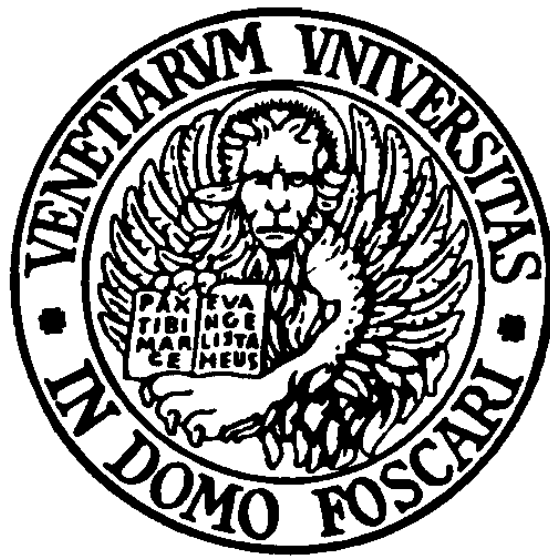


A.A. 2008/2009
Reti di Calcolatori

PROTOCOLLO

RTP/RTCP



Sebastiano Vascon
788442

Indice generale

Real-Time, cosa significa ?.....	3
RTP/RTCP.....	3
Introduzione.....	3
Sender.....	3
Receiver.....	4
Collocazione nella pila di livelli.....	5
Entra in gioco UDP.....	5
RTP.....	6
Esempio di utilizzo.....	6
Profili RTP.....	7
RTCP.....	7
Esempio di utilizzo.....	7
Pacchetti RTP.....	8
Pacchetti RTCP.....	10
RR - Receiver Report.....	11
Interpretazione dei dati di un RR.....	13
SR - Sender Report.....	14
Interpretazione dei dati di un SR – Report.....	14
SDES - Source Description.....	15
List of SDES Items.....	15
BYE - goodbye.....	17
APP - Application defined.....	17
Codice delle tipologie di pacchetti RTCP.....	18
Sessione RTP/RTCP.....	19
Banda di Sessione.....	20
Members Database.....	20
Temporizzazione dell'invio dei pacchetti RTCP.....	20
Reconsideration.....	21
Reverse Reconsideration.....	21
Fasi di una sessione RTP Multicast.....	22
Mixer.....	23
Translator.....	23
Jitter.....	24
Tecniche di gestione del jitter.....	24
Ritardo di riproduzione fissato.....	24
Ritardo di riproduzione adattivo.....	25
Recupero dei pacchetti persi.....	27
Forward Error Correction (F.E.C.).....	27
Interleaving (Interlacciamento).....	28
Ottimizzazioni.....	29
Header Compression.....	29
CRTP.....	29
ROHC.....	29
Algoritmi.....	30
SSRC_generator.....	30
collision_resolution.....	31
wrap_around_count.....	31
Contatti & Licenza.....	34

Real-Time, cosa significa ?

Definiremo Real-Time quell'insieme di entità tra le quali intercorrono relazioni la cui temporalità è un parametro imprescindibile.

Ad esempio la telefonia via internet, la videoconferenza, le lezioni a distanza, meeting e lavoro a distanza necessitano di trasferire informazioni quali video, audio, immagini ed altro ancora, che devono essere ricevuti in accordo con precise caratteristiche riguardo a tempo di trasmissione, variazione del tempo di trasmissione (jitter), percentuale di pacchetti persi e qualità (intesa come codifica) del dato ricevuta [1]

RTP/RTCP

Introduzione

RTP/RTCP è un protocollo¹ standard per la comunicazione in tempo reale emesso dalla IETF.

Nasce quasi spontaneamente col crescere delle applicazioni real time come radio, telefonia e videoconferenze. Ci si è accorti che ogni produttore, bene o male, reinventava il protocollo in modo molto simile ai concorrenti, da questo nacque la necessità di avere un protocollo generico e comune per la comunicazione...et voilà RTP !

E' stato creato, inizialmente, per poter permettere la partecipazione a conferenze da parte di più utenti ma, per come è stato strutturato, può essere utilizzato in altre applicazioni. Possiamo infatti trovare RTP in giochi dove la componente temporale è importante (ad esempio Quake) oppure utilizzare RTP per salvare un insieme continuo di dati (ad esempio un sensore che analizza la bontà di una certa fonte, connesso ad una rete, potrebbe inviare continuamente i dati raccolti).

Possiamo quindi individuare uno, o più, sender (produttore di dati) e uno, o più, receiver (consumatore di dati). Chiaramente un host può essere sia sender che receiver.

Sender

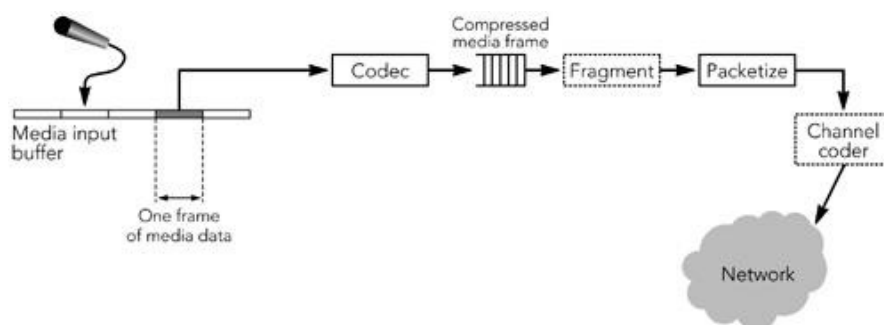


Illustrazione 1: Diagramma a blocchi di un processo sender in una tipica sessione RTP. Figura tratta da [2]

Come si può vedere nell'illustrazione 1 il sender è una fonte audio (in questo caso un microfono, ma potrebbe essere qualsiasi altra cosa), questa fonte genera un segnale che viene spezzettato in frame, codificata, compressa e inserita in pacchetti RTP che vengono inviati sulla rete.

¹ Insieme di convenzioni e regole stabilite tra due entità per comunicare

Receiver

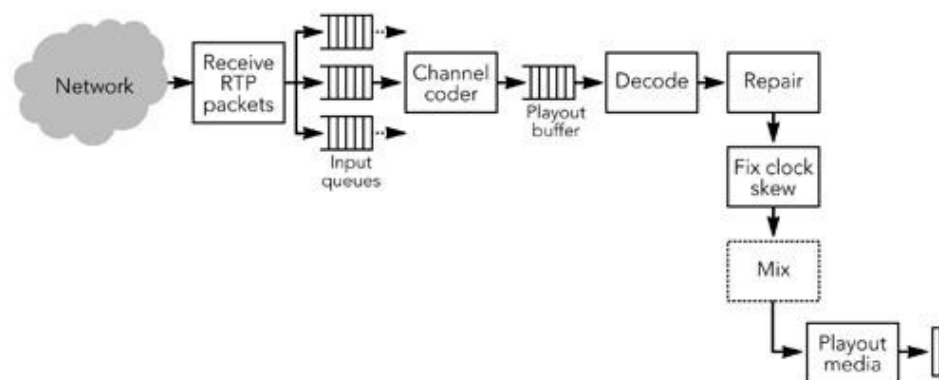


Illustrazione 2: Diagramma a blocchi di un processo receiver in una tipica sessione RTP. Figura tratta da [2]

Allo stesso modo il processo receiver riceve dalla rete i pacchetti RTP li decodifica, li unisce e li esegue in accordo alle specifiche temporali.

I dettagli implementativi si trovano nell'RFC3550 [3] che, dal 2003, sostituisce in toto l'RFC1889.

Le differenze tra le due RFC (3550 e 1889) le troveremo esclusivamente negli algoritmi che governano il protocollo mentre il formato dei pacchetti resta identico. La maggior differenza sta nell'introduzione di un sistema a “timer variabile” (trattato in questo documento) per inviare i pacchetti RTCP in modo da minimizzare l'utilizzo della banda che, vedremo, in caso di parecchi partecipanti potrebbe occupare percentuali importanti.

RTP si occupa della consegna mentre RTCP fornisce un supporto per monitorare la qualità del servizio.

Collocazione nella pila di livelli

RTP viene integrato (usando librerie esterne ad esempio JRTPLIB² o JMF³ se lavoriamo in ambiente Java) o implementato direttamente all'interno delle applicazioni scritte dall'utente. Pertanto la sua naturale collocazione è nello strato applicativo, ma a tutti gli effetti fornisce servizi di trasporto quindi sembrerebbe appartenere anche al livello di trasporto. Pertanto la definizione migliore è quella di un “*protocollo di trasporto implementato nello strato di applicazione*” [4]

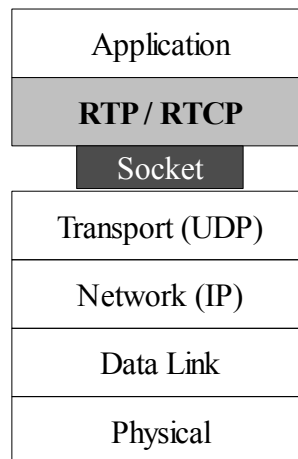


Illustrazione 3: Collocazione di RTP nella pila dei livelli

Entra in gioco UDP

RTP si appoggia principalmente sul protocollo UDP (*User Datagram Protocol*) proprio per le peculiarità del suo modello trasmissivo. In una comunicazione real-time la temporalità è vista come variabile critica nella trasmissione e l'unico protocollo a minimizzare questo parametro, a discapito di altri meno rilevanti in questo contesto, è appunto UDP. La ritrasmissione di un pacchetto perso (tipica di applicazioni che sfruttano TCP), il più delle volte, è inutile perchè il tempo di richiesta e arrivo dell'informazione è troppo, tanto da renderla inutilizzabile. Gli ACK non sono necessari per lo stesso motivo, è solo banda sprecata e computazioni addizionali inutili.

Svantaggi

- non garantisce l'arrivo del singolo pacchetto
- non esistono meccanismi di conferma (acknowledge)
- l'ordine di arrivo può non essere rispettato
- non garantisce la temporalità del pacchetto (arrivo entro un certo lasso di tempo)

Vantaggi

- Velocità

2 Implementata da Jori Liesemberg (jori@lumumba.luc.ac.be) per il suo lavoro di tesi "Voice over IP in networked virtual environments" e disponibile liberamente all'indirizzo <http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html>

3 <http://java.sun.com/javase/technologies/desktop/media/jmf/>

- Meno congestione della rete grazie all'assenza degli ACK (presenti in TCP)
- Possibilità di inviare in broadcast ⁴ e multicast ⁵ (assai importante per l'instaurazione, ad esempio, di conferenze)

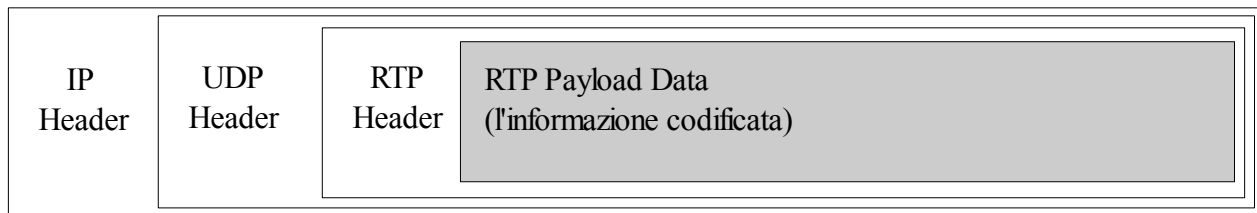


Illustrazione 4: Incapsulamento di un pacchetto RTP in datagrammi IP

RTP

RTP è l'acronimo per *Realtime Transport Protocol* (protocollo di trasporto in tempo reale) e assolve a tutti i compiti necessari per instaurare una comunicazione real time.

Abbiamo detto che RTP offre un servizio di trasporto in tempo reale appoggiandosi al protocollo UDP quindi dovrà sopperire a certe mancanze di UDP, in particolar modo:

- Fornire informazioni sul payload type (il formato dei dati inviati, ovvero la codifica effettuata sull'informazione annidata al pacchetto che vedremo potrà variare da pacchetto a pacchetto)
- Il sequence number (identificatore di sequenza per poter ristabilire l'ordine dei pacchetti ed eventualmente valutare quanti pacchetti sono andati perduti o scartati)
- Il timestamping (posizione temporale in millisecondi dei dati contenuti nel pacchetto, utile a sincronizzare ad esempio audio e video e vedremo abbassare/eliminare il jitter)
- Identificare le sorgenti per sincronizzare diversi flussi (ad esempio in una conferenza ci può essere la sovrapposizione della voce di due persone che parlano contemporaneamente)

RTP è il protocollo di trasporto e non fornisce nessun meccanismo per garantire la consegna del pacchetto, l'arrivo in tempo, la qualità del servizio e l'ordine di ricezione; il tutto viene demandato agli strati sottostanti.

Esempio di utilizzo

Consideriamo l'utilizzo di RTP per il trasporto vocale. Supponiamo che la voce sia codificata in PCM⁶ a 64 kbit/sec (normale comunicazione telefonica) e che i dati vengano raccolti ogni 20 ms.

$$1 \text{ sec} = 64 \text{ kbit} \Rightarrow 1 \text{ msec} = 64 \text{ bit} \Rightarrow 20 \text{ msec} = 1280 \text{ bit} = 160 \text{ byte}$$

Questo blocco di 20 ms (160 byte) viene incapsulato in un pacchetto RTP dove, tra le varie informazioni, verrà specificato il formato (PCM 64kbit/s) col quale sono stati campionati i dati, successivamente verrà passato al livello di trasporto per essere inviato. Il destinatario riceverà il pacchetto RTP tramite il livello di trasporto, vedrà il metodo di codifica dei dati (PCM), e passerà il

⁴ Broadcast: invio di un pacchetto a tutti gli host di un segmento di rete (quello che succede ad esempio con la radio)

⁵ Multicast: Invio di un pacchetto ad un gruppo di host

⁶ PCM: Un segnale analogico (ad esempio la voce) per essere digitalizzato viene campionato N volte al secondo e il valore assunto dal segnale ad ogni campionamento viene quantizzato in un intervallo rappresentabile con M bit $[0..2^M]$. Abbiamo quindi che ogni secondo necessita di $N \cdot M$ bit. Questa tecnica è detta Pulse Code Modulation (PCM) [5] pg 504-505

tutto all'applicazione che riprodurrà l'informazione.

Profili RTP

I profili RTP permettono di definire parametri comuni in una comunicazione semplicemente indicando il profilo di riferimento. I profili Audio/Video standard sono definiti nell'RFC3551 [6]. È possibile specificare un profilo proprietario semplicemente seguendo le istruzioni fornite nell'RFC3555.

RTCP

RTCP è l'acronimo di *Realtime Transport Control Protocol* ed è un protocollo di controllo che lavora parallelamente ad RTP e fornisce periodicamente informazioni sulla QoS e altre variabili della sessione, come ad esempio:

- Numero di pacchetti persi tra due host
- Jitter medio rilevato (vedremo in seguito le tecniche per il trattamento del jitter. pg 24)
- Informazioni sugli utenti attivi nella conferenza (canonical name CNAME, telefono, nome etc..)
- Desiderio di uno o più utenti di lasciare la sessione

RTP non necessita di RTCP per funzionare ma è consigliato utilizzare il protocollo di controllo per poter offrire un'esperienza migliore agli utilizzatori del nostro software e interfacciarsi con altri software che lo utilizzano.

Esempio di utilizzo

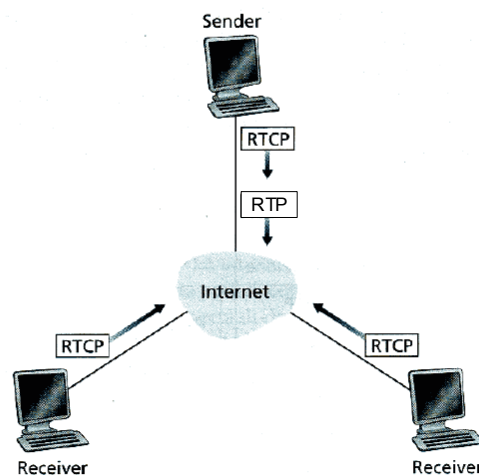


Illustrazione 5: Comunicazione RTCP

Ipotizziamo di essere in modalità multicast, tutti gli host della sessione si scambiano pacchetti RTP questi vengono riprodotti nei relativi client e parallelamente vengono inviati pacchetti RTCP relativi allo stato attuale della comunicazione.

In questo modo tutti sanno chi è ancora presente nella sessione, è possibile scambiarsi informazioni aggiuntive come il nome, il telefono, si può avvisare della volontà a disconnettersi e soprattutto si viene a conoscenza dell'effettiva QoS nella sessione.

Se la comunicazione non sta rispettando certi parametri impostati dall'applicazione, e lo si capisce dai pacchetti RTCP, è possibile attuare delle tecniche per migliorare la qualità (diminuire il bitrate della codifica, utilizzare un formato differente, attivare o meno dei mixer etc...).

Senza RTCP non potremo sapere cosa sta effettivamente succedendo, è lo strumento di feedback del sistema.

RTCP definisce quindi 5 tipologie di messaggi (*SS* p.14, *RR* p.11, *SDES* p.15, *BYE* p.17, *APP* p.17) con i relativi formati che vedremo nel dettaglio dopo aver trattato il formato generico dei pacchetti RTCP.

Pacchetti RTP

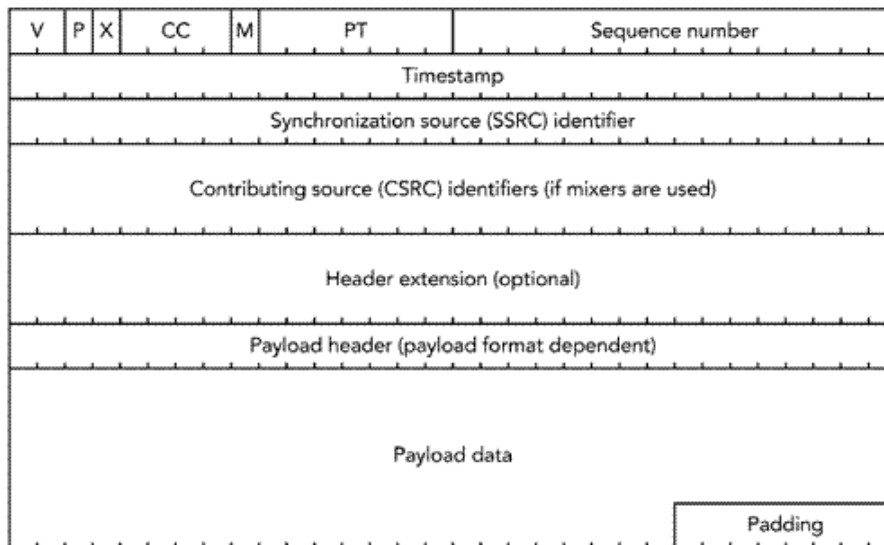


Illustrazione 6: Formato di un pacchetto RTP con campi opzionali e non [2]

V – Version Number

Indica la versione del protocollo, ad oggi siamo alla versione 2 quando si supererà la 3a versione sarà possibile specificarla nel campo opzionale “Header Extension”.

Campo a 2 bit

P – Padding

Indica se il pacchetto è stato riempito con bit di padding o meno. L'ultimo otteto del padding contiene il numero di otteti da ignorare alla fine dei dati compreso se stesso. L'operazione di padding non è obbligatoria per tutti i pacchetti RTP. Viene spesso utilizzata quando si richiede la crittazione dei dati dove, un formato a lunghezza “stabile”, è preferibile.

Campo a 1 bit

X – Extensions

Se impostato ad uno indica la presenza di una estensione dopo l'header statico. Il formato dell'estensione è descritto in [3] cap 5.3.1

Campo a 1 bit

CC – Count of contributing sources

Solitamente un pacchetto è inviato da una sola sorgente, nel caso in cui fossero presenti più sorgenti (ad esempio quando siamo in presenza di un mixer), CC indica il numero di sorgenti presenti all'interno del campo CSRC.

Campo a 4 bit

M – Marker

Abilitando il bit M possiamo specificare al receiver che i dati contenuti nel pacchetto sono, ad esempio, un keyframe di un video oppure l'inizio di una sequenza audio con sottotitoli da renderizzare o altre azioni scaturibili lato receiver dal nostro client. Spesso viene utilizzato per informare il receiver che il pacchetto è l'inizio di una talkspurt (vedi *talkspurt* pag. 25).

Campo a 1 bit

PT – Payload Type

Identifica il formato dei dati inclusi nel pacchetto permettendo all'applicazione un corretto playout.

Campo a 7 bit

Sequence Number

Ogni pacchetto inviato dal sender contiene un sequence number incrementale. Non possono esistere due pacchetti con lo stesso sequence number. Questo campo permette al receiver di ripristinare il corretto ordine dei dati e individuare il numero di pacchetti persi (o scartati per discordanze temporali). Il valore iniziale del sequence number viene impostato casualmente e successivamente incrementato di 1 ad ogni invio. Una conseguenza importante della lunghezza del campo a 16 bit è che quando si raggiunge il massimo (2^{16}) si torna a 0. Questo rappresenta un problema in quanto: se sono presenti forti ritardi nella comunicazione (come nel caso degli spikes) o semplicemente la comunicazione è lunga (se pensiamo ad un campionamento audio ogni 20 msec avremo conversazioni di al più $20 * 2^{16} = 20 * 65536 = 1310720 \text{ msec} = 1310,72 \text{ sec} = 21 \text{ min}$ circa), possono arrivare due pacchetti con lo stesso sequence number creando incongruenze in fase di playout. Si utilizza perciò, nell'applicazione, un contatore aggiuntivo detto `wrap_around_counter` [2] (algoritmo a pag. 31) che ci permette di mantenere traccia del numero di volte che si raggiunge il massimo sequence_number (o più semplicemente che si passa per 0). Ritroveremo il `wrap_around_counter` nel protocollo RTCP (vedi pag. 11).

Campo a 16 bit

Timestamp

Il timestamp viene registrato nell'istante di cattura del primo byte della fonte. Se ad esempio sappiamo che ogni pacchetto contiene 80 ms di campione, ogni pacchetto avrà un timestamp di distanza pari a 80ms. Questa informazione risulterà essenziale per un corretto playback del contenuto del pacchetto lato receiver e per il calcolo del jitter.

Campo a 32 bit

SSRC – Synchronization Source Identifier

Identificativo della sorgente che ha creato il contenuto del payload. Solitamente è un valore numerico generato a caso dalla sorgente quando inizializza la sessione ed è univoco all'interno della

sessione stessa. Nel caso di collisione, in quanto non è da escludere che due host non generino lo stesso SSRC, vi è un algoritmo risolutivo (vedi *collision_resolution* a pag. 31).

Durante la sessione ad ogni SSRC viene associato il CNAME relativo all'host.

Campo a 32 bit

CSRC – Contributing source

Contiene una lista delle fonti presenti nel payload. Il numero di fonti è specificato nel campo CC (vedi sopra) e può essere al massimo di 15 elementi. Questo campo viene utilizzato dai mixer per inserire gli SSRC di ogni fonte presente nel payload.

Lista (da 0 a 15) di campi da 32 bit. Max 60 byte.

Payload Header

In alcuni casi è necessario specificare informazioni aggiuntive utili all'applicazione per effettuare una migliore interpretazione dei dati del pacchetto, tali informazioni possono essere specificate all'interno del campo.

Campo a 32 bit

Payload Data

L'effettiva informazione codificata da inviare.

Pacchetti RTCP

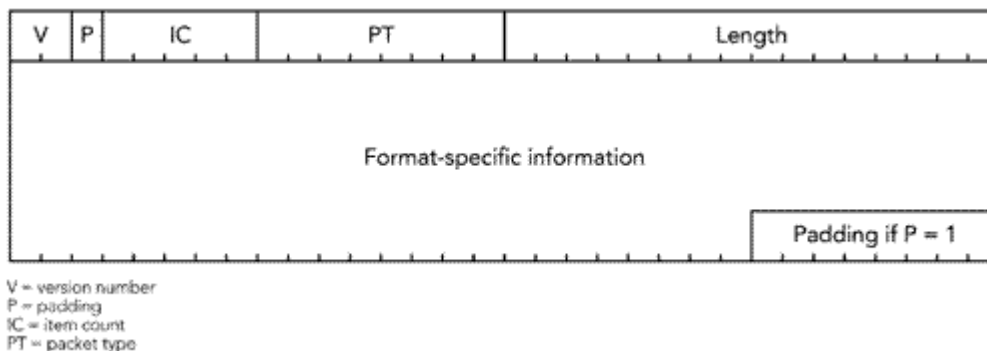


Illustrazione 7: Formato di un generico pacchetto RTCP [2]

I pacchetti RTCP sono formati da una parte iniziale (header) identica per tutti e il resto del pacchetto viene differenziato in base alle 5 tipologie previste.

V – Version Number

Versione del pacchetto RTCP, attualmente siamo alla 2a versione.

Campo a 2 bit

P – Padding

Stesso significato del bit nel pacchetto RTP (vedi sopra). Diventa obbligatorio nel caso in cui non si raggiunga una lunghezza pari ad un multiplo di 32.

Campo a 1 bit

IC – Item Count

Indica il numero di report presenti all'interno di un pacchetto RTCP, la lista va da 0 a 2^5 . Se diventa necessario spedire più di 32 report allora sarà necessario generare più pacchetti RTCP. Se IC = 0 non vuol dire che il pacchetto non porta con sé informazioni.

Campo a 5 bit

PT – Packet Type

Attualmente sono definiti 5 tipologie di pacchetti RTCP (vedi pag. 11), in futuro tale numero potrà espandersi con altre tipologie di report.

Campo a 8 bit

Length

Identifica la lunghezza complessiva del pacchetto RTCP. Il suo valore rappresenta un multiplo di 32 bit in quanto ogni pacchetto RTCP è composto da multipli di 32 bit (vedi padding).

Campo a 16 bit

Ora passiamo alle specifiche dei 5 tipi di report.

RR - Receiver Report

Uno degli utilizzi primari di RTCP è la ricezione dei report sulla qualità della trasmissione. Ogni receiver di una sessione invia pacchetti RR al resto dei partecipanti per informarli sullo stato della ricezione.

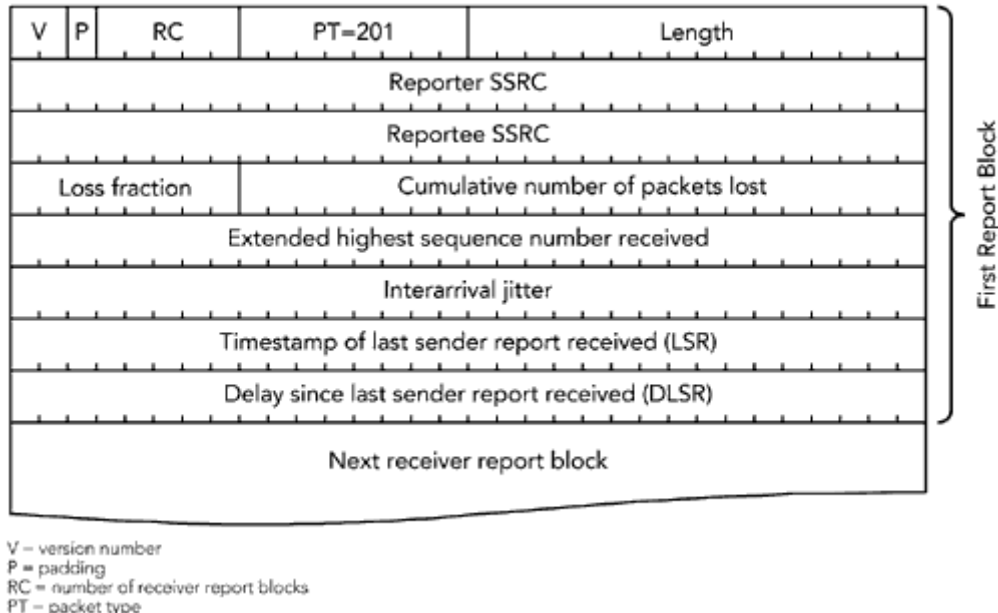


Illustrazione 8: Pacchetto RTCP – RR. Figura presa da [2]

Per i primi 32 bit vedi “Pacchetti RTCP” a pagina 10

Reporter SSRC

SSRC del mittente del report. *Campo a 32 bit*

Reportee SSRC

Soggetto SSRC del report. *Campo a 32 bit*

Loss fraction

Indica il rapporto dei pacchetti persi su quelli attesi.

$$lossFraction = \frac{\text{numero di pacchetti persi nell'intervallo}}{\text{numero di pacchetti attesi nell'intervallo}} * 256$$

numero di pacchetti attesi nell'intervallo = seq_num ultimo pacchetto – seq_num primo pacchetto

Campo a 8 bit

Cumulative number of packet lost

Numero di pacchetti persi in tutta la sessione (non solo nell'intervallo attuale).

Campo a 24 bit

Extended highest sequence number received

Indica l'ultimo sequence number ricevuto calcolando anche i wrap around.

$$\forall \text{pacchetto}_i \in \text{sessione}$$
$$extended_seq_num_i = sequence_number_i + (2^{16} * wrap_around_count)$$
$$ext_high_seq_num = \max\{extended_seq_num_i\}$$

Campo a 32 bit

Interarrival jitter

Rappresenta il jitter medio rilevato dal destinatario per ogni pacchetto e si calcola nel seguente modo:

R_i = Timestamp dell'i-esimo pacchetto rilevato dal receiver

S_i = Timestamp indicato dal mittente nel pacchetto RTP dell'i-esimo pacchetto

$$relative_transit_time = R_i - S_i$$

Ora definiano la differenza tra due pacchetti

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

...e il valore del campo Interarrival Jitter

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

La divisione per 16 viene spiegata nel dettaglio in [3] e [7].

Campo a 32 bit

LSR (Last Sender Report)

Se nessun ReporteeSSRC SR è stato ricevuto il campo viene impostato a 0 (zero) altrimenti viene preso l'ultimo sender report ricevuto (quello con NTP timestamp maggiore, vedi sotto).

L'LSR è composto da 32 bit mentre un NTP timestamp da 64 bit, bisogna perciò trovare una rappresentazione consona. NTP Timestamp è diviso in due blocchi da 32 bit ciascuno, il primo blocco (high) rappresenta la parte intera (i secondi) mentre il secondo (low) la parte

frazionaria. Vengono, perciò, presi i 32 bit centrali.

$$LSR_{\text{high 16 bit}} = NTP_{\text{low integer 16 bit}}, LSR_{\text{low 16 bit}} = NTP_{\text{high fractional 16 bit}}$$

$$LSR = LSR_{\text{high 16 bit}} \circ LSR_{\text{low 16 bit}}$$

◦ = concatenazione

Campo a 32 bit

DLSR (Delay since Last Sender Report)

Se nessun ReporteeSSRC SR è stato ricevuto allora il campo viene impostato a 0 (zero) altrimenti è calcolato con la seguente formula:

$$Now = \text{Timestamp di invio del RR report}$$

$$TSLSR = \text{Timestamp dell'ultimo SR ricevuto dal Reportee in oggetto}$$

$$DLSR = \frac{(Now - TSLSR)}{65536}$$

Interpretazione dei dati di un RR

Congestion Detection

L'utilizzo del campo Jitter può essere utile a rilevare stati di congestione. Statisticamente, si è osservato, che picchi di jitter preannunciano un elevato numero di pacchetti persi.

Round Trip Time (RTT)

Un sender che riceve RR da tutti i receiver può calcolare il Round Trip Time⁷ tra se e tutti gli altri host e valutare se bisogna attuare delle politiche per migliorare la trasmissione. Di seguito trovate un esempio tratto dall'RFC:

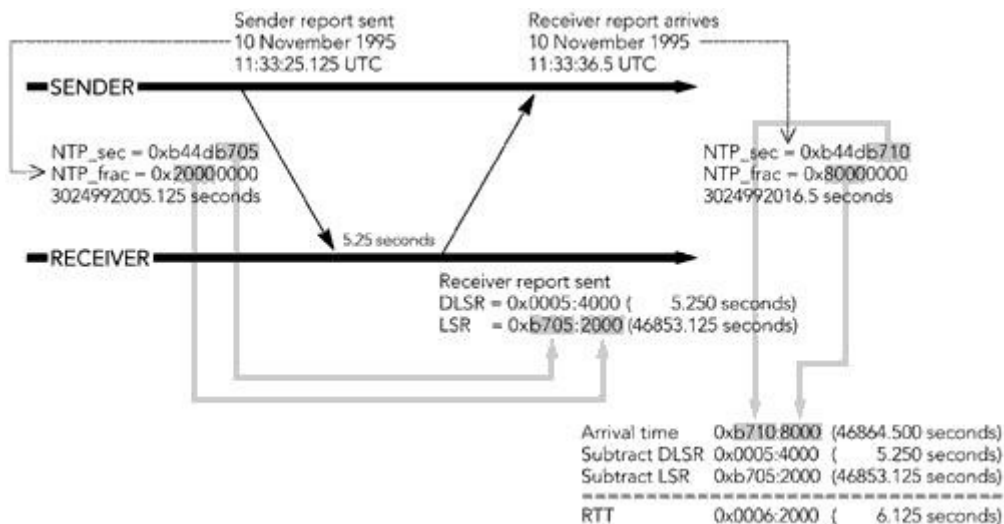


Illustrazione 9: Esempio di calcolo dell'RTT

L'esempio inoltre chiarisce ulteriormente il calcolo dell' LSR.

⁷ Tempo necessario ad un pacchetto per raggiungere la destinazione dalla sorgente. In questo caso viene calcolato il Network RTT cioè non sono presi in considerazione i tempi computazionali degli host. L'RTT è estremamente importante nelle applicazioni interattive come ad esempio i giochi. Alcuni studi [8] hanno dimostrato che un RTT >300 msec limita se non impedisce la conversazione tra due persone.

SR - Sender Report

I Sender Report vengono inviati dai sender poco dopo aver terminato l'invio di un pacchetto RTP e contengono le seguenti informazioni:

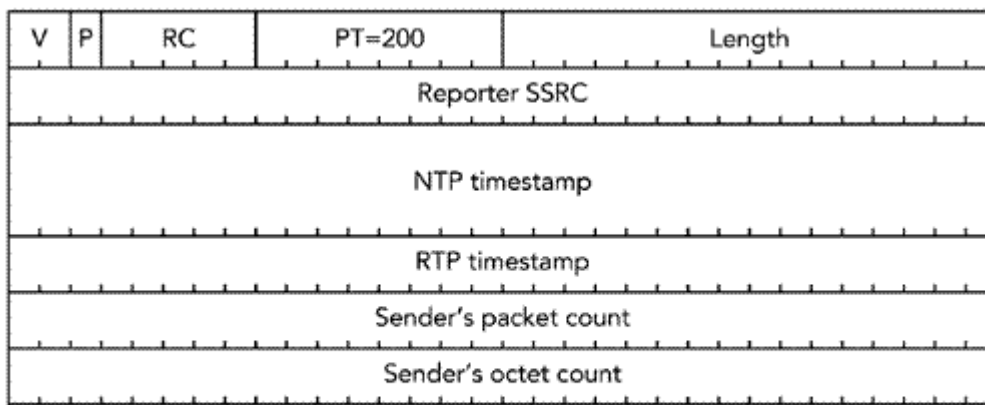


Illustrazione 10: Pacchetto RTCP - SR. Figura presa da [2]

Per i primi 32 bit vedi “Pacchetti RTCP” a pagina 10

Reporter SSRC

Vedi Reporter SSRC a pagina 11

NTP Timestamp

Timestamp in formato NTP dell'istante di invio del pacchetto RTP.

Campo a 64 bit

RTP Timestamp

Corrisponde allo stesso istante di lettura dell'NTP Timestamp ma relativo alla macchina locale.

Campo a 32 bit

Sender's Packet Count

Numero complessivo di pacchetti RTP inviati dall'inizio della sessione. Nel caso in cui nascano collisioni sul nome tale valore viene resettato.

Campo a 32 bit

Sender's Octet Count

Numero complessivo di byte di dati (sono escluse intestazioni etc...) inviati dall'inizio della sessione. Nel caso in cui nascano collisioni sul nome tale valore viene resettato.

Campo a 32 bit

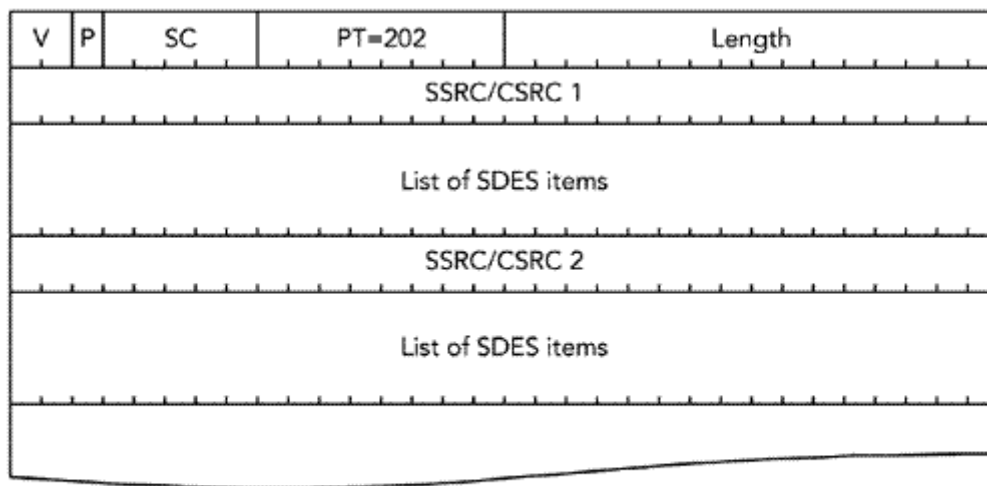
Interpretazione dei dati di un SR – Report

È possibile utilizzare il valore *Sender's Octet Count* per calcolare il peso medio di un pacchetto RTP.

Il valore di *NTP Timestamp* permette di risolvere il problema del *Lip-Synchronization* che si incontra quando si vuole sincronizzare ad esempio un video e la sua componente audio in modo da far combaciare il parlato video con il parlato audio.

SDES - Source Description

Vengono utilizzati i pacchetti SDES per inviare informazioni aggiuntive relative ai partecipanti della sessione. Ad esempio possono essere inviati il numero di telefono, l'indirizzo e-mail etc...



V = version number
P = padding
SC = number of SDES items
PT = packet type

Illustrazione 11: Pacchetto RTCP - SDES. Figura tratta da [2]

SSRC / CSRC

Specifica il riferimento dei dati presenti in List of SDES items.

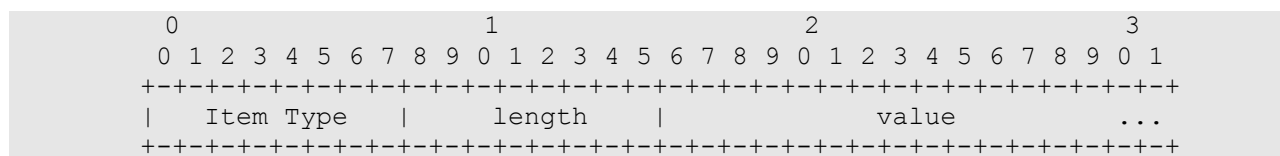
Campo a 32 bit

List of SDES Items

Il protocollo specifica 8 tipologie standard di elementi SDES.

Nessuno vieta la creazione di elementi aggiuntivi, basta seguire le linee guida indicate in [3] cap15.

Ogni elemento ha una struttura comune composta da:



Item Type è un campo a 8 bit che identifica la tipologia dell'elemento, *Length* è un campo a 8 bit e serve a definire la lunghezza in byte del campo value, *Value* è il valore vero e proprio dell'elemento e viene codificato in UTF-8 [9] la sua lunghezza massima viene limitata dal campo length a 256 byte che risultano più che sufficienti ad inviare le informazioni necessarie e a non sovraccaricare la comunicazione.

CNAME - Canonical End-Point Identifier SDES Item

È l'unico campo obbligatorio in ogni pacchetto RTCP-SDES e porta con se 3 importanti proprietà:

1. Crea l'associazione SSRC – Macchina. Dato che l'SSRC, in caso di conflitti, può variare nel tempo (vedi *collision_resolution*) il CNAME resta costante.
2. CNAME come SSRC sono unici all'interno di una sessione RTP.
3. Crea un collegamento tra i diversi flussi generati dallo stesso host (che avranno SSRC diversi in quanto ogni flusso è a se stante in una sessione RTP).

Il CNAME viene calcolato algebricamente dalla macchina e può essere nel seguente formato [user@host](#) dove per host si intende un “full qualified domain”.

Item Type = 1

NAME - User Name SDES Item

Permette ai partecipanti di scambiarsi il proprio nominativo, solitamente è un valore inserito dall'utente perciò non deve rispettare criteri di unicità.

Item Type = 2

EMAIL - Electronic Mail Address SDES Item

Permette di inviare il proprio indirizzo mail ([nome@dominio.ext](#)) ai partecipanti. Il formato dell'indirizzo è specificato in [10]. Il destinatario non fa assunzioni sulla validità dell'indirizzo pertanto, è opportuno, che vengano effettuati controlli lato sender prima di inviare il pacchetto.

Item Type = 3

PHONE - Phone Number SDES Item

Permette di inviare il proprio numero di telefono.

Item Type = 4

LOC - Geographic User Location SDES Item

Permette di inviare la propria localizzazione geografica (il formato viene definito dall'applicazione).

Item Type = 5

TOOL - Application or Tool Name SDES Item

Permette di inviare informazioni circa lo strumento utilizzato per generare lo stream di dati (ad esempio la versione del software oppure il modello di webcam...).

Item Type = 6

NOTE - Notice/Status SDES Item

Viene utilizzato per indicare lo stato dell'host ad esempio “al telefono” , “occupato”

Item Type = 7

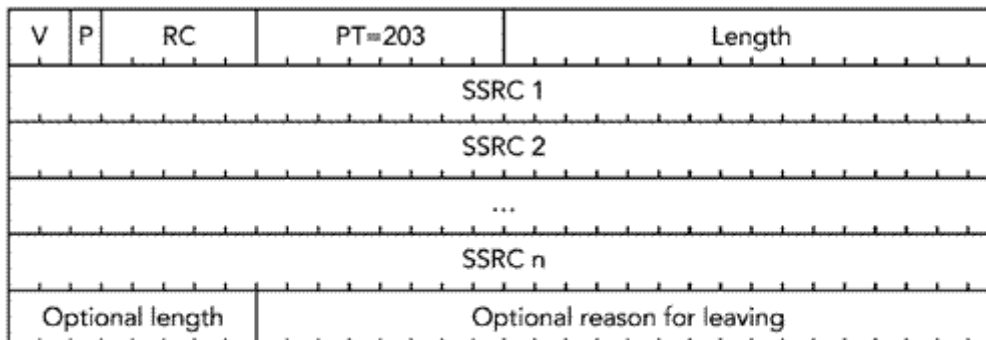
PRIV - Private Extensions SDES Item

Viene utilizzato per inviare estensioni particolari o sperimentali di un'applicazione.

Item Type = 8

BYE - goodbye

Indica la volontà di una o più sorgenti nel lasciare la sessione.



V = version number
P = padding
RC = number of SSRC headers
PT = packet type

Illustrazione 12: Pacchetto RTCP - BYE. Figura tratta da [2]

Inoltre è possibile indicare la motivazione della disconnessione nel campo “*optional reason for leaving*” la cui lunghezza è indicata in byte nel campo “*optional length*”.

SSRC-i

Vedi Reporter SSRC a pagina 11

APP - Application defined

Permette la creazione di pacchetti di estensione definiti dall'applicazione.

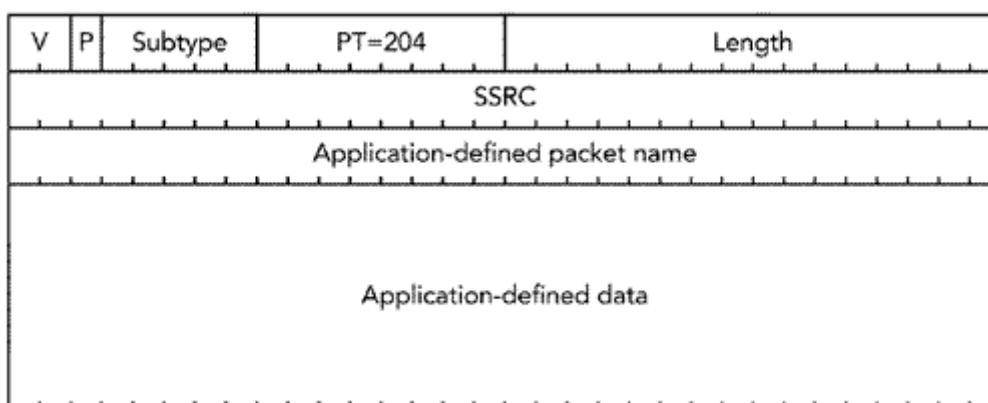


Illustrazione 13: Pacchetto RTCP - APP. Figura tratta da [2]

SSRC

Vedi Reporter SSRC a pagina 11

Application-defined packet name

Campo da 4 byte che definisce il nome del pacchetto.

Application-define data

Dati relativi al pacchetto.

Codice delle tipologie di pacchetti RTCP

<i>Tipologia Pacchetto</i>	<i>Codice Associato per il campo PT</i>
RR – Receiver Report	201
SR – Sender Report	200
SDES – Source Description	202
BYE – GoodBYE	203
APP – Application Defined	204

Banda di Sessione

Per ogni sessione si assume che il traffico sia assoggettato ad una ampiezza di banda comune detta “*session bandwidth*” e calcolata nel seguente modo:

m = numero di partecipanti attivi nella sessione
 $bandwidth_i$ = bandwidth dell' i -esimo partecipante

$$sessionBW = \sum_{i=1}^m bandwidth_i$$

Tutti i partecipanti devono utilizzare lo stesso session bandwidth in quanto su questo valore vengono calcolati i tempi di trasmissione per i pacchetti RTCP.

Il valore viene impostato dall'applicazione (solitamente uguale per tutti). È dipendente dai limiti fisici dell'infrastruttura mentre è indipendente dalla qualità della codifica scelta per i dati, mentre il valore della codifica dei dati dipende fortemente dal valore scelto per la banda di sessione (codifiche “sprecone” potrebbero occupare troppa banda e degradare le prestazioni generali della sessione).

Members Database

È importante, ai fini di una corretta gestione della sessione, essere a conoscenza di tutti i membri attivi e non. Per fare ciò ogni partecipante deve tenere traccia in una database interno che aggiornerà di volta in volta in base alla ricezione dei pacchetti RTCP.

Se un utente non invia pacchetti RTCP entro il limite massimo atteso (solitamente $5 \text{ sec} * 1.5$, in quanto 5 sec è il valore costante del timer di invio e 1.5 il limite superiore del metodo random vedi sotto) può essere eliminato dal proprio database e richiesta un'azione. Il valore è importante ai fini di un corretto conteggio dei tempi di invio dei pacchetti RTCP

Temporizzazione dell'invio dei pacchetti RTCP

All'aumentare del numero di host nella sessione il numero di pacchetti RTCP scambiati tra tutti i partecipanti cresce linearmente e inizia ad avere un peso sostanzioso sulla banda complessiva disponibile. Bisogna quindi prevedere un algoritmo che limiti il numero di pacchetti inviati e che, allo stesso tempo, permetta di mantenere l'utilità dei pacchetti RTCP.

L'RFC consiglia di utilizzare il 5% del valore della banda di sessione per i pacchetti RTCP, di questa porzione è raccomandato dedicarne il 25% ai sender e il restante¹¹, tali valori possono variare in base alla tipologia dei dati scambiati in una sessione e sono specificati nei profili RTP. L'algoritmo per il calcolo del tempo di intervallo di trasmissione è il seguente, definiamo:

$$RTCPpacketsize_i = RTCPsize + UDPsize + IPsize$$

$$avgrtcpsize = \frac{\sum_{i=1}^n RTCPpacketsize_i}{n}$$

$$rtcpBW = sessionBW * 0.05$$

$$Tmin = 5 \text{ sec}$$

1. Se il numero di sender è minore del 25% del numero totale di partecipanti e maggiore di 0 allora il valore si diversifica se ci troviamo nella situazione di host sender o receiver:

¹¹ Approfondimento in [3] cap 6.2

- **Host Sender**

$weSent = true$ (la stazione è sender)

$$C = \frac{AvgRtcpSize}{0.25 * rtcpBW}$$

n = numero di mittenti

- **Host Receiver**

$weSent = false$ (la stazione è receiver)

$$C = \frac{AvgRtcpSize}{0.75 * rtcpBW}$$

n = numero di receiver

Se il numero di sender è > del 25% del numero totale di partecipanti, allora sender e receiver sono trattati allo stesso modo:

$weSent = true$ (la stazione è sender) oppure $weSent = false$ (la stazione è receiver)

$$C = \frac{AvgRtcpSize}{rtcpBW}$$

2. Se i partecipanti non hanno ancora emesso nessun pacchetto RTCP allora la variabile T_{min} viene modificata a 2.5 sec altrimenti resta 5 sec.

3. Ora possiamo calcolare l'intervallo $T_d = \max\{T_{min}, C * n\}$

4. Non vogliamo che l'algoritmo sia deterministico in quanto ci potrebbero essere picchi di pacchetti RTCP quindi introduciamo un parametro casuale $T = \frac{T_d * rand(0.5, 1.5)}{e^{-\frac{3}{2}}}$ e

normalizziamo tutti dividendo per $e^{-\frac{3}{2}}$

La seguente procedura garantisce un intervallo casuale che in media distribuisce il 25% della banda RTCP ai sender e il restante 75% ai receiver. Se il numero di sender è maggiore del 25% dei partecipanti, questa procedura divide in modo equo la banda RTCP.

Non ci resta che calcolare quando inviare il prossimo pacchetto RTCP

$$NextRTCPTimeSend = CurrentTime + T$$

Di seguito troviamo due delle differenze sostanziali tra l'RFC1889 e quella attuale la RFC3550.

Reconsideration

Con questa tecnica si vogliono minimizzare le trasmissioni in eccesso quando molti partecipanti accedono alla sessione simultaneamente.

Reverse Reconsideration

Con questa tecnica si vuole ridurre la durata del timeout dei partecipanti quando vogliono lasciare la sessione.

Fasi di una sessione RTP Multicast

Possiamo evidenziare 4 fasi una sessione RTP

Fase 1: Join del Gruppo Multicast

In questa fase, descritta dall'immagine 15, viene negoziato l'accesso al gruppo multicast.

Tale procedura non interessa direttamente RTP in quanto risolta agli strati inferiori e da altri protocolli.

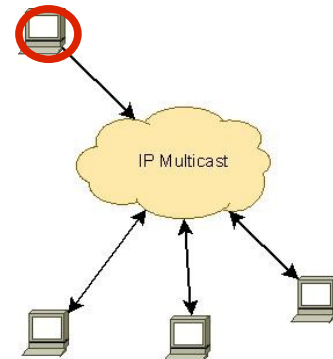


Illustrazione 15: Join di un host ad un gruppo multicast

Fase 2: Inizializzazione

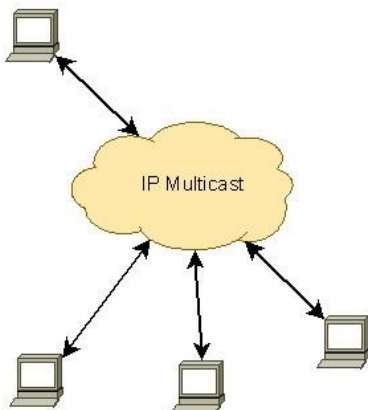


Illustrazione 16: Join avvenuta e inizio scambio dati con altri partecipanti

Una volta avuto accesso al gruppo (immagine 16) multicast vengono scatenate le procedure di inizializzazione dell'applicazione.

Viene generato l'SSRC evitando collisioni, viene inviato un SDES RTCP per informare gli altri della propria presenza e si iniziano a calcolare i tempi di invio dei pacchetti RTCP.

Il nuovo utente inizia a ricevere i pacchetti RTCP ed RTP degli altri host così può incrementare il suo database e calcolare i propri parametri RTCP e il timer di invio dei pacchetti.

Fase 3: Scambio di dati

Dopo la fase 2 si iniziano a ricevere pacchetti RTP e RTCP da tutti i partecipanti.

Fase 4: Terminazione

Una volta conclusa la propria partecipazione viene inviato un pacchetto RTCP-BYE per avvisare gli altri che ci si sta disconnettendo.

Mixer

Grazie all'introduzione dei pacchetti RTCP e alla flessibilità del protocollo RTP possiamo notevolmente aumentare la qualità del servizio:

- Modificando a “run-time” le codifiche dei singoli pacchetti in relazione alla situazione della rete (questo è possibile grazie al payload variabile dei pacchetti RTP).
- Posizionando opportunamente degli apparati detti mixer (convogliatori) nella nostra rete che uniscano più sorgenti.

Un mixer unisce flussi e ricodifica i dati.

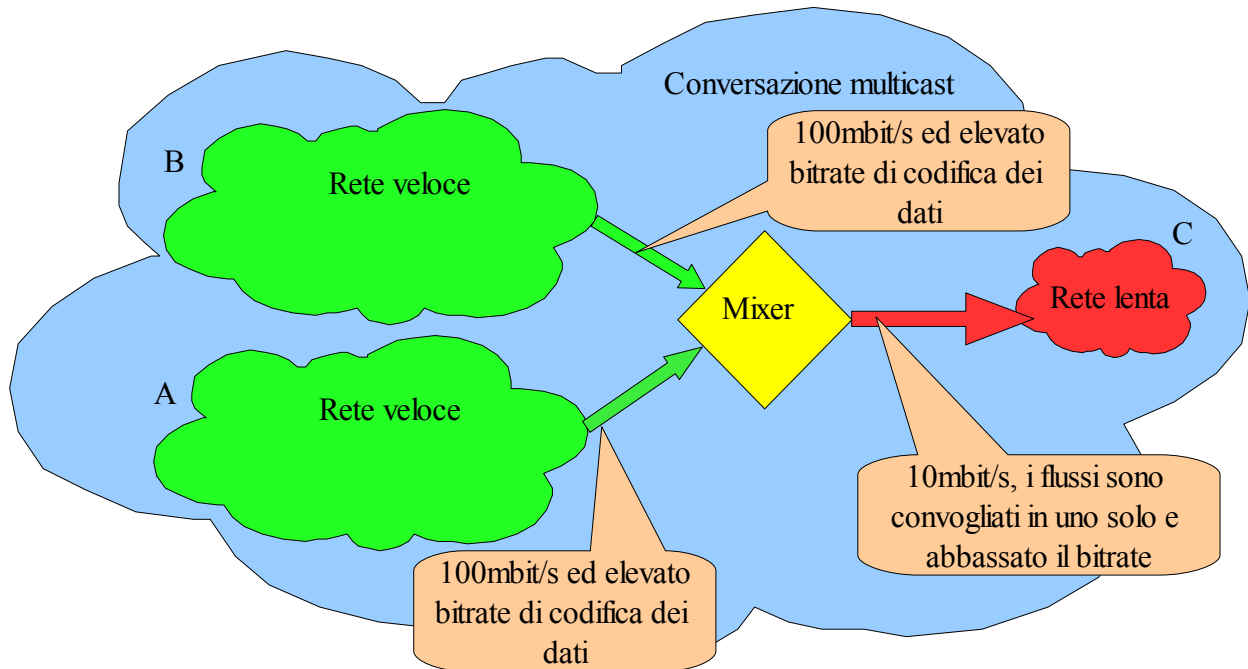


Illustrazione 17: Una tipica applicazione di un mixer in una rete con specificità diverse.

I flussi provenienti dalle reti "veloci" sono convogliati, attraverso il mixer, in uno solo ricodificando i dati. In questo modo gli utenti della rete “lenta” possono partecipare alla sessione (chiaramente a discapito della qualità ma in accordo con i vincoli fisici imposti dalla infrastruttura).

L'inserimento di un mixer aggiunge ulteriori problematiche al sistema, in quanto le sorgenti convogliate devono essere sincronizzate per generare un flusso coerente da inviare al destinatario.

Inoltre, per migliorare le prestazioni, dovrà impedire la trasmissione da una rete all'altra di report RTCP ritenuti indesiderati e dovrà generare i propri report SR e RR. Nell'esempio sopra non ha senso che un host della rete A invii report SR o RR a un'host della rete C, è tutto traffico sprecato mentre saranno fatti passare report SDES, APP e BYE.

Un mixer ha un proprio SSRC (vedi pag. 9) e, una volta convogliati i flussi in uno solo, provvederà ad inserirlo nel pacchetto RTP in uscita aggiungendo gli SSRC delle fonti al CSRC.

Translator

Il translator è sistema intermedio che agisce sui dati RTP mantenendo integre le informazioni del mittente e i relativi timestamp. Esempi di utilizzo possono essere:

- Ricodifica dei dati contenuti in un pacchetto RTP (simile al mixer, solo che non convoglia

più sorgenti)

- Bridging verso protocolli di trasporto differenti
- Rimozione o aggiunta di crittazione dei dati per passare da una connessione protetta a una non protetta.

Un translator è un'entità che non partecipa alla sessione (non aggiunge il proprio SSRC) e ogni flusso in ingresso scatena un flusso in uscita.

Jitter

L'argomento sconfinava al di fuori del protocollo RTP/RTCP in quanto vuol essere un compendio utile alla trattazione di uno dei problemi di maggior peso in ambito real-time. Per applicazioni come la trasmissione di flussi audio e video, non importa se i pacchetti sono trasmessi in 20 o 30 msec, l'essenziale è che il tempo di transito sia costante per poter garantire una riproduzione lineare della sorgente a destinazione. La variazione (ossia la deviazione standard) nel tempo di arrivo del pacchetto è chiamata jitter. Un jitter elevato, in cui per esempio alcuni pacchetti arrivano dopo 20 msec e altri dopo 30 msec, genererà un suono o un filmato di qualità variabile.¹² Grazie all'introduzione del sequence number, del timestamp e di alcune tecniche saremo in grado di abbassare o eliminare il fenomeno del jitter.

Tecniche di gestione del jitter

Usando la combinazione di *sequence number*, *timestamping* e l'introduzione di un ritardo di riproduzione nel receiver (*playout delay*¹³) possiamo eliminare il jitter.

Il *playout delay* aggiunto deve essere abbastanza elevato in modo da permettere la ricezione del maggior numero di pacchetti ma non troppo perchè ci troveremo ad avere una comunicazione (ad esempio nel caso di conversazione telefonica) dove dovremo aspettare che il receiver carichi tutti i pacchetti nell'intervallo. I pacchetti che non arrivano in tempo sono considerati persi, il reinvio potrebbe impegnare troppo tempo tanto da far perdere di significato una sua eventuale esecuzione.

Ritardo di riproduzione fissato

Il receiver tenta la riproduzione di un ciascun blocco q millisecondi dopo. In questo modo se un pacchetto ha timestamp t verrà riprodotto $q + t$. Ci rendiamo subito conto che la scelta di q è critica.

Una comunicazione telefonica può accettare un ritardo fino a 400 msec, solitamente si tenta di non arrivare a 400 msec ma a tenersi un po' più bassi.

Dato che il reinvio di dati non ha senso, esiste quindi un legame tra intervallo scelto e perdita dei pacchetti.

¹² L'argomento è approfondito in [4] Pagine 395-396

¹³ L'argomento è approfondito in [5] Pagina 518

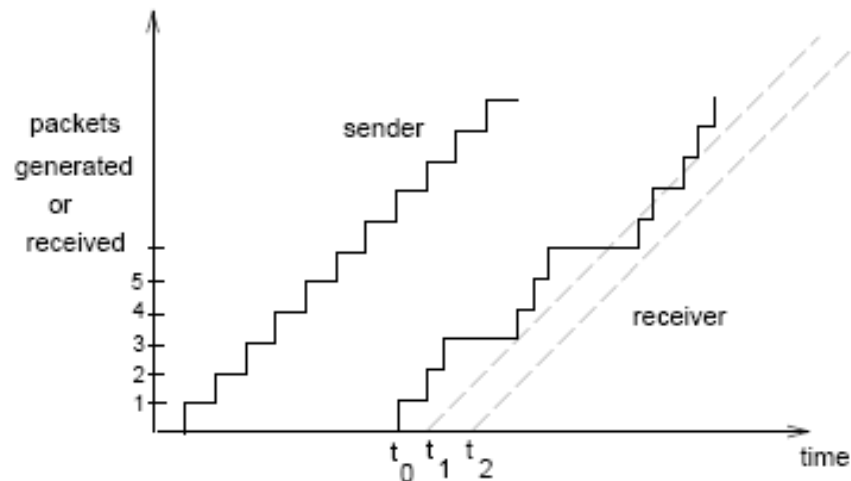


Illustrazione 18: Si può notare come nel caso di t_2 i pacchetti non riprodotti siano pari a 0. Figura tratta da [11]

Nell'esempio dell'illustrazione 18 abbiamo 3 ritardi di esecuzione. t_0 rappresenta l'esecuzione al tempo di arrivo ($t_0=0$) e $t_0 < t_1 < t_2$ ritardi applicabili. Utilizzando t_1 ci sono dei pacchetti che devono ancora arrivare quindi non saranno riprodotti, utilizzando t_2 si vede come tutti i pacchetti vengano correttamente eseguiti ma con un ritardo complessivo maggiore.

Ritardo di riproduzione adattivo

Abbiamo visto che la tecnica del ritardo fisso crea una forte relazione tra pacchetti persi e ritardo di esecuzione. Nel ritardo adattivo si cerca di stimare il ritardo di propagazione sulla rete all'inizio di ogni sessione e applicarlo ai pacchetti in fase di riproduzione.

Viene di seguito indicato l'algoritmo generico per il calcolo del ritardo proposto da Ramjee nel 1994 [11]:

t_i = tempo di generazione dell' i -esimo pacchetto , valore del timestamp

r_i = timestamp dell' i -esimo pacchetto ricevuto dal receiver

p_i = timestamp della riproduzione nel receiver

$j_i = r_i - t_i$ = jitter riscontrato dall' i -esimo pacchetto

d_i = Ritardo medio calcolato per l' i -esimo pacchetto

v_i = Variazione del ritardo calcolato in relazione al ritardo medio

Il valore di j_i non è detto sia corretto in quanto non si assume una sincronizzazione tra sender e receiver questo comunque è ininfluente in quanto tutti i pacchetti saranno assoggettati a questa differenza che sarà sempre costante. I 4 algoritmi proposti da Ramjee si differiscono esclusivamente per il calcolo del parametro d_i mentre v_i dipende da d_i .

L'algoritmo generico discrimina due casi:

1. Il pacchetto ricevuto è il primo di una *talkspurt*¹⁴

$$p_i = t_i + d_i + 4 * v_i$$

2. I restanti pacchetti vengono trattati nel seguente modo:

¹⁴ Per ottimizzare la comunicazione, solitamente, in una conversazione vengono separati i momenti di silenzio da quelli "attivi" detti talkspurt.

Supponiamo p_i il primo pacchetto della talkspurt in esame e p_j un'altro pacchetto generico della talkspurt allora $p_j = p_i + t_j - t_i$.

In pratica viene calcolato il nuovo punto di esecuzione come offset tra il primo pacchetto e il tempo di arrivo.

Si noti che il calcolo viene effettuato indiscriminatamente per ogni pacchetto, di seguito sono accennati i 4 algoritmi.

Primo algoritmo:

Dobbiamo definire quanto la storia dei ritardi influenza il calcolo del ritardo da applicare al nostro pacchetto

$$d_{i-1} = \text{valore medio dei precedenti ritardi}$$
$$d_i = \alpha * d_{i-1} + (1 - \alpha) * j_i$$

inoltre ci interessa calcolare il valore relativo al discostamento dal ritardo medio appena calcolato

$$v_{i-1} = \text{deviazione media del ritardo precedente}$$
$$v_i = \alpha * v_{i-1} + (1 - \alpha) * |j_i - d_i|$$

In base agli studi condotti da Ramjee un buon valore per il vincolo α è impostarlo pari 0,998002.

Secondo Algoritmo:

È una variazione del primo algoritmo, il calcolo di d_i viene effettuato in base al valore di j_i .

Se $j_i > d_i$ allora $d_i = \beta * d_{i-1} + (1 - \beta) * j_i \wedge \beta = 0.75$ altrimenti viene utilizzato il calcolo di d_i del primo algoritmo. La formula per calcolare v_i resta invariata.

Terzo Algoritmo:

Sia $S_i = \{\text{insieme dei pacchetti} \in \text{talkspurt}\}$ e sia i primo pacchetto della talkspurt allora $d_i = \min_{j \in S_i} \{j_j\}$ cioè il più piccolo jitter osservato nell'insieme dei pacchetti ricevuti.

Quarto Algoritmo:

Ramjee, nel corso dei suoi esperimenti, ha osservato che in una trasmissione possono presentarsi picchi di ritardo (vedi illustrazione 19) e che il tempo di risposta dei primi 3 algoritmi non è abbastanza veloce ad adattarsi alla situazione. Da questa problematica nasce il quarto algoritmo.

Una volta impostata la soglia (*spikesThreshold*) individuare un picco è relativamente semplice:

$$\text{if } (|j_i - j_{i-1}|) > \text{spikesThreshold then mode} = \text{IMPULSE};$$

Come si nota dall'illustrazione 19 un picco, solitamente, non è relativo ad un solo pacchetto quindi risulta naturale seguire l'andamento della trasmissione stando in "allerta". Maggiori dettagli implementativi sono reperibili in [11]

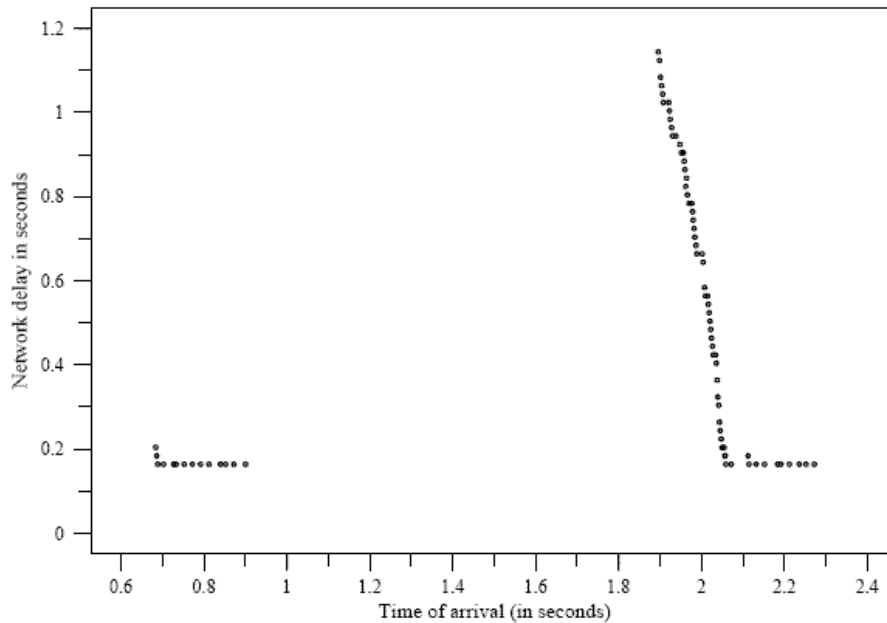


Illustrazione 19: Esempio di spikes [11]

Fin'ora abbiamo visto algoritmi basati sull'ipotesi che il primo pacchetto arrivi sempre e che si riesca sempre a riconoscere. E se ciò non dovesse accadere perchè ad esempio si è perso proprio il primo pacchetto? Si ricorre al sequence number per determinare il primo pacchetto.

Recupero dei pacchetti persi

Oltre alle tecniche viste per trattare il jitter esistono tecniche per poter recuperare i pacchetti persi in caso di ritardi in particolare esistono due metodi.

Forward Error Correction (F.E.C.)

L'idea alla base della “*correzione dell'errore in avanti*” è l'aggiunta di informazione ridondante ai pacchetti. Esistono due metodologie di FEC:

1. Viene inviato un blocco di codifica ridondante dopo n blocchi. Il blocco viene ottenuto tramite una XOR degli n blocchi originali.
2. Viene inviato, come informazione ridondante, uno stream della sorgente originale codificata in bassa risoluzione. Con questo sistema si riesce facilmente a ricostruire la sequenza originale.

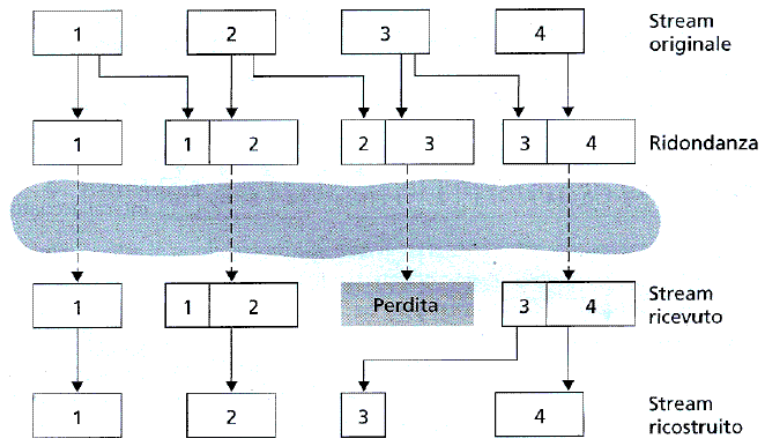


Illustrazione 20: Esempio di F.E.T. dove ad eccezione del primo per ovvi motivi, viene accodato il precedente con codifica più bassa. Figura tratta da [5]

Chiaramente aggiungendo più blocchi ridondanti in ogni pacchetto (ad esempio l'n-1 e l'n-2 esimo pacchetto o anche di più) si potranno recuperare più errori.

Freephone e RAT sono due applicazioni di telefonia internet che utilizzano la FET.

Interleaving (Interlacciamento)

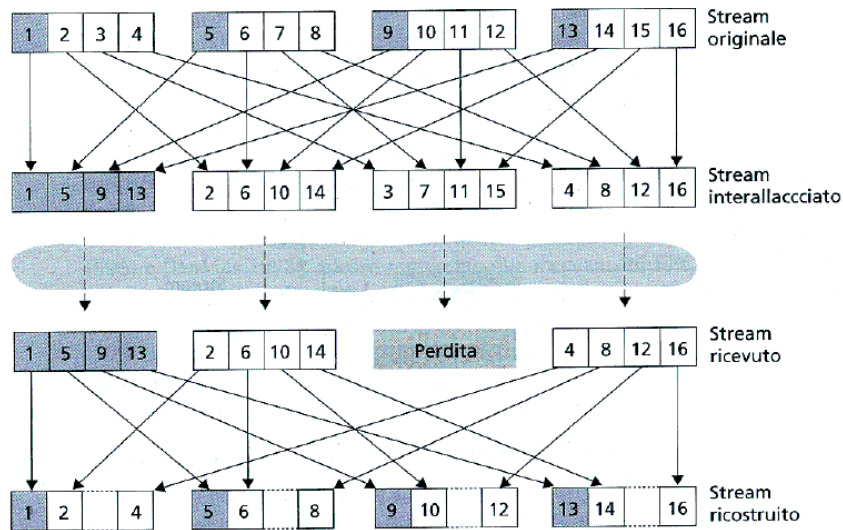


Illustrazione 21: Esempio di interleaving sui pacchetti. Figura tratta da [5]

Una singolo pacchetto viene diviso in n sotto pacchetti e intrecciato con i successivi n, in figura 21 abbiamo un esempio.

Grande vantaggio è l'assenza totale di ridondanza e di aumento nell'occupazione della banda, grosso svantaggio è il ritardo che il processo comporta (devo avere in memoria n pacchetti per provvedere alle divisioni) e ciò limita l'utilizzo di questa tecnica alle sole applicazioni di streaming e risulta difficilmente applicabile a processi real-time. Se pensiamo a dei pacchetti audio campionati ogni 20 ms, nel caso di perdita di un pacchetto interlacciato avremo dei silenzi di 5ms ogni 15ms e l'orecchio umano, in una conversazione, riesce a comprendere ugualmente il senso della frase.

Ottimizzazioni

Header Compression

Una delle parti più fortemente criticate nel protocollo RTP è lo spreco di spazio utilizzato nelle intestazioni (IP/UDP/RTP). Per questo motivo sono stati introdotti due standard CRTP (Compressed RTP) e ROHC (Robust Header Compression) per ottimizzare le intestazioni.

CRTP

Se il checksum sul protocollo UDP è disabilitato riesce a comprimere i 40 bytes dell'intestazione complessiva in 2 byte in caso contrario verrà compresso a 4 byte. I dettagli implementativi li trovate nell'RFC2508 [12]

ROHC

Si è notato che CRTP non lavorava bene in ambienti con forte ritardo o con un numero considerevole di pacchetti persi (ad esempio in una connessione Voip WiFi), da questo nasce ROHC.

Tale tecnologia è ormai parte integrante della nuova generazione di sistemi telefonici che implementano il protocollo RTP.

Le specifiche tecniche sono disponibili nell'RFC3095 [13]

Algoritmi

SSRC_generator

Nell'RFC3550 viene specificata una modalità comoda per la generazione del proprio SSRC basata su MD5 [14] e alcuni parametri del sistema. Di seguito trovate il codice in C dell'RFC3550:

```
/*
 * Generate a random 32-bit quantity.
 */
#include <sys/types.h> /* u_long */
#include <sys/time.h> /* gettimeofday() */
#include <unistd.h> /* get..() */
#include <stdio.h> /* printf() */
#include <time.h> /* clock() */
#include <sys/utsname.h> /* uname() */
#include "global.h" /* from RFC 1321 */
#include "md5.h" /* from RFC 1321 */

#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final

static u_long md_32(char *string, int length)
{
    MD_CTX context;
    union {
        char c[16];
        u_long x[4];
    } digest;
    u_long r;
    int i;

    MDInit (&context);

    MDUpdate (&context, string, length);
    MDFinal ((unsigned char *)&digest, &context);
    r = 0;
    for (i = 0; i < 3; i++) {
        r ^= digest.x[i];
    }
    return r;
} /* md_32 */

/*
 * Return random unsigned 32-bit quantity. Use 'type' argument if
 * you need to generate several different values in close succession.
 */
u_int32 random32(int type)
{
    struct {
        int type;
        struct timeval tv;
        clock_t cpu;
        pid_t pid;
        u_long hid;
        uid_t uid;
    } s;
```

```

        gid_t    gid;
        struct  utsname name;
    } s;

    gettimeofday(&s.tv, 0);
    uname(&s.name);
    s.type = type;
    s.cpu  = clock();
    s.pid  = getpid();
    s.hid  = gethostid();
    s.uid  = getuid();
    s.gid  = getgid();
    /* also: system uptime */

    return md_32((char *)&s, sizeof(s));
} /* random32 */

```

collision_resolution

Nel caso di SSRC identici all'interno della sessione si procede con questa modalità:

1. L'host che rileva la collisione invia un pacchetto RTCP – BYE
2. Genera un nuovo SSRC

Gli host possono abbassare la probabilità di collisioni semplicemente mantenendo una tabella dei partecipanti e inviare il primo pacchetto col proprio SSRC dopo aver ricevuto gli SSRC degli altri host.

wrap_around_count

max_misorder = 100 and max_dropout = 3000 → Consigliato dalla RFC

```

uint16_t udelta = seq - max_seq
if (udelta < max_dropout) {
    if (seq < max_seq) {
        wrap_around_count++
    }
    max_seq = seq;
} else if (udelta <= 65535 - max_misorder) {
    // The sequence number made a very large jump
    if (seq == bad_seq) {
        // Two sequential packets received; assume the
        // other side has restarted without telling us
        ...
    } else {
        bad_seq = seq + 1;
    }
} else {
    // Duplicate or misordered packet
    ...
}

```

Indice delle illustrazioni

Illustrazione 1: Diagramma a blocchi di un processo sender in una tipica sessione RTP. Figura tratta da [2]	3
Illustrazione 2: Diagramma a blocchi di un processo receiver in una tipica sessione RTP. Figura tratta da [2].....	4
Illustrazione 3: Collocazione di RTP nella pila dei livelli.....	5
Illustrazione 4: Incapsulamento di un pacchetto RTP in datagrammi IP.....	6
Illustrazione 5: Comunicazione RTCP.....	7
Illustrazione 6: Formato di un pacchetto RTP con campi opzionali e non [2].....	8
Illustrazione 7: Formato di un generico pacchetto RTCP [2].....	10
Illustrazione 8: Pacchetto RTCP – RR. Figura presa da [2].....	11
Illustrazione 9: Esempio di calcolo dell'RTT.....	13
Illustrazione 10: Pacchetto RTCP - SR. Figura presa da [2].....	14
Illustrazione 11: Pacchetto RTCP - SDES. Figura tratta da [2].....	15
Illustrazione 12: Pacchetto RTCP - BYE. Figura tratta da [2].....	17
Illustrazione 13: Pacchetto RTCP - APP. Figura tratta da [2].....	17
Illustrazione 14: Esempio di uno schema di sessioni RTP multicast (Audio e Video).....	19
Illustrazione 15: Join di un host ad un gruppo multicast.....	22
Illustrazione 16: Join avvenuta e inizio scambio dati con altri partecipanti.....	22
Illustrazione 17: Una tipica applicazione di un mixer in una rete con specificità diverse.	23
Illustrazione 18: Si può notare come nel caso di t2 i pacchetti non riprodotti siano pari a 0. Figura tratta da [11].....	25
Illustrazione 19: Esempio di spikes [11]	27
Illustrazione 20: Esempio di F.E.T. dove ad ogni pacchetto, ad eccezione del primo per ovvi motivi, viene accodato il precedente con codifica più bassa. Figura tratta da [5].....	28
Illustrazione 21: Esempio di interleaving sui pacchetti. Figura tratta da [5].....	28

Bibliografia

- [1] : Trasmissione Video su reti a commutazione di pacchetto, Antonio Caputo , 2001
Università degli studi di Napoli Federico II
- [2] : RTP: Audio and Video for the Internet , Colin Perkins , 2003, Addison Wesley - 1a
- [3] : RFC3550 - IETF, RTP: A Transport Protocol for Real-Time Applications, Internet Engineering Task Force , Luglio 2003 <http://tools.ietf.org/html/rfc3550>
- [4] : Reti di calcolatori , A.Tanenbaum , 2003, Pearson - 4a
- [5] : Internet e Reti di Calcolatori , J. Kurose , K. Ross , 2003, McGraw Hill - 2a
- [6] : RFC3551 - IETF, RTP Profile for Audio and Video Conferences with Minimal Control, Internet Engineering Task Force , Luglio 2003 <http://tools.ietf.org/html/rfc3551>
- [7] : Foundations of Digital Signal Processing and Data Analysis , Cadzow J. , 1987, Macmillan - a
- [8] : G.114 - , One-way Transmission Time, ITU-T Recommendation , May 2000
- [9] : RFC2279 - F. Yergeau, UTF-8, a Transformation Format of ISO 10646, Internet Engineering Task Force , Gennaio 1998
- [10] : RFC2822 - P. Resnick, Internet Message Format, QUALCOMM Incorporated , Aprile 2001
- [11] : Ran Ramjee, Jim Kurose, Don Towsley, Henning Schulzrinne, Adaptive playout mechanisms for packetized audio applications in wide-area networks, 1994-
<http://research.microsoft.com/~ramjee/>
- [12] : RFC2508 - S. Casner - V. Jacobson, Compressing IP/UDP/RTP Headers for Low-Speed

Serial Links, Internet Engineering Task Force , Febbraio 1999

[13] : RFC3095 - C. Bormann, Robust Header Compression (ROHC): Framework and Four Profiles: RTP, UDP, ESP and Uncompressed, Internet Engineering Task Force , Luglio 2001

[14] : RFC1321 - R. Rivest, The MD5 Message-Digest Algorithm, MIT Laboratory for Computer Science and RSA Data Security Inc. , Aprile 1992 <http://tools.ietf.org/html/rfc1321>

Contatti & Licenza

Sebastiano Vascon

Cell: +39 340 62 63 885

Mail: me@xwasco.com

Url: <http://sebastiano.vascon.it>

Avviso Importante:

Il contenuto di questo testo è da ritenersi “as is” e non mi assumo la responsabilità per eventuali errori sia di forma che di contenuto. Se doveste trovare imprecisioni o errori siete pregati di informarmi e provvederò a effettuare le dovute correzioni.

Creative Common License:

Sei libero di: riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire, recitare e modificare quest'opera.

Alle seguenti condizioni:

Devi attribuire questo lavoro a Sebastiano Vascon (indicando questo link <http://sebastiano.vascon.it>).

Non puoi usare quest'opera per fini commerciali. Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.

Ulteriori dettagli, che vi consiglio di leggere, circa la licenza Creative Common li trovate a questo link: <http://creativecommons.org/licenses/by-nc-sa/2.5/it/>

Grazie !

SEBASTIANO