalità di RTP. Si è sempre assunto che tutti i siti vogliano ricevere le informazioni multimediali nello stesso formato, tuttavia questo non è sempre vero. Consideriamo il caso in cui alcuni dei partecipanti collegati siano in una zona connessa tramite un link a bassa velocità, mentre la maggior parte dei partecipanti siano in una zona con un link ad alta velocità. Invece di costringere tutti i partecipanti a usare una minore banda riducendo le qualità della codifica audio, un ripetitore di livello RTP, chiamato *mixer*, può essere posizionato vicino alle aree a bassa velocità. Questo mixer è in grado di risincronizzare i pacchetti audio ricevuti, ricostruire i pacchetti e adattarli al link a bassa velocità. In questo caso l'header RTP viene fornito di tutti gli identificatori dei mixer che hanno contribuito alla costruzione del pacchetto per poter essere chiaramente letto dal ricevitore e per poter successivamente rispondere.

Alcuni dei partecipanti della conferenza audio invece possono essere connessi con un accesso a banda larga, ma non essere direttamente raggiungibili da pacchetti IP, per esempio perché dietro un firewall che non lascia passare i pacchetti IP. In questi casi i mixer non sono necessari, ma serve un altro tipo di ripetitore, chiamato *traduttore*. Sono necessari due traduttori, uno per ogni lato del firewall, con quello esterno che convoglia tutti i pacchetti ricevuti attraverso una connessione sicura all'interno del firewall. Il traduttore interno invece li ritrasforma per farli arrivare ai giusti destinatari.

Mixer e traduttori possono essere progettati per una grande varietà di scopi. Un esempio è un mixer video che scala le immagini di singole persone in stream video

separati e li ricompone in un unico video stream per simulare una scena di gruppo. Altro esempio di traduttore può essere dato quando è necessario tradurre un flusso IP/UDP in un flusso ST-II, oppure una traduzione pacchetto per pacchetto da fonti che non sono sincronizzate

4.4.1 RTP Data Transfer Protocol

Una delle prime considerazioni da fare è che RTP supporta una notevole quantità di formati multimediali (come H.264, MPEG−4, MJPEG, MPEG, etc.,) e permette l'aggiunta di nuovi formati mantenendo la compatibilità e senza dover rivedere lo standard. RTP è basato su un'architettura principale conosciuta come *Application Level Fraiming*. Le informazioni richieste da una specifica applicazione sono specificate da un determinato *Profilo* e da uno o più *Payload* associati. Il *Profilo* definisce il tipo di codec utilizzati per codificare i dati di *Payload*, che a sua volta contiene i dati che ci interessano.

RTP Fixed Header Fields

In figura 4.5 è visualizzato l'header di un pacchetto RTP.

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Ver.	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers (optional) ...						

Figura 4.5: RTP Header

L'Header RTP ha una dimensione minima di 12 bytes. A seguito dell'header può essere preesistente un'estensione opzionale. I campi dell'header sono i seguenti:

Ver. (2bits) indica la versione del protocollo.

P (Padding) (1 bit) Usato per indicare se ci sono bytes addizionali che non fanno parte del payload. Necessario nel caso si usino alcuni algoritmi di crittografia.

X (Extension) (1 bit) Indica la presenza dell'estensione dell'header.

CC (CSRC Count) (4 bits) Contiene il numero degli identificativi CSRC che seguirà la parte fissa dell'header.

M (Marker) (1 bit) Utilizzato a livello applicazione e definito dal profilo. Se è settato indica che i dati presenti hanno una rilevanza speciale per l'applicazione.

PT (Payload Type) (7 bits) Indica il formato del payload e determina la sua interpretazione da parte dell'applicazione.

Sequence Number (16 bits) Il numero di sequenza si incrementa di uno per ogni pacchetto RTP inviato e può essere usato dal ricevitore per rilevare la perdita di pacchetti. A RTP non è comunque concessa la correzione degli errori poiché è compito dell'applicazione. Inoltre il valore iniziale di SN dovrebbe essere casuale per questioni di sicurezza.

Timestamp (32 bits) Usato per indicare l'esatto istante di campionamento del primo byte del pacchetto dati RTP. L'istante di campionamento deve essere derivato da un clock che deve essere incrementato monotonicamente e linearmente in tempo in modo da consentire i calcoli di sincronizzazione. La frequenza del clock dipende dal formato della portante dei dati di payload ed è specificata nel profilo. Anche in questo caso, come per il Sequence Number, il valore iniziale dovrebbe essere casuale.

SSRC (32 bits) Campo identificativo univoco per la sincronizzazione della sorgente. Questo identificatore dovrebbe essere scelto in maniera casuale con l'intento di non avere due sorgenti nella stessa sessione RTP e con lo stesso valore SSRC.

CSRC Identifica i contributi delle sorgenti contenuto nel payload del pacchetto nel caso di stream generato da sorgenti multiple. Il numero degli identificativi è dato dal campo CC. Se le sorgenti sono più di 15, solo 15 potranno essere identificate.

4.4.2 RTP Control Protocol - RTCP

Come detto in precedenza RTP ha un sotto-protocollo. Il RTP control protocol (RTCP) è basato su una trasmissione periodica di pacchetti di controllo a tutti i partecipanti della sessione, usando lo stesso meccanismo di distribuzione dei pacchetti dati. Il protocollo sottostante deve fornire un multiplexaggio tra i pacchetti dati e quelli di controllo. RTCP svolge quattro funzioni:

1. La funzione primaria è di prevedere un feedback della qualità nella distribuzione dei dati. Questa è una parte integrale del ruolo di RTP come protocollo di trasporto ed è connesso con le funzioni di controllo del flusso e della congestione di altri protocolli di trasporto. Il feedback può essere direttamente utile per tecniche di encoding adattativo.

2. RTCP trasporta un identificatore di trasporto-livello per la sorgente RTP chiamato *canonical name* o *CNAME*. Dato che l'identificatore SSRC può cambiare se viene scoperto un conflitto o il programma viene riavviato, il ricevitore necessita del CNAME per tenere traccia di ogni partecipante. Il ricevitore può richiedere il CNAME per associare multipli stream di dati da un dato partecipante, per esempio per sincronizzare audio e video.

3. Le prime due funzioni richiedono che tutti i partecipanti inviino pacchetti RTCP, però bisogna comunque mantenere il rate sotto controllo anche nel caso di un elevato numero di partecipanti. Poiché ogni partecipante invia questi pacchetti a tutti gli altri, ognuno può valutare indipendentemente il numero di partecipanti. Questo numero è usato per calcolare il rate.

4. Una quarta funzione (opzionale) è quella di trasmettere le informazioni minime di sessione, per esempio la visualizzazione a schermo delle informazioni dei partecipanti. Funzione molto utile in caso di sessione in cui i partecipanti possono entrare ed uscire senza un controllo di accesso o dei parametri di negoziazione.

Le funzioni 1-3 dovrebbero essere usate in tutti gli ambienti, ma particolarmente in un ambiente IP multicast. Trasmissioni RTCP possono essere controllate separatamente da mittenti e riceventi nel caso si instauri una comunicazione unidirezionale dove non sia possibile inviare feedback.

Formato pacchetto RTCP

Il pacchetto RTCP trasporta una vasta varietà di informazioni:

SR *Sender report*, per le statistiche di trasmissione e ricezione dei mittenti attivi

RR *Receiver report*, per le statistiche di ricezione dai partecipanti che non sono mittenti attivi

SDES voci che descrivono la sorgente, incluso il CNAME

BYE indica la fine della partecipazione

APP funzioni specifiche per l'applicazione.

Ogni pacchetto RTCP inizia con una parte fissa simile a quella di RTP, seguita da un blocco di elementi strutturati che può variare in lunghezza ma deve finire con un limite di 32 bit.

Per quanto riguarda i pacchetti multipli RTCP, questi possono essere concatenati senza interventi di separazione in un *pacchetto RTCP composto* che verrà inviato come un singolo pacchetto del protocollo del livello inferiore. Non c'è un contatore esplicito dei singoli pacchetti RTCP nel pacchetto composto, ma ci devono pensare i livelli inferiori a fornire ai pacchetti la lunghezza totale per poter determinare la fine del pacchetto composto. Ogni singolo pacchetto RTCP del pacchetto composto può essere processato indipendentemente senza richieste particolari di ordine o combinazione dei pacchetti.

Nonostante tutte queste libertà, sono comunque imposti dei vincoli da seguire per il corretto funzionamento del protocollo:

- le informazioni statistiche SR e RR devono essere inviate spesso, sempre rispettando i vincoli imposti dalla banda a disposizione, con lo scopo di massimizzare la risoluzione delle statistiche; perciò periodicamente i pacchetti RTCP composti devono contenere un pacchetto di resoconto.

- I nuovi clienti devo riuscire a ricevere il prima possibile il CNAME dalle sorgenti per poterle identificare e associare loro i rispettivi stream, quindi ogni pacchetto RTCP deve includere SDES CNAME.

- Il numero dei tipi di pacchetti deve apparire all'inizio dei pacchetti composti deve essere limitato per aumentare il numero di bit costanti nell'ultima parola.

Così, tutti i pacchetti RTCP devono essere inviati in un pacchetto composto di almeno due singoli pacchetti, con il seguente formato:

Prefisso crittografato Se e solo se il pacchetto composto deve essere crittografato, qui deve essere inserito il prefisso di 32bit casuali ricalcolato per ogni pacchetto composto trasmesso.

SR o RR Il primo pacchetto RTCP in un pacchetto composto deve essere sempre un pacchetto di resoconto per facilitare la convalida dell'header.

RRs addizionali Se il numero di sorgenti risultano essere superiori a 31 i pacchetti addizionali RR seguiranno il pacchetto iniziale di resoconto.

SDES Un pacchetto SDES contenente la voce CNAME deve essere incluso in ogni pacchetto RTCP composto. Altre informazioni di sorgente possono essere incluse in base alle richieste dell'applicazione.

BYE o APP Altri tipi di pacchetti RTCP possono seguire in qualunque ordine, eccetto BYE che deve essere l'ultimo pacchetto inviato.

È raccomandato che i codificatori e i mixer combinino i singoli pacchetti RTCP provenienti da sorgenti multiple in un pacchetto composto al fine di ammortizzare il pacchetto overhead. Un esempio di pacchetto composto prodotto da un mixer è dato in figura 4.6. Se la lunghezza totale del pacchetto composto supera l'unità massima di trasmissione (MTU) della connessione di rete, potrebbe essere segmentato in più corti pacchetti composti multipli per essere inviati separatamente. Questo non danneggia la stima dell'occupazione di banda perché ogni singolo pacchetto composto rappresenta al massimo un distinto partecipante.

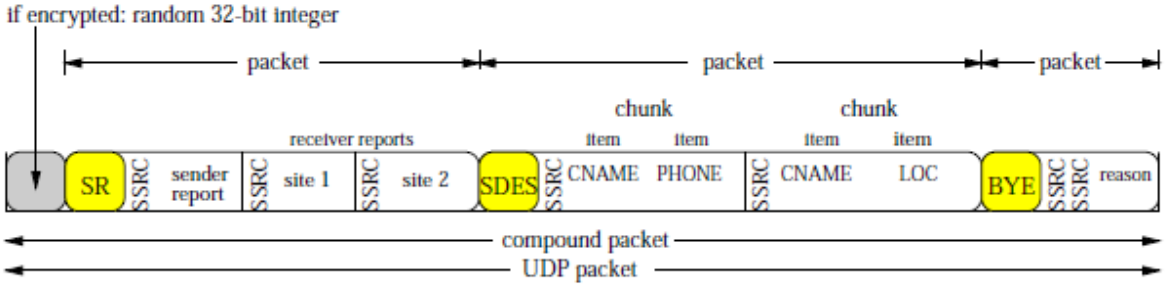


Figura 4.6: Esempio di pacchetto RTCP composto

5 Audio

In questo capitolo si andrà a parlare della parte audio del sistema, cioè della codifica utilizzata e dei vari problemi che sono sorti come l'echo e che si è andati a cercare di sistemare.

5.1 La voce umana

Per capire come deve funzionare un codificatore per la voce è necessario capire prima come funziona la voce umana. Quando una persona parla forza, con l'uso dei muscoli interni, l'aria a muoversi dai polmoni, attraverso il *tratto vocale* fino a uscire e arrivare al microfono. Il tratto vocale si estende dalla glottide alla bocca. Il suono viene prodotto quando la glottide, che è l'apertura delle corde vocali, vibra aprendosi e chiudendosi.

L'interruzione del flusso d'aria crea delle sequenze d'impulsi che hanno alla base delle frequenze chiamate *portanti*. Per gli uomini questa frequenza è stimata sugli 80-160 Hz, mentre per le donne si aggira sui 180-320 Hz. In figura 5.1 si può vedere la natura periodica di un tipico segnale voce prodotto dalla pronuncia della sequenza "aaa". Per la creazione del codec risulta molto importante questa sua natura periodica.

La più forte frequenza tra le componenti voce viene appunto chiamata portante. In figura 5.2 si può osservare lo spettro del campione vocale di figura 5.1 e si vede che le portanti sono posizionate a frequenze di n*100 Hz.

Un gran numero di codec voce modellano il tratto vocale come un filtro e poiché il tratto vocale cambia molto di rado, anche la funzione di trasferimento del filtro di modellizzazione non deve essere aggiornato di frequente, circa ogni 10-40 ms.

Come è noto, le frequenze che l'essere umano è in grado di percepire vanno dai 20 Hz fino ai 20 kHz. La voce umana però ne utilizza solo una parte, per cui sarebbe inutile, per l'utilizzo che serve a noi, andare ad analizzare l'intera banda di frequenze e trasmetterla tutta, perché andrebbe solo a occupare inutilmente una grande quantità di banda internet. Solitamente quindi si va a sfruttare la *banda vocale* per la trasmissione della voce che va da 300 a 3400 Hz. Per questo motivo la banda presa in considerazione è di 4kHz, per includere le bande di guardia. Infatti a noi serve che venga trasmessa la voce e non è necessario che vengano trasmessi altri

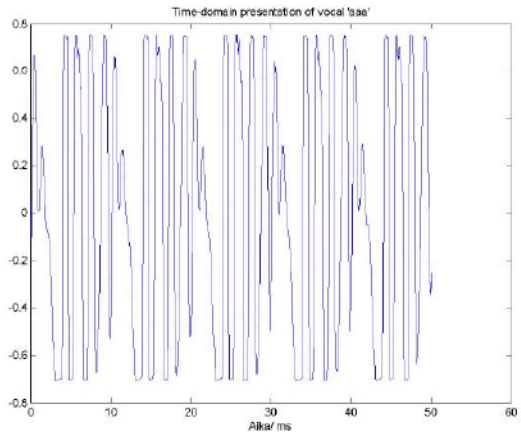


Figura 5.1: Esempio di forma d'onda dalla pronuncia di "aaa"

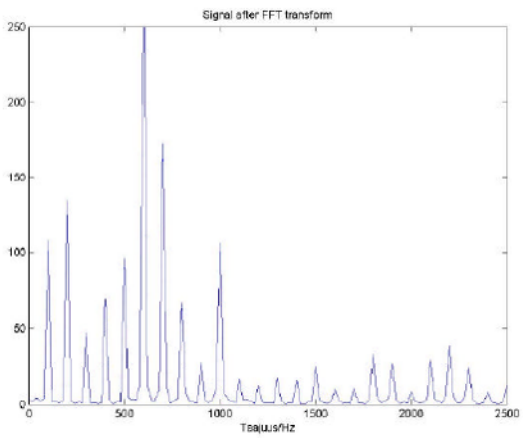


Figura 5.2: Trasformata della forma d'onda di "aaa"

suoni particolari. Per quanto riguarda la voce, noi possiamo inoltre tollerare una certa quantità di distorsione, ma non accettiamo che il ritardo tra utente e utente sia troppo elevato. In realtà la tolleranza massima di ritardo accettabile scientificamente è ancora sotto dibattito, ma generalmente può essere accettabile un range tra i 100 e i 600 ms. Se prendiamo ad esempio la telefonia pubblica abbiamo una tolleranza massima specificata di 600 ms.

Un'altra caratteristica importante per il sistema di voce umana è che l'orecchio umano non può sentire le variazioni del segnale vocale al di sotto di certe grandezze. Quindi possiamo considerare che l'orecchio umano abbia una divisione logaritmica della percezione dei suoni, più sensibile per basse frequenze e meno sensibile per alte frequenze.

Ma andiamo a vedere ora alcune caratteristiche che deve avere il codificatore.

5.2 La codifica

La conversione della voce come forma d'onda analogica in digitale viene chiamata appunto codifica. I maggiori benefici nell'uso della codifica consistono nella possibilità di comprimere il segnale per ridurne il bit rate mantenendo comunque una qualità del livello audio accettabile.

La codifica audio può essere caratterizzata sostanzialmente da quattro fattori: bit rate, ritardo, complesità e qualità. È necessario però dare una maggiore importanza ai requisiti più importanti a discapito di quelli meno rilevanti.

Il bit rate è quello che viene preso in considerazione per primo quando è necessario fare una codifica. Bisogna infatti pensare a dove andranno a finire i nostri dati e con che mezzo verranno trasportati. Nel caso come il nostro di una trasmissione via internet, è quindi necessario rispettare certi vincoli imposti dalla banda che ci viene messa a disposizione.

Il ritardo dovuto al codec può avere un forte impatto per particolari applicazioni. I codec voce per applicazioni real-time non possono avere ritardi troppo elevati, altri-

menti nel corso della conversazione gli interlocutori non riusciranno a farsi capire. Se invece consideriamo applicazioni di memorizzazione multimediale, dove la trasmissione ha direzione unica, allora in quel caso non ci saranno problemi perché avremo un tempo virtualmente illimitato per codificare la voce. È stato studiato infatti che in una conversazione, già con un ritardo superiore a 300ms si inizia a percepire la comunicazione come half-duplex, mentre un tale ritardo per un file audio non viene neanche percepito. Il fattore ritardo è quindi necessario che venga valutato in base al tipo di applicazione che si vuole usare.

Naturalmente abbiamo che anche la complessità e la qualità vanno ad influire sul ritardo e sul bit rate. Infatti una maggiore complessità dell'algoritmo andrà a migliorare la qualità della voce trasmessa, a discapito però del ritardo e del bit rate che andranno ad aumentare. Viceversa una minore complessità può andare a migliorare le prestazioni relative al ritardo e al bit rate, ma va a peggiorare la qualità dell'audio.

Il codec voce ideale dovrebbe quindi avere un basso bit rate, un'alta qualità, un basso ritardo e una bassa complessità. Purtroppo non esistono codec con tutte queste caratteristiche messe assieme, ma bisogna cercare di scegliere il codec con le caratteristiche che più si adattano ai nostri scopi.

5.3 GSM 06.10 Full Rate

Si è scelto di andare a sfruttare il codec GSM 06.10 per la trasmissione della voce perché tra quelli a disposizione è il più robusto, con una buona qualità dell'audio e con un bit rate ridotto.

Lo standard di codifica GSM 6.10, detto "Full Rate" si basa sulla tecnica RPE-LTP (Regular Pulse Excitation - Long Term Prediction) con codifica della voce a 13 kbit/s.

L'encoder processa blocchi di voce da 20 ms, converte il segnale digitale di 13 bit campionati a 8 kHz in blocchi di 260 bit per ogni 160 campioni originali. Da questi 260 bit nascono i 13kbits/s, infatti $260bits/20ms = 13kbits/s$. Ogni blocco da 260 bit è costituito da tre parti (188+36+36) come mostrato in figura 5.3, e nascono da:

- un filtro di predizione lineare (short-term prediction)
- un filtro di predizione a lungo termine (long-term prediction)
- un filtro RPE

La suddivisione dei bit per il codec GSM Full-Rate nel dettaglio è mostrata in figura 5.4.

Andando a osservare la figura 5.5 andremo a vedere più in dettaglio cosa avviene in un encoder RPE-LTP. La voce che viene passata in ingresso viene campionata con una frequenza di 8 kHz e quantizzata con 13 bit usando una codifica PCM. Il segnale viene segmentato in pezzi da 20 ms. Ogni pezzo contiene 160 campioni ed è codificato in 260 bit.

Nella parte di pre-processing avviene una compensazione dell'offset e un processo di pre-enfasi. L'uscita viene quindi inviata all'analizzatore LPC per calcolare i

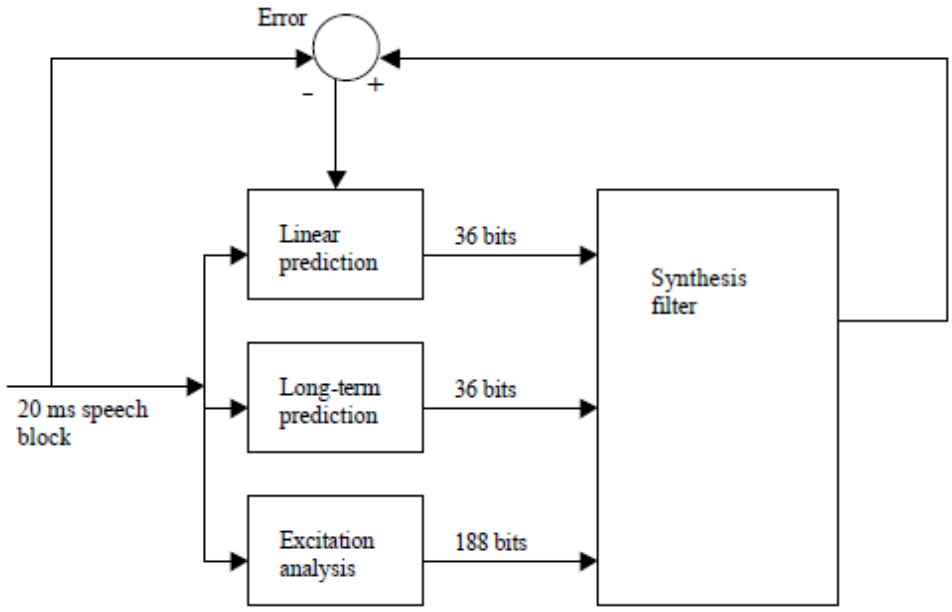


Figura 5.3: Schema rappresentativo codec GSM Full-Rate LPC-RTE

		Bits per 5 ms block	Bits per 20 ms block
LPC filter	8 parameters		36
LTP filter	Delay parameter	7	28
	Gain parameter	2	8
Excitation signal	Subsampling phase	2	8
	Maximum amplitude	6	24
	13 samples	39	156
Total			260 bits

Figura 5.4: Distribuzione dei bit in un codec GSM Full-Rate

parametri dell'ottavo ordine del filtro di predizione lineare composto da 36 bit e con funzione di trasferimento del tipo

$$A(z) = 1 + \sum_{k=1}^{8} a_k z^{-k}$$

Questi parametri verranno utilizzati per filtrare gli stessi 160 campioni iniziali, creando così i 160 campioni *residui a breve termine*. Al fine di quantizzare in modo efficiente i parametri del filtro di predizione lineare, questi vengono trasformati in LARs (logarithmic area rations) prima della trasmissione.

Nelle operazioni successive, ogni frame viene diviso in 4 subframe con 40 campioni che rappresentano 5 ms del segnale residuo a breve termine. I parametri del filtro di predizione a lungo termine, che sono il *ritardo LTP* e il *guadagno LTP*, sono stimati in ciascun sub-frame e sono rispettivamente di 7 e di 2 bit. Mettendo assieme le quattro operazioni che compie nei 20 ms lo stimatore produce $4 * (7 + 2) = 36$ bits.

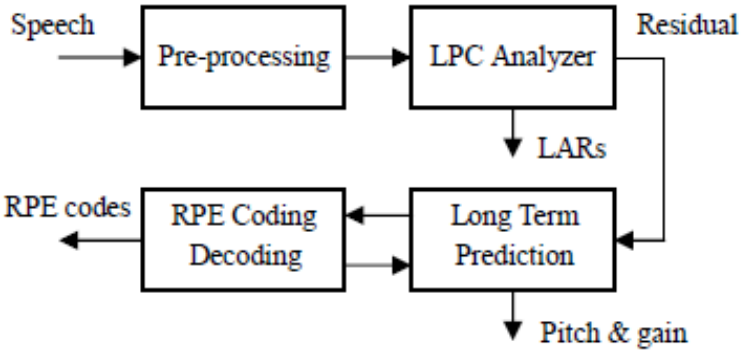


Figura 5.5: Encoder RPE-LTP

Il fattore di guadagno del campione voce previsto assicura che la voce sintetizzata abbia lo stesso livello di energia del segnale voce originale. Il segnale *residuo a lungo termine* si ottiene filtrando il segnale residuo a breve termine con il filtro LTP.

Alla fine, i residui LTP vengono scalati con un rapporto di 3 ottenendo 3 sequenze di interleaved che poi vengono di nuovo separate in 4 sottosequenze RPE finite, ciascuna con 13 campioni. Le sottosequenze vengono selezionate e identificate dalla griglia di posizione di RPE e vengono poi quantizzate a 3 bit per campione usando una PCM adattativa. I bit così ottenuti svolgeranno le funzioni base dell'algoritmo di compressione.

L'encoder voce GSM produce quindi tre diversi gruppi di dati: i parametri del filtro di predizione lineare, i parametri del filtro di predizione a lungo termine e i parametri RPE. Questi parametri e i loro bit individuali non hanno uguale importanza per il rispetto della ricostruzione della voce con qualità. Nel canale voce GSM Full-Rate, i 260 bit codificati devono essere riordinati con un certo ordine d'importanza definito da GSM 05.03 prima di essere sottoposti alle funzioni di codifica di canale. I bit riorganizzati sono poi classificati in tre gruppi: I_a, I_b e II, che verranno protetti in modi diversi come mostrato in figura 5.6.

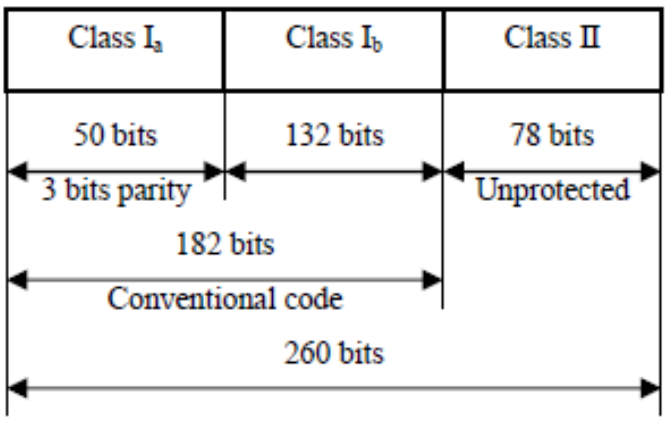


Figura 5.6: Struttura del frame RPE-LTP

La classe I_a contiene 50 bit che sono molto importanti per la qualità della voce. Errori in questi bit possono generare un elevato livello di rumore ed è quindi necessario che questa classe venga protetta con una codifica a doppio canale. I 50 bit della classe I_a vengono anche protetti con tre bit CRC per il rilevamento di errori. I bit ottenuti, assieme ai 132 bit della classe I_b, sono protetti con un codice convoluzionale con rapporto 1/2. Nel ricevitore, se viene rilevato un errore nella casse I_a, i 50 bit vengono sostituiti con una stima ottenuta dall'estrapolazione dei blocchi precedenti. I rimanenti 78 bit della classe II non sono considerati molto importanti, per cui non è necessario proteggerli.

5.4 Il problema dell'eco acustico

Durante gli esperimenti è nato però un grave problema relativo all'eco. L'eco nasce dall'ingresso nel microfono dell'audio emesso dalle casse e quindi proveniente dall'altro utente. Ciò significa che un utente, dopo aver parlato, si sente tornare indietro la propria voce e ciò può causare problemi d'interpretazione nonché addirittura oscillazioni del segnale che si autosostengono fino ad arrivare all'instabilità dell'intero sistema. È necessario quindi riuscire a inserire un cancellatore d'eco per eliminare questo problema.

5.4.1 Cancellatore d'eco

Il sistema, come mostrato in figura 5.7 è costituito da un *parlatore lontano* che è infastidito dall'ascoltare l'eco della propria voce, un *parlatore vicino* che si trova nel terminale ricevente caratterizzato da un determinato comportamento acustico e da una coppia microfono-altoparlante. Il compito del cancellatore d'eco è di rimuovere dal segnale catturato dal microfono la voce del parlatore lontano. Si nota come

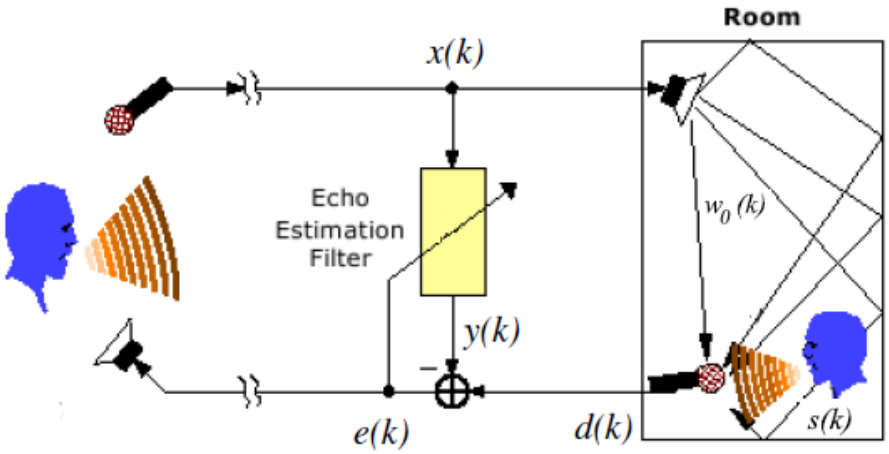


Figura 5.7: Schema di principio del cancellatore d'eco

sia necessario andare ad analizzare il segnale ricevuto e quello trasmesso, ricavando

dal loro confronto un "eco stimato", che viene sottratto al segnale da trasmettere in maniera tale da eliminare l'eco prima della codifica del segnale. La stima deve avvenire in modo sufficientemente accurato.

In figura 5.7 sono indicati:

- $x(k)$ = segnale ricevuto e $s(k)$ = segnale utile da trasmettere,
- $\omega_0(k)$ = eco effettivo,
- $d(k)$ = segnale affetto da eco,
- $y(k)$ = eco stimato,
- $e(k)$ = segnale corretto dalla stima.

Vediamo quindi che il segnale affetto da eco d(k) sarà dato dalla somma $d(k) = s(k) + \omega_0(k)$, mentre il segnale corretto dalla stima che interessa a noi sarà dato da

$$e(k) = d(k) - y(k) = s(k) + R(k)$$

La differenza *R(k)* è nota come "eco residuo" e lo scopo è quello di renderla minima, in modo da ottenere $e(k) \simeq s(k)$.

Si osservi innanzi tutto che il canale d'eco è certamente lineare e praticamente invariante nel tempo per le frazioni di tempo in cui viene preso in considerazione. Quindi è rappresentabile da una risposta impulsiva, supposta incognita, $\omega_0(k) = x(k) \otimes c^\star(k)$, dove $c^\star(k)$ rappresentano i coefficienti della risposta impulsiva corrispondente al cammino acustico nella stanza.

Poiché x(k), a differenza di $c^\star(k)$, è noto, il problema del cancellatore d'eco è quello di determinare, in modo adattativo, l'eco stimato in funzione di x(k) e del segnale ricevuto d(k).

Possibile soluzione

Abbiamo detto quindi che il nostro scopo è quello di cercare di rendere $e(k) \simeq s(k)$. Si è pensato quindi, di inserire due componenti *Grabber* rispettivamente uno dopo il filtro di sorgente RTP per quanto riguarda la ricezione (x(t)) e uno prima del codificatore per quanto riguarda la trasmissione (y(t)). Grazie a questi due elementi è possibile andare, all'avvenuta di un evento, a prelevare il contenuto del buffer presente in quel momento all'interno del Grabber. Sappiamo che il segnale presente nei buffer è il segnale voce codificato PCM in stereo e con 44100 campioni per secondo. Il valore tipico di x(t) risulta di 22050 questo perché il segnale registrato da un microfono è in mono, quindi per trasformarlo in un segnale stereo è sufficiente copiare lo stesso segnale in entrambi i canali, perciò arriva solo la parte mono che poi deve essere trasformata in stereo mediante una trasformazione del tipo "$ABC \Rightarrow AABBCC$". Abbiamo quindi ricostruito una parte del segnale x(t) pari a circa 5 secondi.

Per motivi di tempo non siamo stati in grado di implementare l'algoritmo, però una possibile soluzione è la seguente. Poiché il comportamento acustico del sistema è in generale tempo-variante, le tecniche numeriche utilizzate per realizzare un cancellatore d'eco si basano su un filtraggio adattativo, e tipicamente, per motivi di

semplicità ed efficienza, viene impiegato l'algoritmo LMS (*Least-Mean-Square*) [1]. L'idea è quella di utilizzare il blocco LMS per identificare un sistema incognito, che nella fattispecie corrisponde al cammino acustico tra l'altoparlante e il microfono del terminale di ricezione. Più precisamente il suo compito è di rimuovere, nel terminale ricevente, dal segnale catturato dal microfono quella porzione di segnale che presenta correlazione con il segnale prodotto dal parlatore lontano, ovvero il segnale x(k) convoluto con la risposta impulsiva g(k) corrispondente al cammino acustico fra l'altoparlante e il microfono.

Il cancellatore deve pertanto emulare il comportamento filtrante della stanza, generando a partire da x(k) e dal segnale errore e(k), una stima di d(k), quindi sottrarla dal segnale microfono realizzando di fatto la cancellazione dell'eco come da figura 5.8. Affinché il filtro adattativo possa riprodurre la risposta impulsiva del

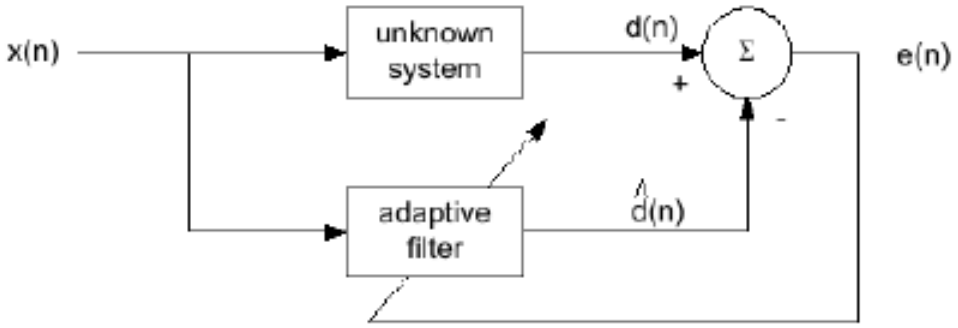


Figura 5.8: Cancellatore d'eco come emulatore di un sistema sconosciuto

cammino acustico, l'ordine del filtro deve essere comparabile con quello della risposta impulsiva desiderata. La conseguenza principale di questo fatto è che la lunghezza del filtro adattativo cresce proporzionalmente al massimo ritardo che si desidera coprire, pertanto se ci si trova in ambienti con tempi di riverbero significativi la lunghezza richiesta al filtro potrebbe essere proibitiva dal punto di vista computazionale.

L'algoritmo LMS, detto algoritmo *del gradiente*, è un algoritmo a bassa complessità computazionale e questo è molto utile per il nostro tipo di applicazione.

Il blocco d'implementazione per l'algoritmo LMS è mostrato in maggiore dettaglio in figura 5.9 ed è caratterizzato dai seguenti parametri:

1. N, numero dei coefficienti del filtro
2. $0 < \mu < \frac{2}{statistical\ power\ input\ vector}$

L'uscita del filtro è data da

$$y(t) = \mathbf{x}^T(k)\mathbf{c}(k)$$

Per quanto riguarda le caratteristiche della parte adattativa abbiamo

1. Errore stimato
$$e(k) = d(k) - y(k)$$

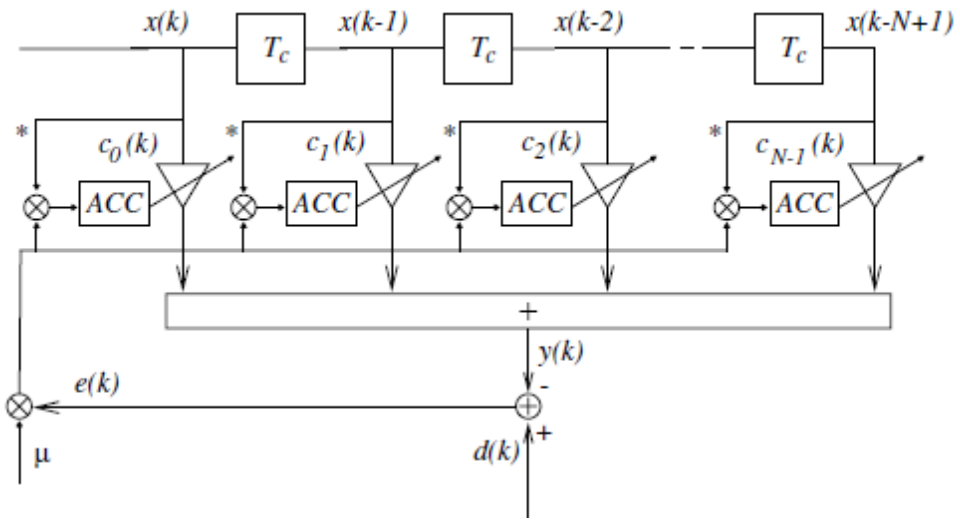


Figura 5.9: Visione più dettagliata del filtro adattativo

2. Vettore coefficiente adattativo

$$\mathbf{c}(k+1) = \mathbf{c}(k) + \mu e(k)\mathbf{x}^*(k)$$

Le condizioni iniziali se non ci sono informazioni disponibili saranno

$$\mathbf{c}(0) = \mathbf{0}$$

Gli accumulatori (ACC) in figura 5.9 sono usati per memorizzare i coefficienti, che sono aggiornati dal valore corrente di $\mu e(k)\mathbf{x}^*(k)$.

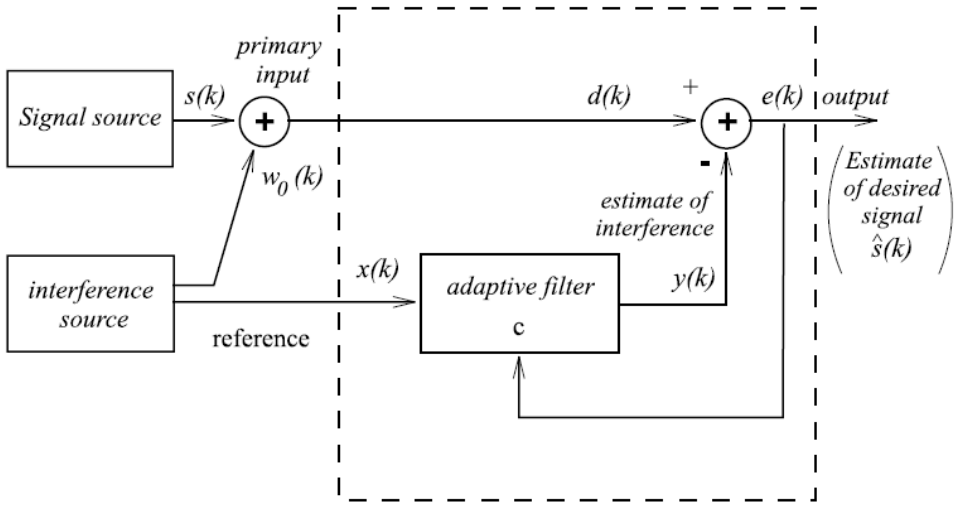


Figura 5.10: Schema dell'uso del filtro adattativo nel caso di cancellatore d'eco

Abbiamo quindi sostanzialmente che, come in figura 5.10, consideriamo:

1. *Input primario*, il segnale desiderato s corrotto dal rumore ω_0,

$$d(k) = s(k) + \omega_0(k)$$

2. *Input di riferimento*, il segnale x, con $s \perp x$.

x è filtrato con un filtro adattativo con coefficienti $\{c_i\}$, $i = 0, 1, ..., N-1$, quindi l'uscita si ottiene da

$$y(k) = \sum_{i=0}^{N-1} c_i(k)x(k-i)$$

che è la miglior replica di ω_0. Definendo l'errore

$$e(k) = d(k) - y(k) = s(k) + \omega_0(k) - y(k)$$

il costo della funzione diventa

$$J = E[e^2(k)] = E[s^2(k)] + E[(\omega_0(k) - y(k))^2]$$

risulta quindi che

$$\min_c J = r_s(0)$$

con $e(k) = s(k)$

Si va in questo modo a trovare il vettore dei coefficienti c necessari per eliminare l'eco. Questa operazione va effettuata in maniera periodica per poter permettere l'aggiornamento del filtro adattativo.

Per quanto riguarda la complessità noi abbiamo a ogni iterazione $2N + 1$ moltiplicazioni e $2N$ addizioni. Quindi l'algoritmo LMS ha una complessità di $O(N)$.

6 Video

In questo capitolo si andrà a descrivere cos'è un segnale video e quali sono le sue principali caratteristiche, perché si usa la compressione e le principali tecniche di codifica di un codec video. Sarà poi spiegata la scelta fatta per il codec.

Un segnale video digitale consiste in una sequenza di frame video, ciascun frame è fatto da un insieme di valori numerici che rappresentano i campioni dell'immagine originaria.

Una sequenza video di buona qualità richiede uno spazio per la memorizzazione, o equivalentemente un bit-rate per la trasmissione, molto elevato. Si pensi ad esempio ad immagini con una risoluzione spaziale pari a 720×576 pixel e quantizzate a 8 bit/pixel, nel caso di un frame-rate pari a 25 frame/s, la trasmissione del segnale richiede un bit-rate r pari a: $r = 720 * 576 * 8 \simeq 80 Mbit/s$.

Un valore questo troppo elevato da rendere improponibile la trasmissione del segnale, ma anche la sua memorizzazione risulterebbe difficoltosa, infatti un solo minuto della sequenza richiederebbe uno spazio pari a 600 MByte. Questi problemi rendono indispensabile il ricorso a tecniche di codifica video che forniscono una rappresentazione efficiente del segnale permettendo un risparmio delle risorse richieste.

6.1 Ridondanza

Ogni segnale presenta un certo grado di ridondanza al suo interno, quindi riducendo questa ridondanza nei domini del tempo, dello spazio e/o della frequenza si ottiene una rappresentazione più compatta del segnale originario.

La ridondanza è essenzialmente di due tipi:

ridondanza statica : spesso il valore di un campione è molto simile a quello dei campioni che gli sono vicino, in altre parole ogni nuovo campione mostra una certa dipendenza da quelli vicini e quindi è predicibile a partire da essi;

ridondanza psicofisica : tipicamente il segnale video contiene molte più informazioni di quelle che un osservatore può apprezzare, ad esempio il sistema visivo umano è poco sensibile alle elevate frequenze spaziali in modo particolare

per quanto riguarda i colori. Quindi è possibile eliminare queste informazioni superflue mantenendo la stessa qualità soggettiva.

Nella classificazione appena fatta non si distingue tra campioni appartenenti ad uno stesso frame o a frame diversi, e infatti si usa distinguere fra ridondanza *intraframe*, presente tra campioni vicini (spazialmente) di uno stesso frame e ridondanza *interframe*, che riguarda campioni vicini (temporalmente) di frame contigui.

6.2 La perdita d'informazione

L'operazione di codifica può essere o meno invertibile, dove con invertibilità intendiamo la possibilità di ricostruire esattamente l'immagine di partenza. Quindi si può distinguere tra algoritmi di compressione *lossy* (con perdita d'informazione e quindi non invertibili, ma capace di ridurre il flusso video anche di 20 volte) e *lossless* (senza perdite, in questo caso è possibile ricostruire esattamente il segnale originale).

Un parametro che permette di confrontare i vari tipi di algoritmi di codifica è il rapporto di compressione, che può essere definito come il rapporto fra il tasso del segnale originario e quello del segnale codificato. Quanto maggiore è questo valore, tanto maggiore sarà la riduzione ad esempio del bit-rate del segnale codificato rispetto a quello del segnale originario. È chiaro che per gli algoritmi lossy, questo parametro non è sufficiente a caratterizzare le prestazioni di codifica. In questo caso è necessario quantificare la perdita di qualità, e questo tipicamente viene fatto attraverso misure oggettive della distorsione come ad esempio l'errore quadratico medio o il rapporto segnale-rumore di picco.

Per quanto sofisticate possano essere, non è detto che le misure oggettive della distorsione diano una buona indicazione della qualità dell'immagine ricostruita. Specialmente nel campo della codifica video, non è difficile trovare situazioni in cui immagini caratterizzate dalla stessa qualità oggettiva abbiano una qualità soggettiva molto diversa. Per questo motivo spesso alle misure oggettive si affiancano quelle soggettive in modo da dare una valutazione più precisa delle prestazioni del codificatore.

6.3 Tecniche di compressione

In generale gli algoritmi di codifica vengono costruiti mettendo insieme più strategie, al fine di soddisfare i requisiti del progetto cercando di riutilizzare quanto più possibile risultati che sono già consolidati. Di seguito sono illustrate alcune tra le più diffuse tecniche di compressione.

6.3.1 Codifiche intraframe

Vediamo ora le tecniche di codifica intraframe.

Codifica con quantizzazione vettoriale

La prima che andiamo a vedere è una tecnica che ci permette di rappresentare un'immagine usando solo un numero limitato di "tasselli" predefiniti. Questo è il concetto che sta alla base della quantizzazione vettoriale, che altro non è che l'estensione della più nota quantizzazione scalare al caso multidimensionale. In pratica l'immagine viene suddivisa in blocchi di dimensioni $M \times N$ e ciascun blocco viene confrontato con un numero limitato di blocchi di riferimento (*codebook*). Tra questi si sceglie quello che meglio rappresenta il blocco originale e ne viene trasmesso l'indice che lo identifica all'interno del codebook. Essendo il numero di blocchi di riferimento limitato, molto inferiore rispetto a quelle che sono tutte le possibili configurazioni di blocchi $M \times N$, trasmettere l'identificativo di tale blocco è meno dispendioso di trasmettere l'intero blocco originale. La quantizzazione vettoriale è una tecnica molto generale con la quale si riesce a sfruttare bene la dipendenza statistica presente tra i pixel contigui, tuttavia la sua complessità computazionale cresce notevolmente al crescere delle dimensioni dei blocchi e questo impedisce di ottenere rapporti di compressione elevati. Per questo motivo spesso se ne usano versioni "vincolate" in cui, a patto di vincolare opportunamente la struttura del codice e la strategia di ricerca, si riesce a ottenere una complessità computazionale e una richiesta di memoria accettabili.

Codifica con trasformata

Nella codifica con trasformata l'ingresso è diviso in blocchi che vengono poi sottoposti ad una trasformazione lineare. Lo scopo della trasformazione è quello di rappresentare il blocco in un dominio trasformato nel quale i campioni di uscita siano indipendenti (o quanto meno abbiano una bassa correlazione). Un'altra caratteristica è la compattazione dell'energia in pochi campioni di uscita. A valle della trasformazione la maggior parte dell'energia sarà contenuta in pochi campioni, rendendo di fatto trascurabili gli altri. Tornando nel dominio di partenza, l'errore commesso non trasmettendo le componenti a bassa energia si distribuisce su tutto il blocco divenendo meno visibile. È possibile dimostrare che la trasformata ottima secondo i due criteri di cui sopra è la trasformata di Karhunen-Loeve che tuttavia è di notevole complessità computazionale. Nella pratica vengono utilizzate trasformazioni sub-ottime che però hanno complessità computazionale molto più bassa. La DCT (Discrete Cosine Transform) è una delle più utilizzate, anche perché data la particolare forma della funzione di correlazione delle immagini, le sue prestazioni si avvicinano molto a quelle della trasformata ottima e inoltre esiste un algoritmo per il calcolo veloce simile alla FFT (Fast Fourier Transform), utilizzata per il calcolo della trasformata di Fourier. La codifica con trasformata è forse ancora le tecnica più utilizzata nella codifica video, offre un'elevata qualità abbinata a un ottimo rapporto di compressione, tuttavia anche in questo caso la complessità computazionale è discretamente elevata.

Codifica predittiva

La codifica predittiva, invece, sfrutta la ridondanza statica esistente tra i campioni. Quindi invece di trasmettere i campioni viene trasmessa la differenza tra il campione

attuale e il campione predetto sulla base dei campioni precedenti. A causa della dipendenza statistica l'errore sarà molto contenuto e quindi è possibile rappresentarle l'immagine con un numero minore di bit rispetto al campione, riducendo in questo modo il bit-rate.

Codifica per sintesi

Nella codifica per sintesi si cerca di estrarre dall'immagine delle informazioni che permettano al decodificatore di sintetizzare l'immagine, ad esempio le informazioni possono riguardare la posizione di alcuni punti caratteristici di un volto. In questo modo si riesce a ottenere una compressione molto spinta, ma chiaramente tali tecniche richiedono un'elevata complessità di calcolo.

6.3.2 Codifiche interframe

Per quanto riguarda la codifica interframe si possono individuare principalmente due tecniche, il ***movement compensation*** e il ***conditional replenishment***.

Codifica movement compensation

Nel caso di codifica movement compensation si sfrutta il fatto che spesso nelle sequenze in movimento in pratica si hanno dei blocchi di immagine che compiono delle traslazioni, e per ognuno di essi invece che ritrasmettere tutto il blocco di dati, si può semplicemente trasmettere la traslazione corrispondente.

Codifica conditional replenishment

Nel caso del conditional replenishment invece si suddivide l'immagine in tanti blocchi, ogni blocco viene poi confrontato con quello che occupa la stessa posizione nel frame precedente (o comunque in un altro frame utilizzato come riferimento) e il blocco viene trasmesso solo se risulta cambiato (tipicamente si confronta l'energia della differenza dei due blocchi con un opportuno valore di soglia).

Tra i due codec interframe quello che offre il maggiore rapporto di compressione è sicuramente il primo, tuttavia richiede anche una complessità computazionale nettamente superiore.

Molto spesso comunque gli algoritmi di codifica vengono costruiti combinando alcune di queste tecniche, soprattutto quelle intraframe con quelle interframe.

6.4 La scelta fatta in questo lavoro

Per la nostra applicazione il codec che si andrà a utilizzare deve risentire il meno possibile delle limitazioni che caratterizzano il singolo utente, sia in termini di capacità del canale che di potenza di calcolo.

Nel nostro caso, poiché si tratta di un software in fase di sviluppo, siamo andati a testare i principali codec video che possono essere trovati free in commercio e tra questi abbiamo scelto quello che ci sembrava il più adatto. Ne sono stati testati una

decina e su questi è stata fatta una prova di bit-rate e poi in maniera soggettiva si è andati a valutare la qualità dell'immagine ricevuta.

Vediamo in tabella 6.1 il bit-rate rilevato dai codec testati.

Codec	Bit-rate [Mbps/s]
H.264 Encoder DMO	0,07
VP70 General Profile	0,15
Emuze MPEG-4	0,16
Emuze H.263	0,17
ffDShow codec	0,23
ffDShow Encoder	0,26
XVID MPEG-4	0,30
Pic Video M-JPI63	0,48
Microsoft REF	0,52

Tabella 6.1: Codec e relativo bit-rate

In questo primo test si è andati a verificare l'occupazione di banda da parte di un singolo utente nell'invio del proprio video. Nella seconda parte del test siamo andati appunto ad osservare la qualità del test in ricezione e tra i vari codec, quello che ci sembrava soddisfare maggiormente è stato il VP70 della *On2 Technologies*. L'On2 technologies afferma che ha una compressione ed un rapporto segnale/rumore migliore dei vari MPEG-4 e H.264.

Naturalmente questo codec non è perfetto e necessita di una giusta quantità di luce nella stanza per non avere effetti di sgranamento dell'immagine, però per l'uso che ne dobbiamo fare ora va molto bene.

VP70 TrueMotion

Il codec VP70 sfrutta la tecnologia *TrueMotion*. Il TrueMotion mantiene la struttura degli ultimi dati decompressi per utilizzarla come fattore predittivo per il frame corrente. La struttura che viene tenuta in memoria si chiama "golden frame". Un primo utilizzo di questa tecnica la si ha quando c'è una persona davanti a uno sfondo statico. Il codec è in grado di rilevare lo sfondo e di mantenerlo ad un'elevata qualità nonostante movimenti veloci in primo piano e senza appesantire la trasmissione.

Questa tecnica viene sfruttata anche in caso di perdita di pacchetti. Infatti nel momento in cui il ricevente si trova senza un pacchetto, il mittente può inviare solo i riferimenti al golden frame snellendo così la trasmissione e avendo di conseguenza immagini più fluide.

Per potersi meglio adattare a tutti i tipi di connessione, questo codec sfrutta quattro livelli di sclabilità temporale come mostrato in figura 6.1. Ciò significa che partendo dalle velocità più basse va ad aggiungere il livello successivo solo se la connessione lo permette. Questo non provoca eventuali costi aggiuntivi per la CPU.

Il golden frame viene anche utilizzato in caso di movimenti molto lenti o nei casi di zoom mantenendo l'alta qualità delle immagini.

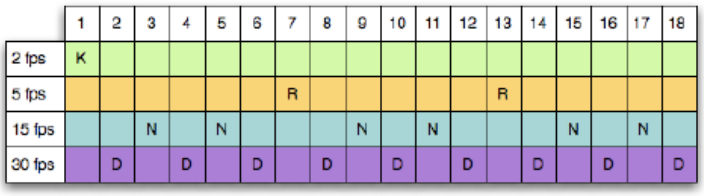


Figura 6.1: Divisione dello stream in quattro livelli

Per applicazioni real-time, come nel nostro caso, il VP70 codifica ogni fotogramma e regola automaticamente la sua complessità, grazie ad algoritmi adattativi, per garantire la migliore qualità con il numero di cicli che ha a disposizione.

Gli algoritmi sono in grado di determinare quali vettori e modi di movimento sono i più adatti, permettendo così di sfruttare solo quelli e annullare l'errore.

7 Il software

In questo capitolo si andrà ad analizzare la struttura del software relativo alla videoconferenza che si è andati a sviluppare, presente in Appendice A, e della gestione dei dati acquisiti dai sensori.

7.1 Struttura della videoconferenza

Il software di videoconferenza può essere diviso sostanzialmente in due parti, una parte relativa alla trasmissione e una parte relativa alla ricezione. C'è poi un'altra parte che invece gestisce l'interfaccia grafica e i vari filtri. Di seguito vedremo come sono strutturate la parte di trasmissione e quella di ricezione.

7.1.1 Trasmissione

Per prima cosa, guardando il codice, si può vedere che si va a caricare la libreria DirectShow di cui verranno sfruttate fin da subito le sue principali caratteristiche esposte nel capitolo 3, tra le quali la creazione del grafo di acquisizione.

In figura 7.1 è possibile osservare lo schema a blocchi relativo alla parte di trasmissione del sistema. Come si può osservare questo schema a blocchi rappresenta

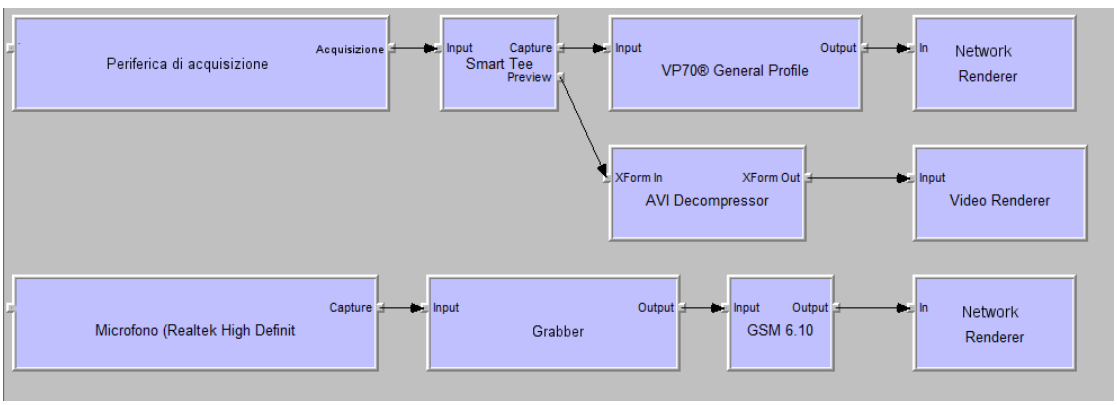


Figura 7.1: Schema a blocchi della parte di trasmissione

sia la parte video, che la parte audio. Questo perché si vuole che sia lo stesso Graph Manager a gestire l'intero grafo che comprende sia il video che l'audio in modo da mantenere la sincronizzazione tra i due, ma allo stesso tempo si vuole che i due flussi possano essere gestiti anche indipendentemente nel caso ci sia la necessità di interrompere l'uno o l'altro.

Video

Per quanto riguarda il video vediamo che la struttura principale è costituita da 6 blocchi. Il primo rappresenta la semplice periferica di acquisizione video che può essere integrata nell'apparecchio oppure può essere aggiunta all'esterno, dovendo però in questo caso stare attenti a modificare il valore assegnato nel codice relativo alla scelta della sorgente video.

Il secondo blocco rappresenta uno *smart tee*. Questo blocco è necessario poiché è nata la necessità di dover sdoppiare il flusso video. Questo per permettere, da una parte di riuscire mandare in trasmissione il video verso il paziente, mentre dall'altra di poter osservare il proprio video in una finestra di dimensioni minori in modo tale da poter eventualmente dare anche dei punti di riferimento al paziente.

Una volta avvenuto lo sdoppiamento del segnale video quindi, da un lato avviene la decompressione AVI e successivamente la proiezione diretta su schermo per il motivo che è stato appena descritto, dall'altro viene inviato al compressor VP70, descritto nel paragrafo 6.4. Nel momento in cui si va a selezionare il codec da utilizzare, bisogna stare attenti al codice perché bisogna anche andare a selezionare il tipo di formato Media che avremo all'uscita, questo per non causare incompatibilità tra i vari blocchi.

Una volta che è stato ottenuto il video compresso nel formato desiderato, si potrà inviare il flusso di dati al filtro che andrà a gestire poi la trasmissione di questi nella rete.

Quest'ultimo filtro, denominato "rendFilter", assieme al filtro posto in ricezione, si occuperà di creare la sessione RTP. È qui infatti che viene impostato l'rtpSender contenente le informazioni sul video, e quindi sulla gestione dei pacchetti che dovrà essere attuata da RTP, nonché in questo filtro si andranno a fissare i valori relativi ai nomi assegnati ai partecipanti, all'indirizzo IP e alla porta di comunicazione alla quale ci si vuole andare a collegare.

Audio

Per quanto riguarda l'audio invece vediamo che la struttura è molto simile, infatti possediamo anche in questo caso un blocco relativo alla periferica di acquisizione audio che, anche in questo caso, è possibile selezionare con la modifica di un semplice valore relativo alla sorgente.

Lo smart tee in questo caso non è necessario, poiché non ci può essere di nessuna utilità ascoltare l'audio che inviamo e creerebbe anzi solo dei problemi di comprensione. Si osserva invece che è stato inserito un oggetto Grabber. Questo oggetto messo lì e basta non fa sostanzialmente niente, lui prende il flusso di dati che gli arriva, lo mette in un buffer e poi li ributta fuori in uscita. In realtà questo oggetto ha una grande potenzialità, infatti ridefinendo i metodi contenuti al suo interno è

possibile andare a modificare i dati che lo attraversano andando appunto a gestire i buffer che vengono in esso trattati.

L'utilità di questo oggetto inserito in questo posto nasce dalla necessità di voler inserire un cancellatore d'eco, come descritto nel paragrafo 5.4. Questo oggetto infatti, assieme a quello inserito nella parte di ricezione che vedremo dopo, andrà a costituire uno degli elementi fondamentali per la futura costruzione del cancellatore d'eco. All'interno di questo oggetto si andrà a prelevare il flusso di dati che lo attraversa e operando su di esso, si andrà a eliminare la parte d'eco che risulta presente nel segnale da trasmettere e che verrà opportunamente calcolata grazie al filtro adattativo.

Una volta che il flusso di dati è stato filtrato dal cancellatore d'eco ed esce da questo va a finire nel blocco compressor "GSM 6.10", descritto nel paragrafo 5.3. A questo punto la voce, che è stata ripulita e compressa, è pronta per essere inviata all'altro utente grazie al filtro per la renderizzazione in rete, denominato "rendFilterA" che ha le stesse caratteristiche del corrispettivo filtro per il video e che usa la stessa connessione creata in precedenza, ma cambiando solo le caratteristiche di rtpSender poiché in questo caso trattiamo l'audio.

7.1.2 Ricezione

Si può notare che la parte di ricezione non parte in automatico con l'avvio del form, come accadeva con la trasmissione, ma come è logico resta in attesa di un evento che la faccia partire.

In figura 7.2 è possibile osservare lo schema a blocchi relativo alla parte di ricezione del sistema.

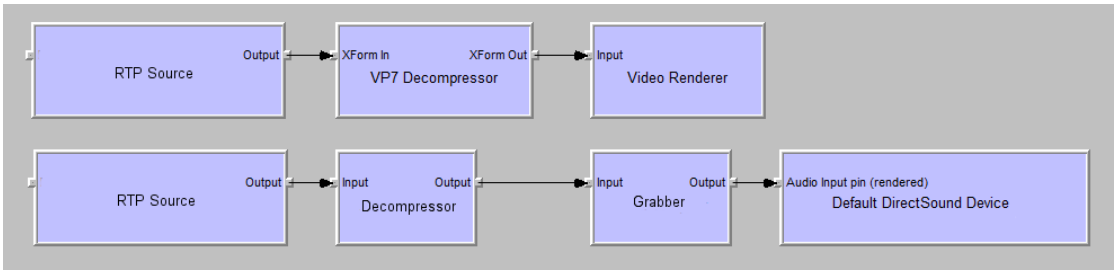


Figura 7.2: Schema a blocchi della parte di ricezione

Anche questo schema risulta essere molto simile, ma ovviamente simmetrico, rispetto a quello di trasmissione, nonché molto simile nelle due configurazioni per l'audio e per il video. In questo caso il blocco che identifica la sorgente è costituito da un filtro "rtpSource", uno per il video e uno per l'audio, che ha lo scopo di andare a prelevare lo stream di dati proveniente dall'utente sorgente e presente nella sessione RTP precedentemente instaurata permettendo così sia l'invio che la ricezione dei dati.

Successivamente i dati vengono decompressi con la stessa categoria di filtro usato nella parte di trasmissione ma dedicato alla decompressione e poi, per il sistema video, abbiamo che lo stream di dati viene renderizzato nell'interfaccia video predisposta.

Per quanto riguarda lo stream dati audio invece, prima di essere indirizzato alla periferica più adatta, lo facciamo passare per il Grabber. Si tratta questo del secondo Grabber di cui si è parlato nel paragrafo 5.4, relativamente al cancellatore d'eco. Dal buffer contenuto in questo oggetto infatti verranno prelevati i dati che, salvati, andranno a ricostruire una parte del segnale peri a circa 5 s, e serviranno per essere elaborati dal filtro adattativo come segnale di riferimento.

7.2 Gestione del flusso dati

In questa sezione si andrà a vedere come si è andati a gestire il flusso dei dati acquisiti dal VRRS in real time contenenti le informazioni relative ai sensori e quindi la posizione dell'arto durante la fase di analisi. In questa fase di trasmissione delle informazioni acquisite è necessario andare a limitare la quantità di dati inviati in modo da non saturare la banda disponibile, adattando quindi il throughput di dati, ma allo stesso stesso tempo bisogna stare attenti a non ridurlo troppo perché ciò non permetterebbe di andare a rilevare in maniera precisa la posizione dell'arto e l'analisi dei dati da parte del medico sarebbe compromessa.

Anche in questa fase di programmazione ci troviamo nella condizione di necessitare di un *Server* che andrà a inviare i dati e un *Client* che andrà a ricevere i dati per poi poterli elaborare. I dati acquisiti dal VRRS sono raccolti con una frequenza di campionamento di 240Hz per sensore, con un massimo di 16 sensori. Per la nostra applicazione però possiamo accettare fino a 30 campioni al secondo senza che si vada a deteriorare troppo la qualità delle informazioni. I dati che ci arrivano sono strutturati in modo da avere una serie di informazioni sulla posizione del sensore rispetto ad un punto di riferimento. Abbiamo quindi 6 valori di tipo *float* che ci indicano le coordinate del sensore e un valore che ci indica il numero del sensore preso in considerazione.

Relativamente al protocollo di comunicazione da utilizzare in questa parte di software per il trasferimento dei dati si è scelto di sfruttare le potenzialità di UDP. Come spiegato nel capitolo 4 il protocollo TCP, ogni volta che invia un pacchetto, aspetta sempre l'acknowledgment prima di inviare il pacchetto successivo, mentre per come è stato progettato UDP non gli interessa se il messaggio è arrivato a destinazione oppure no. A noi infatti, una volta trasmesso il frame, se non arriva non ci interessa che venga rispedito poiché sarebbe inutile e porterebbe solo a rallentare il flusso di dati in ricezione oltre al fatto che non rispecchierebbe la vera posizione dell'arto sotto analisi in quel preciso istante ma si riferirebbe a istanti precedenti.

Andiamo a vedere in dettaglio come sono strutturate rispettivamente la classe Client e la classe Server per come sono state costruite.

7.2.1 Client

La classe Client è quella classe che sarà utilizzata dal VRRS.Client. Sarà quindi posizionata nello studio medico e il suo scopo principale sarà quello di ricevere le informazioni relative ai sensori e di riuscire a gestirle.

Per prima cosa questa classe fa in modo che possa avvenire la connessione. Infatti partendo dalla condizione che il Server si trova in attesa, quindi in ascolto, di una

connessione, il Client deve andare a fare una richiesta di connessione al Server di cui conosce l'IP e la porta in cui è in ascolto. Bisogna però fare i conti col fatto che ora il Server deve sapere a chi deve inviare quei dati e durante questa prima fase di comunicazione il Client deve informare il Server dell'indirizzo IP e della porta in cui starà in ascolto per la ricezione dei dati.

Poiché stiamo usando UDP però, essendo questo protocollo non orientato alla connessione, ci troviamo a dover inserire un metodo che permetta di verificare se il Server è ancora attivo. Per far ciò si è pensato di sfruttare la classe Ping. Questa classe offre funzionalità simili allo strumento della riga di comando Ping.exe e invia messaggi di richiesta echo ICMP a un computer remoto attendendo un messaggio di risposta echo ICMP da tale computer. In caso questa verifica avesse un esito negativo per più di un certo numero di volte, si andrà a comunicare a schermo che il Server remoto non è più raggiungibile.

All'interno della classe Client è presente anche il metodo FrameReceived. Questo metodo viene richiamato ad ogni evento di ricezione e serve per la gestione dei dati. In questo metodo, alla ricezione dei dati, si va a controllare per prima cosa se si tratta di dati riguardanti la posizione dei sensori o se si tratta di un blocco di dati inviato dal Server per controllare il traffico. Questo blocco è riconoscibile perché è caratterizzato da un numero identificativo. A questo punto, se si tratta di un sensore si andrà a salvare l'informazione in una determinata tabella che sarà usata poi dal VRRS.Client per rappresentare il sensore nella realtà virtuale, se invece si tratta di un'informazione di controllo, il metodo andrà a generare una risposta che sarà utilizzata dal Server come vedremo poi nella sezione 7.2.2.

7.2.2 Server

La classe Server invece sarà utilizzata dal VRRS.Server. Sarà questa posizionata a casa del paziente e si occuperà della trasmissione dei dati acquisiti dai vari sensori e della gestione del relativo traffico.

Come detto nella sezione precedente, questa classe una volta creata rimane in attesa della connessione da parte di un Client. Poiché il Sever non è a conoscenza dell'indirizzo IP e della porta di ascolto del Client, appena gli arriva una richiesta, sulla sua porta in ascolto, va a estrapolarsi le informazioni necessarie per poter cominciare la trasmissione dei dati relativi alla posizione dei sensori. Nella prima fase di comunicazione che avviene tra il Client e il Server verranno trasmesse anche le informazioni riguardanti il tempo medio della richiesta di echo ICMP effettuata dal Client e che servirà poi per poter valutare la congestione del traffico.

Una volta che il Server è stato in grado di ottenere le informazioni necessarie, inizia con l'invio dei dati che gli arrivano dai sensori, rispettando le condizioni imposte dal metodo che gestisce la congestione del traffico.

In questo metodo appunto si va a verificare se in ricezione i dati arrivano senza eccessivi ritardi. Per far questo si valuta il tempo medio ricavato dal ping e lo si prende come riferimento, considerando naturalmente il tempo di andata, di ritorno e una certa percentuale di accettabilità, e lo si fissa come costante di riferimento. A questo punto ogni 10 secondi si va a inserire nella coda di trasmissione un pacchetto di riferimento caratterizzato da un certo numero identificativo e si fa partire il timer. Il

Client, come spiegato nella sezione 7.2.1, se riceve questo tipo di pacchetto genera una determinata risposta che dovrà essere a sua volta ricevuta entro un tempo massimo dal Server. A questo punto si va a fermare il timer e a confrontare il valore ottenuto con quello di riferimento fissato in precedenza. Effettuato questo confronto si va quindi ad aggiornare la costante relativa alla congestione per determinare la qualità della trasmissione. Questa costante servirà per definire il rapporto di dati che si andranno a inviare a quel determinato Client e che risulterà essere $1 : 1$, se la trasmissione risulta essere di ottima qualità, potendo quindi trasmettere tutti 240 campioni al secondo ricevuti, e si andrà via via calando fino ad arrivare a un rapporto minimo di $1 : 8$, se la trasmissione è molto difficile, inviando quindi il minimo possibile di informazioni pari a 30 campioni al secondo.

8 Conclusioni

Nella prima parte di questo lavoro si è cercato di sviluppare un sistema per la trasmissione di audio e video in tempo reale destinato ad un utenza che non abbia vincoli troppo stringenti, sia per quanto riguarda le caratteristiche della rete di accesso, che per la potenza di calcolo disponibile, nonché per le capacità cognitive dei pazienti stessi.

Nella scelta del tipo di supporto per una trasmissione in real-time si è fatto ricorso al protocollo RTP (che si affida ai servizi UDP) perché è quel protocollo che riesce appunto a gestire meglio le connessioni real-time senza aggiungere ritardi troppo elevati che non sarebbero accettabili dall'utente finale che non sarebbe più in grado di percepire con i giusti tempi le indicazioni del medico.

Purtroppo non è stato possibile verificare il funzionamento di questo protocollo su una rete esterna, però sono state fatte delle prove in una condizione di intranet sia per quanto riguarda l'occupazione di banda di una videoconferenza col software sviluppato, che è risultata mediamente di circa $300kbps$ come si vede anche in figura 8.1, sia per quanto riguarda la robustezza. Relativamente a quest'ultimo fattore si è andati a vedere come si comportava la connessione nel caso di una rete già a pieno carico. Si è provato quindi a caricare la rete con un trasferimento massiccio di file e si è fatta partire, in un secondo momento, la videoconferenza. Si è potuto constatare che all'avvio della conferenza, il bit-rate relativo al trasferimento dei file è calato bruscamente per permettere che la videoconferenza potesse avvenire senza problemi. Successivamente, nel momento in cui è stata interrotta la comunicazione la velocità di trasferimento dei file si è riportata al livello iniziale e cioè al massimo disponibile. Si è inoltre provato a interrompere fisicamente il collegamento tra i due computer e si è visto che, per brevi tempi, la connessione rimaneva instaurata.

Per la scelta dei codec ci siamo fidati, per quanto riguarda l'audio, dell'ormai robusto e testato codec GSM 6.10 che ha portato un'occupazione di banda, relativa alla sola trasmissione, di $13kbps$.

Nella scelta del codec video invece la scelta è stata un po' più difficile perché sono sorti vari problemi. Dopo la prima fase in cui si è andati a testare il bit-rate del flusso video codificato con ogni tipo di codec a disposizione, si è dovuti andare a vedere come si comportava quel determinato codec quando ci spostavamo dal lato della ricezione. I risultati sono stati i più svariati, infatti alcuni codec, che avevano anche un buon rapporto di compressione, in ricezione risultava che l'immagine era

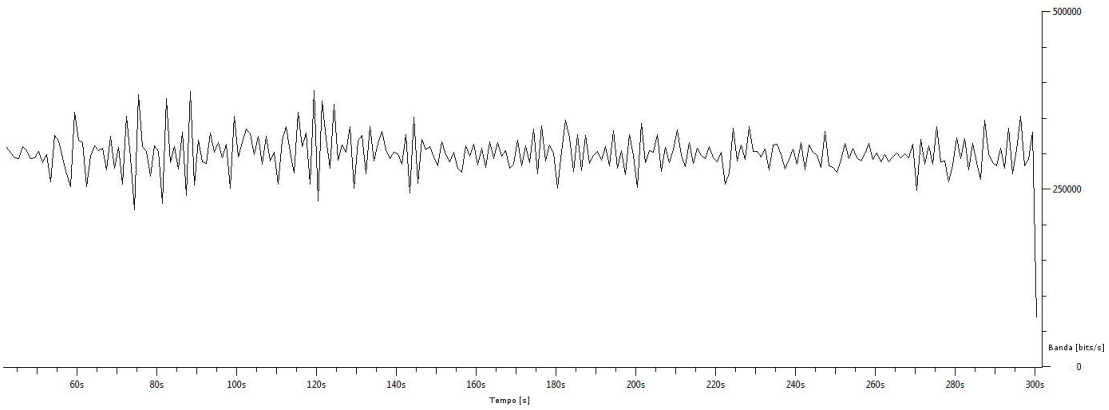


Figura 8.1: Grafico dell'occupazione di banda relativa alla videoconferenza

completamente sgranata, oppure che non veniva visualizzata. Altra nota negativa relativa a questi codec "free" risultata essere la dipendenza elevata dall'hardware. È stato riscontrato infatti che lo stesso codec aveva prestazioni diverse in base al tipo di telecamera utilizzata e alla potenza di calcolo disponibile. Alla fine, cercando di trovare il giusto compromesso si è scelto il codec VP70.

Ci sono ancora alcuni punti su cui lavorare per rendere funzionale l'applicazione realizzata. Uno di questi è proprio il cancellatore d'eco. Una volta inserito l'algoritmo necessario bisognerà stare attenti al fatto che si vanno a introdurre dei ritardi. Sarà quindi necessario eseguire le operazioni di ottimizzazione dei filtri stando ben attenti a non rallentare troppo il sistema che andrebbe a creare degli effetti molto sgradevoli in ricezione.

La seconda parte del lavoro invece si è sviluppata sulla gestione real time dei dati acquisiti dai sensori. Il protocollo scelto per il livello trasporto è stato il protocollo UDP per le sue caratteristiche che si adattavano perfettamente ai nostri scopi. Si è andati quindi a sviluppare due classi, una Server e una Client, utilizzate rispettivamente per l'invio e la gestione della qualità del trasferimento dati per quanto riguarda la classe Server, mentre la classe Client si è occupata della connessione con relativa gestione e della ricezione delle informazioni da comunicare a VRRS.Client. Il problema principale con cui si è avuto a che fare in questa parte è stata la connessione tra server e client e la relativa gestione.

A

Videoconferenza

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Net;
using System.Net.Sockets;
using MSR.LST.Net.Rtp;
using MSR.LST;
using DirectShowLib;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        _senderCB = new SampleGrabberCBClass("SENDER");
        _receiveCB = new SampleGrabberCBClass("RECEIVER");
        _senderCB.SetRic(_receiveCB);
        RtpStream.FirstFrameReceived += new RtpStream.FirstFrameReceivedEventHandler
            (RtpStream_FirstFrameReceived);
        RtpEvents.RtpStreamAdded += new RtpEvents.RtpStreamAddedEventHandler(
            RtpEvents_RtpStreamAdded);
        RtpEvents.RtpStreamRemoved += new RtpEvents.RtpStreamRemovedEventHandler(
            RtpEvents_RtpStreamRemoved);
        Main();
    }

    MSR.LST.Net.Rtp.RtpSession rtpSessione = null;
    IGraphBuilder graphBuilder = null;
    MSR.LST.MDShow.IVideoWindow iVW2 = null;
    IMediaControl mediaControl = null;
    IBaseFilter capFilter = null;
    IBaseFilter capFilterA = null;
    DsDevice[] compressor;

    SampleGrabberCBClass _senderCB = null;
    SampleGrabberCBClass _receiveCB = null;
```

```
    private void Main()
50  {
        //network sender comune
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.150"), 5000);
        MSR.LST.Net.Rtp.RtpParticipant rp = new MSR.LST.Net.Rtp.RtpParticipant("2",
            "1");
        rtpSessione = new MSR.LST.Net.Rtp.RtpSession(iep, rp, true, true);
55      System.Collections.Hashtable prix = new System.Collections.Hashtable();

        //creo graph builder
        this.graphBuilder = (IGraphBuilder)new FilterGraph();
        IFilterGraph2 filterGraph = graphBuilder as IFilterGraph2;
60
        #region Video

        //aggiungo source video
        DsDevice[] dev = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
65      int hr = filterGraph.AddSourceFilterForMoniker(dev[0].Mon, null, dev[0].Name
            , out capFilter);

        //aggiungo copressor video
        IBaseFilter compFilter = null;
        compressor = DsDevice.GetDevicesOfCat(FilterCategory.VideoCompressorCategory
            );
70      hr = filterGraph.AddSourceFilterForMoniker(compressor[18].Mon, null,
            compressor[18].Name, out compFilter);

        IPin p1 = GetPin(capFilter, PinDirection.Output);
        AMMediaType media = null;

75      IEnumMediaTypes enumMT;
        AMMediaType[] mts = new AMMediaType[1];
        p1.EnumMediaTypes(out enumMT);
        while (enumMT.Next(mts.Length, mts, IntPtr.Zero) == 0)
        {
80          media = mts[0];
            VideoInfoHeader videoInfoHeader = (VideoInfoHeader)Marshal.
                PtrToStructure(media.formatPtr, typeof(VideoInfoHeader));
            int videoWidth = videoInfoHeader.BmiHeader.Width;
            int videoHeight = videoInfoHeader.BmiHeader.Height;
            if (media.majorType == MediaType.Video && media.subType == MediaSubType.
                YUY2 && videoWidth == 320)
85          {
                break;
            }
            else
                DsUtils.FreeAMMediaType(media);
90      }
        Marshal.ReleaseComObject(p1);

        //SmartTee
        IBaseFilter smartTee = FilterGraphTools.AddFilterFromClsid(graphBuilder,
            typeof(SmartTee).GUID, "smartTee");
95      IBaseFilter videoRenderer = FilterGraphTools.AddFilterFromClsid(graphBuilder
            , typeof(VideoRenderer).GUID, "VideoRenderer");

        //network sender Video
        MSR.LST.Net.Rtp.RtpSender rtpSender =
            rtpSessione.CreateRtpSender("Christian1", PayloadType.dynamicVideo, prix)
                ;
100
        MSR.LST.MDShow.Renderer renderer =
            (MSR.LST.MDShow.Renderer)MSR.LST.MDShow.Filter.NetworkRenderer();
        ((MSR.LST.MDShow.Filters.IRtpRenderer)renderer.BaseFilter).Initialize(
            rtpSender);

105     IBaseFilter rendFilter = renderer.BaseFilter as IBaseFilter;
```

```
        hr = graphBuilder.AddFilter(rendFilter, renderer.FriendlyName);

        //Connetto i filtri
        hr = ConnectFilter(graphBuilder, capFilter, smartTee, media);
        hr = Connect(filterGraph, smartTee, compFilter);
        hr = Connect(filterGraph, compFilter, rendFilter);
        hr = Connect(filterGraph, smartTee, videoRenderer);

        //inserimento video nel form
        iVW2 = (MSR.LST.MDShow.IVideoWindow)videoRenderer;
        iVW2.Owner = this.picVideo.Handle.ToInt32();
        iVW2.Width = 320;
        iVW2.Height = 260;
        iVW2.Left = 0;
        iVW2.Top = 0;

        #endregion Video


        #region Audio

        //aggiungo source audio
        DsDevice[] devA;
        devA = DsDevice.GetDevicesOfCat(FilterCategory.AudioInputDevice);
        hr = filterGraph.AddSourceFilterForMoniker(devA[0].Mon, null, devA[0].Name,
              out capFilterA);

        //Aggiungo filtro per Echo

        ISampleGrabber sampGrabber = new SampleGrabber() as ISampleGrabber;
        IBaseFilter baseGrabFlt = sampGrabber as IBaseFilter;

        //impostazioni media
        AMMediaType mediaA;
        mediaA = new AMMediaType();
        mediaA.majorType = MediaType.Audio;
        mediaA.subType = MediaSubType.PCM;
        mediaA.formatType = FormatType.WaveEx;
        hr = sampGrabber.SetMediaType(mediaA);
        DsUtils.FreeAMMediaType(mediaA);

        hr = filterGraph.AddFilter(baseGrabFlt, "Grabber");
        hr = sampGrabber.SetCallback(_senderCB, 1);

        //aggiungo "GSM 6.10" come copressor audio

        DsDevice[] compressorA = DsDevice.GetDevicesOfCat(FilterCategory.
              AudioCompressorCategory);
        IBaseFilter compFilterA = null;
        for (int i = 0; i < compressorA.Count(); i++)
        {
            if (compressorA[i].Name == "GSM 6.10")
            {
            hr = filterGraph.AddSourceFilterForMoniker(compressorA[i].Mon, null,
                  compressorA[i].Name, out compFilterA);
            break;
            }
            if (i == compressorA.Count())
            {
            //errore
            }
        }

        //network sender audio
        MSR.LST.Net.Rtp.RtpSender rtpSenderA =
            rtpSessione.CreateRtpSender("Christian1", PayloadType.dynamicAudio, prix
                  );

        MSR.LST.MDShow.Renderer rendererA =
            (MSR.LST.MDShow.Renderer)MSR.LST.MDShow.Filter.NetworkRenderer();
```

```
        ((MSR.LST.MDShow.Filters.IRtpRenderer)rendererA.BaseFilter).Initialize(
            rtpSenderA);

        IBaseFilter rendFilterA = rendererA.BaseFilter as IBaseFilter;
        hr = graphBuilder.AddFilter(rendFilterA, rendererA.FriendlyName);

        //Connetto i filtri

        hr = Connect(filterGraph, capFilterA, baseGrabFlt);
        hr = Connect(filterGraph, baseGrabFlt, compFilterA);
        hr = Connect(filterGraph, compFilterA, rendFilterA);

        #endregion Audio


        mediaControl = graphBuilder as IMediaControl;
        mediaControl.Run();

    }

    #region Gestione filtri

    private int ConnectFilter(IGraphBuilder gb, IBaseFilter f1, IBaseFilter f2,
        AMMediaType mt)
    {

        IPin p1 = GetPin(f1, PinDirection.Output);
        IPin p2 = GetPin(f2, PinDirection.Input);

        return gb.ConnectDirect(p1, p2, mt);
    }

    private int Connect(IGraphBuilder gb, IBaseFilter f1, IBaseFilter f2)
    {

        IPin p1 = GetPin(f1, PinDirection.Output);
        IPin p2 = GetPin(f2, PinDirection.Input);

        return gb.Connect(p1, p2);

    }

    private IPin GetPin(IBaseFilter filter, PinDirection dir)
    {
        IPin ret_pin = null;

        IEnumPins enumPins;
        IPin[] pins = new IPin[1];

        int hr = filter.EnumPins(out enumPins);
        DsError.ThrowExceptionForHR(hr);

        try
        {
            while (enumPins.Next(pins.Length, pins, IntPtr.Zero) == 0)
            {
                try
                {
                    PinDirection pinDir;
                    hr = pins[0].QueryDirection(out pinDir);
                    DsError.ThrowExceptionForHR(hr);
                    PinInfo pinfo;
                    pins[0].QueryPinInfo(out pinfo);
                    IPin pin_con;
                    bool not_connected = ((uint)pins[0].ConnectedTo(out pin_con) ==
                        0x80040209);

                    if (pinDir == dir && not_connected)
                    {
                        ret_pin = pins[0];
```

```
240                         break;
                        }
                        else
                        {
                            Marshal.ReleaseComObject(pins[0]);
245                         }
                    }
                    finally
                    {
                    }
250             }
        }
        finally
        {
            Marshal.ReleaseComObject(enumPins);
255         }

        return ret_pin;
    }

260 #endregion


    #region Ricezione

265 RtpStream rtpStream = null;
    void RtpStream_FirstFrameReceived(object sender, EventArgs ea)
    {
        RtpStream s = sender as RtpStream;
        if (s.Properties.Name != "Christian1")
270     {
            if (s.PayloadType == PayloadType.dynamicVideo)
            {
                if ((rtpStream == null) || (rtpStream.Properties.Name != s.
                    Properties.Name))
                {
275                 //caso prima ricezione
                    RicVid(s);
                    rtpStream = s;
                }
                else
280             {
                    //caso riavvio da remoto
                    iVW.Visible = 0;
                    RicVid(s);
                }
285         }
            else
                if (s.PayloadType == PayloadType.dynamicAudio)
                {
                    RicAud(s);
290             }
        }
    }

    List<RtpStream> _streams = new List<RtpStream>();
295 void RtpEvents_RtpStreamAdded(object sender, RtpEvents.RtpStreamEventArgs ea)
    {
        _streams.Add(ea.RtpStream as RtpStream);
    }

300 void rtpStream_FrameReceived(object sender, RtpStream.FrameReceivedEventArgs ea)
    {
    }

    delegate void _ricezione(RtpStream stream);
305 MSR.LST.MDShow.IVideoWindow iVW = null;

    void RicVid(RtpStream stream)
    {
```

```
        MSR.LST.MDShow.FilgraphManagerClass fgm = null;

        if (this.InvokeRequired)
        {
            _ricezione d = new _ricezione(RicVid);
            this.Invoke(d, stream);
        }
        else
        {
            stream.IsUsingNextFrame = true;
            fgm = new MSR.LST.MDShow.FilgraphManagerClass();
            MSR.LST.MDShow.IGraphBuilder iGB = (MSR.LST.MDShow.IGraphBuilder)fgm;
            MSR.LST.MDShow.IBaseFilter bfSource = MSR.LST.MDShow.RtpSourceClass.
                CreateInstance();
            ((MSR.LST.MDShow.Filters.IRtpSource)bfSource).Initialize(stream);
            iGB.AddFilter(bfSource, "RtpSource");
            iGB.Render(MSR.LST.MDShow.Filter.GetPin(bfSource, MSR.LST.MDShow.
                _PinDirection.PINDIR_OUTPUT, Guid.Empty, Guid.Empty, false, 0));

            //video nel form
            iVW = (MSR.LST.MDShow.IVideoWindow)fgm;
            iVW.Owner = this.picVideo.Handle.ToInt32();
            iVW.Height = picVideo.Height;
            iVW.Width = picVideo.Width;
            iVW.Left = 0;
            iVW.Top = 0;
            iVW.Visible = -1;
            fgm.Run();

            //proprio video sopra al video ricevuto
            mediaControl.Stop();
            iVW2.Visible = 0;
            mediaControl.Run();
            iVW2.Visible = -1;
        }
    }

    MSR.LST.MDShow.FilgraphManagerClass fgmA = null;
    private MSR.LST.MDShow.IBasicAudio iBA = null;
    private int volum = 0;

    void RicAud(RtpStream stream)
    {
        if (this.InvokeRequired)
        {
            _ricezione d = new _ricezione(RicAud);
            this.Invoke(d, stream);
        }
        else
        {
            stream.IsUsingNextFrame = true;
            fgmA = new MSR.LST.MDShow.FilgraphManagerClass();
            MSR.LST.MDShow.IGraphBuilder iGBA = (MSR.LST.MDShow.IGraphBuilder)fgmA;
            MSR.LST.MDShow.IBaseFilter rtpSource = MSR.LST.MDShow.RtpSourceClass.
                CreateInstance();
            ((MSR.LST.MDShow.Filters.IRtpSource)rtpSource).Initialize(stream);
            iGBA.AddFilter(rtpSource, "RtpSource");

            //Aggiungo filtro per Echo
            ISampleGrabber sampGrabber2 = new SampleGrabber() as ISampleGrabber;
            int hr = sampGrabber2.SetCallback(_receiveCB, 1);
            IBaseFilter baseGrabFlt2 = sampGrabber2 as IBaseFilter;

            //impostazioni media
            AMMediaType mediaA;
            mediaA = new AMMediaType();
            mediaA.majorType = MediaType.Audio;
            mediaA.subType = MediaSubType.PCM;
            mediaA.formatType = FormatType.WaveEx;
            hr = sampGrabber2.SetMediaType(mediaA);
```

```
            DsUtils.FreeAMMediaType(mediaA);
            MSR.LST.MDShow.IBaseFilter f2 = (MSR.LST.MDShow.IBaseFilter)baseGrabFlt2
                ;
            iGBA.AddFilter(f2, "Grabber2");

            //connessione rtpSource con f2
            iGBA.Connect(MSR.LST.MDShow.Filter.GetPin(rtpSource, MSR.LST.MDShow.
                _PinDirection.PINDIR_OUTPUT, Guid.Empty, Guid.Empty, false, 0),
                MSR.LST.MDShow.Filter.GetPin(f2, MSR.LST.MDShow._PinDirection.
                PINDIR_INPUT, Guid.Empty, Guid.Empty, false, 0));

            //seleziona Speaker
            MSR.LST.MDShow.FilterInfo[] spk = MSR.LST.MDShow.AudioRenderer.Renderers
                ();
            MSR.LST.MDShow.FilterInfo fi = spk[0];
            iGBA.AddFilter(MSR.LST.MDShow.Filter.CreateBaseFilter(fi), "SpkFilter");

            //Connessione con speaker
            iGBA.Render(MSR.LST.MDShow.Filter.GetPin(f2, MSR.LST.MDShow.
                _PinDirection.PINDIR_OUTPUT, Guid.Empty, Guid.Empty, false, 0));
            iBA = (MSR.LST.MDShow.IBasicAudio)fgmA;
            int volum = (int)Math.Round(Math.Pow(10.0, (2.0 * (double)(iBA.Volume +
                10000)) / 10000.0));
            fgmA.Run();
            isAudioOn = true;

        }
    }

    private object Connect(MSR.LST.MDShow.IGraphBuilder iGBA, MSR.LST.MDShow.
        IBaseFilter rtpSource, IBaseFilter baseGrabFlt2)
    {
        throw new NotImplementedException();
    }

    #endregion

    #region Pulsanti

    private bool isMicrAcive = true;
    private void Micr_Click(object sender, EventArgs e)
    {
        if (isMicrAcive)
        {
            this.Micr.BackColor = Color.Red;
            capFilterA.Pause();
        }
        else
        {
            this.Micr.BackColor = Color.Green;
            capFilterA.Run(0);
        }

        isMicrAcive = !isMicrAcive;
    }

    private bool isVideoAcive = true;
    private void camera_Click(object sender, EventArgs e)
    {
        if (isVideoAcive)
        {
            this.camera.BackColor = Color.Red;
            capFilter.Pause();
        }
        else
        {
            this.camera.BackColor = Color.Green;
            capFilter.Run(0);
        }
        isVideoAcive = !isVideoAcive;
```

```
        }

440     private bool isAudioOn = false;
        private void Audio_Click(object sender, EventArgs e)
        {
            if (isAudioOn)
            {
445             this.Audio.BackColor = Color.Red;
                fgmA.Stop();
            }
            else
            {
450             this.Audio.BackColor = Color.Green;
                fgmA.Run();
            }

            isAudioOn = !isAudioOn;
455     }

        private void SetVolume(int value)
        {
            //settaggio volume in maniera logaritmica
460         iBA.Volume = (volum != 0) ? (int)(Math.Round((10000.0 * Math.Log10((double)
                volum)) / 2.0) - 10000) : -10000;
            if (isAudioOn)
            {
                this.Audio.BackColor = Color.Green;
                fgmA.Run();
465             isAudioOn = !isAudioOn;
            }
        }

        #endregion
470
    }
}
```

B

Client

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Net.NetworkInformation;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;

namespace WindowsFormsApplication2
{
    public class Client
    {
        string _CIPadd = "";
        string _SIPadd = "";
        int _Cport;
        int _Sport;
        string _name = "";
        DataFrames _df;
        DataFrames _test;
        int _count = 0;
        int _ping=1000;
        byte[] buff = new byte[28];
        UdpClient _udpc = null;

        public Client(string ClientAdd, string ServerAdd, int Cport, int Sport,
              string netName)
        {
            _CIPadd = ClientAdd;
            _SIPadd = ServerAdd;
            _Cport = Cport;
            _Sport = Sport;
            _name = netName;
        }
        #region Connessione
        public void StartAcquisition()
        {
            //Creo UdpClient e connetto a _SIPadd su porta _Sport
            IPEndPoint e = new IPEndPoint(IPAddress.Any, _Cport);
            _udpc = new UdpClient(e);
            _udpc.Connect(_SIPadd, _Sport);

            System.Threading.Thread r = new System.Threading.Thread(new ThreadStart(
```

```
                Ricezione));
            r.Start();

            System.Threading.Thread p = new System.Threading.Thread(new ThreadStart(
                Connessione));
            p.Start();

        }

        public void Ricezione()
        {
            while (true)
            {
                int recv = _udpc.Client.Receive(buff);
                FrameReceived(buff);
            }

        }
        public void Connessione()
        {
            while (true)
            {
                Ping ping = new Ping();
                ping.PingCompleted += new PingCompletedEventHandler(
                    PingCompletedCallback);
                ping.SendAsync(_SIPadd,100,null);
            }
        }

        private void PingCompletedCallback(object sender, PingCompletedEventArgs e)
        {
            Object thisLock = new Object();
            lock (thisLock)
            {// Se c'è errore
                if (e.Reply.Status != System.Net.NetworkInformation.IPStatus.Success
                    )
                {
                    _count++;
                    if (_count > 4)
                    {
                        _udpc.Connect(_SIPadd, _Sport);
                        System.Threading.Thread r = new System.Threading.Thread(new
                            ThreadStart(Ricezione));
                        r.Start();
                        _count = 0;
                    }
                }
                else
                {
                    _ping=((int)(e.Reply.RoundtripTime)+10)*2;
                    _count = 0;
                }
        }
        }
        #endregion

        #region gestDati
        void FrameReceived(byte[] data)
        {
            //i dati viaggiano come array di bytes
            _df = new DataFrames(data);
            if (_df.N == 20) //se mi arriva un dataframe con n=20 apro il metodo
                controllo
            {
                System.Threading.Thread t=new System.Threading.Thread(new
                    ThreadStart(Controllo));
                t.Start();
            }
        }
```

```
        void Controllo()
        {   // rispedisco indietro un dataframe con n=21 e x=_ping
            _test = new DataFrames(21, _ping, 0, 0, 0, 0, 0);
            _udpc.Send(ToByte(_test), 28);
        }

        #endregion

        #region Dataframe
        public struct DataFrames
        {
            public float N;
            public float X;
            public float Y;
            public float Z;
            public float Yaw;
            public float Pitch;
            public float Roll;

            public DataFrames(float n, float x, float y, float z, float yaw, float
                  pitch, float roll)
            {
                N = n;
                X = x;
                Y = y;
                Z = z;
                Yaw = yaw;
                Pitch = pitch;
                Roll = roll;
            }

            public DataFrames(byte[] data)
            {
                N = BitConverter.ToSingle(data, 0);
                X = BitConverter.ToSingle(data, 4);
                Y = BitConverter.ToSingle(data, 8);
                Z = BitConverter.ToSingle(data, 12);
                Yaw = BitConverter.ToSingle(data, 16);
                Pitch = BitConverter.ToSingle(data, 20);
                Roll = BitConverter.ToSingle(data, 24);
            }
        }

        public byte[] ToByte(DataFrames data)
        {
            List<byte> result = new List<byte>();
            result.AddRange(BitConverter.GetBytes(data.N));
            result.AddRange(BitConverter.GetBytes(data.X));
            result.AddRange(BitConverter.GetBytes(data.Y));
            result.AddRange(BitConverter.GetBytes(data.Z));
            result.AddRange(BitConverter.GetBytes(data.Yaw));
            result.AddRange(BitConverter.GetBytes(data.Pitch));
            result.AddRange(BitConverter.GetBytes(data.Roll));
            return result.ToArray();
        }
        #endregion
    }

}
```

C Server

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Net.NetworkInformation;
using System.Threading;
namespace WindowsFormsApplication1
{
    public class Server
    {
        int _port = 4014;
        DataFrames _df;
        UdpClient _udpc;
        int _cos = 1;
        int[] _tab = new int[16];
        int _time_limit=500;
        public Server()
        {
            //Creo UdpClient e gli do la porta per ascoltare aventuali connessioni
                in ingresso
            _udpc = new UdpClient(_port, AddressFamily.InterNetwork);
            IPEndPoint remoteIpEndPoint = null;
            _udpc.Receive(ref remoteIpEndPoint);
            _udpc.Connect(remoteIpEndPoint);
            _udpc.Send(new byte[10], 10);

        }


        public void SetFrame(float n, float x, float y, float z, float yaw, float
            pitch, float roll)
        {
                _df = new DataFrames(n, x, y, z, yaw, pitch, roll);
                //fisso valore massimo per la tabella che mi serve per gestire
                    traffico
                if (_df.N < 17) //valuto se Ã" dataframe di controllo
                {
                    if (_tab[((int)_df.N) - 1] < 2016000)
                        _tab[((int)_df.N) - 1] = _tab[((int)_df.N) - 1] + 1;
                    else
                        _tab[((int)_df.N) - 1] = 1;
```

```
                //settato _cos indica ogni quanti frame mandare (_cos compreso
                    tra 1 (connessione ottima) e 8 (connessione scadente) )
                if (_tab[((int)_df.N) - 1] % _cos == 0)
                    //setto il dataframe e inizio a inviare
                    _udpc.Client.Send(ToByte(_df));
            }
            else  //invio diretto per dataframe di controllo
                _udpc.Client.Send(ToByte(_df));
        }

        DateTime startTime;
        //metodo per la gestione del traffico
        public void Traffic()
        {
            System.Threading.Thread traf = new System.Threading.Thread(new
                ThreadStart(traff));
            traf.Start();
        }

        public void traff()
        {
            Object thisLock = new Object();
            lock (thisLock)
            {
                /* Read the initial time. */
                startTime = DateTime.Now;
                SetFrame(20, 0, 0, 0, 0, 0, 0);

                byte[] frames = new byte[28];
                //Attivo la ricezione dei dati
                _udpc.Client.Receive(frames);
                if (BitConverter.ToSingle(frames, 0) == 21)
                {
                    ///Condizioni per il controllo del traffico
                    DateTime stopTime = DateTime.Now;
                    TimeSpan duration = stopTime - startTime;
                    _time_limit = BitConverter.ToSingle(frames, 4);
                    if (duration.TotalMilliseconds > _time_limit)
                    {
                        if (_cos < 8)
                            _cos = _cos + 1;
                    }
                    else
                    {
                        if (_cos > 1)
                            _cos = _cos - 1;
                    }
                }
            }
        }

        #region Dataframe
        public struct DataFrames
        {
            public float N;
            public float X;
            public float Y;
            public float Z;
            public float Yaw;
            public float Pitch;
            public float Roll;

            public DataFrames(float n, float x, float y, float z, float yaw, float
                pitch, float roll)
            {
                N = n;
                X = x;
                Y = y;
                Z = z;
```

```
                Yaw = yaw;
                Pitch = pitch;
                Roll = roll;
            }
        }
        //Converte la struttura Data in un  array di bytes
        public byte[] ToByte(DataFrames data)
        {
            List<byte> result = new List<byte>();

            result.AddRange(BitConverter.GetBytes(data.N));
            result.AddRange(BitConverter.GetBytes(data.X));
            result.AddRange(BitConverter.GetBytes(data.Y));
            result.AddRange(BitConverter.GetBytes(data.Z));
            result.AddRange(BitConverter.GetBytes(data.Yaw));
            result.AddRange(BitConverter.GetBytes(data.Pitch));
            result.AddRange(BitConverter.GetBytes(data.Roll));
            return result.ToArray();
        }
        #endregion
    }
}
```

Bibliografia

[1] Nevio Benvenuto and Giovanni Cherubini. *Algorithms for communications system and their applications*. WILEY, March 2003.

[2] Jim kurose and keuth Ross. *Computer Networking: a top down approach featuring the Internet*. Addison-Wesley, terza edition, July 2004.

[3] K.R. Rao, Zoran S. Bojkovic, and Dragorad A. Milovanovic. *Multimedia communication system: techniques, standards and networks*. Prentice Hall PTR, 2002.

[4] Iain E.G. Richardson. *H.264 and MPEG-4 video compression*. Wiley, 2003.

[5] John R.Barry, Edward A.Lee, and David G.Messerschmitt. *Digital Communication*. Springer, terza edition, 2003.

[6] Rtp documentation avaible on rfc3550.

[7] Tcp documentation avaible on rfc793.

[8] Udp documentation avaible on rfc768.

[9] Icmp documentation avaible on rfc792.

[10] Documentation of directshow. Online avaible http://msdn.microsoft.com.

[11] Gsm codec. Online avaible http://www.3gpp.org.

[12] http://www.ieee.org.

[13] http://www.vocal.com/speech_coders/gsmfr.html.

[14] Hu Licai and Wang Shuozhong. *Information Hinding Based on GSM Full Rate Speech Coding*. IEEE.

[15] http://nem01.altervista.org/.

[16] http://www.on2.com/.

[17] On2's truemotion vp7 video codec, July 2008.

[18] Integrated homecare. Online avaible http://www.integratedhomecare.eu/.

[19] Linda S. Cline, John Du, Bernie Keany, K. Lakshman, Christian Maciocco, and David M. Putzolu. *DirectShow RTP Support for Adaptivity in Networked Multimedia Applications*. IEEE.

[20] Fan Zhang and Bo Li. *DirectShow Based Internet Video on Demand System*. IEEE.

[21] Kristo Lehtonen. *GSM Codec*. IEEE.

[22] Juan M, Huerta, and Richard M. Stern. *Speech Recognition from GSM codec parameters*. IEEE.

Ringraziamenti

Ci sono un sacco di persone che vorrei ringraziare, ma dubito che in questo momento riusciro a ricordare proprio tutti.

Per prima cosa vorrei ringraziare il professor Pupolin che mi ha dato la possibilità di scrivere la tesi con lui, ringrazio poi Denis che mi ha sopportato e aiutato in questo lungo periodo di tirocinio a khymeia e Andrea che "volontariamente" si è offerto come mio responsabile al San Camillo.

Un ringraziamento più che speciale lo faccio a papà Alfiero, mamma Fernanda, Katia, Loris, Scilla & il piccolo Andrea che mi hanno accompagnato nella crescita e mi sono sempre stati vicini.

Un grazie di cuore agli amici con cui sono cresciuto Mattia, Rosy, Stefy, Sara, Mony, Cicciuo, Consu;

Vorrei ringraziare poi:

Mike, Pera, Davide, Kikka e tutto l'agrigruppo per...perché ormai è ora di un'altra FESTAAA;

Dodo, Luca, Mirco & Jack per i tornei di Pes, accompagnati dal buon vinello, di cui si conosceva l'ora d'inizio ma non quella di fine;

Mattia & Massi per le "allegre" giornate trascorse in famiglia;

le compagne di merendine Lore & Sara che mi hanno tenuto compagnia nei lunghi pomeriggi padovani;

A tutti voi che avete contribuito a rendere l'esperienza di Sydney indimenticabile, specialmente a Ciccio e alla Pupassona

Alla pizzeria che mi ha tenuto occupato nei week-end con Fede, Gianni, Anna, Chiara, Selene, Fisnik, Christian, Matteo, Elena

A Br1 e al suo continuo scambio d'insulti via sms;

grazie anche a: Marco, Alberto, Daniele, Coe, Topo, Brutto Porko, Cera, Luca, Demba, Mariachiara, Giacomo, Magro, Giulia, Neno, Lena, Beppe, Nico, Erica, Chiara, Luca, Meme, Francy, Vale, Marta, Anna, Alice, Ale, Lety, Lino, Sonia, Serena, Ema, Alice, Elisa, Volpe, Marta, Laura, Marco, Magro, Simo, Bolzo, Matteo, Chiara, Ben, Cele, Alessia, Berto C, Laura, Skizz e tutti coloro che sono entrati a far parte della mia vita.

A tutta la compagnia del volley che si è succeduta in questi anni di partitelle al martedì sera in palestra Ale, Sabry, Moreno, Serena, Silvia, Chiara, Luana & Andrea, Roberto & Isabella, Davide, Lorenzo, Ale, Daniele.

All'e4a e al Trew per le giornate a passate a prendere il sole sopra i tetti delle case; all'ACR che mi ha tenuto occupato in questi anni; a MSDN per le licenze free donate a noi cari studenti; alle ferrovie che mi hanno lasciato viaggiare gratis per 25 anni, al si, alla corsa :(, alla ryanair per le mitiche offerte low-low-low-cost a 1 cent; alla 3 per i telefoni offerti; alla birra, gli spriss, i mezi mezi, i caffè alpini e tutto il resto, il ponte di Bassano, il parco dell'amicizia; al Topolino che non ha addormentato la mia sete di lettura; alla Ge e alla WikiPonch.

E come la ciliegina sulla torta, a tutto il CdSa, partendo dai fondatori Gabri, Ponch, Agno, Ste, Neckerz, Fiocco, a Fede & Marco per averci fatto scoprire le bellezze di Stoccolma, a Cioa, Bruno, Davide, Ritto e a tutti coloro che hanno avuto l'onore(o la sfiga :P) di averne a che fare, grazie veramente per aver reso questi anni così leggeri con il vostro sostegno e con tutti i bei momenti goliardici passati assieme.

A chi purtroppo ho dimenticato di nominare

A tutti voi un grandissimo GRAZIEEEEEE

Christian
(Grillo)