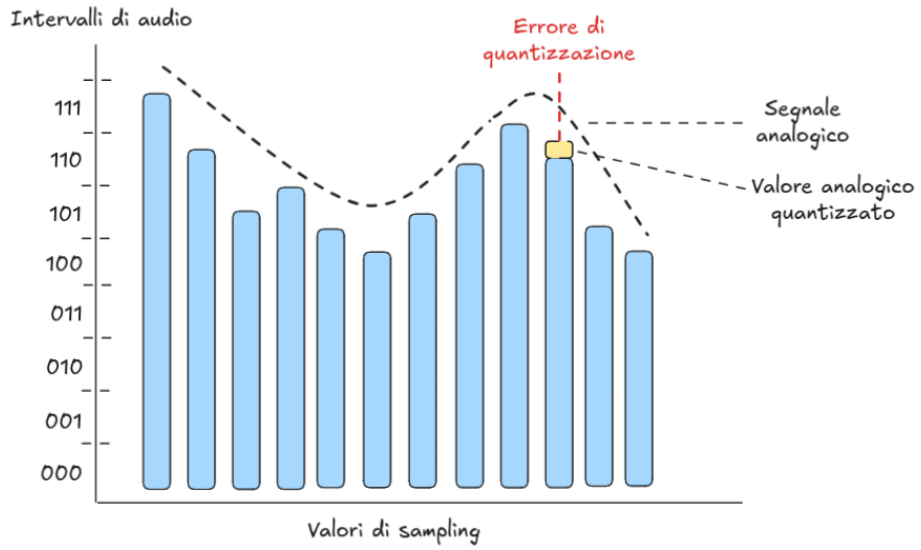
# Reti multimediali

Il concetto fondamentale è il **bit rate** ovvero la capacità di banda occupata durante una trasmissione video o audio. Più la qualità dello stream è elevata più è alto il **bit rate** e viceversa.




## Multimedia: Audio

L'audio viene elaborato attraverso un **processo di sampling**, in cui vengono prelevati dei valori a intervalli regolari nel tempo. Ogni campione viene successivamente **quantizzato**, ovvero trasformato in un **valore discreto binario** che ne permette la digitalizzazione. Tuttavia, questa conversione introduce un **errore di quantizzazione**, poiché i valori analogici vengono approssimati a quelli discreti disponibili.

## Multimedia: Video

I video sono una **sequenza di immagini composte da pixel**, ognuno dei quali possiede una determinata **profondità** di colore espressa in bit. Per **ridurre la quantità** di dati trasmessi o memorizzati, si utilizza la compressione, che sfrutta i **key frame** come punti di riferimento e salva solo le variazioni tra un fotogramma e il successivo, **evitando di ripetere** informazioni identiche. Ci sono due modalità di codifica **CBR (Constant Bit Rate)**, che mantiene un bitrate costante e **VBR (Variable Bit Rate)**, che adatta dinamicamente il bitrate in base alla complessità dell'immagine.

# Tipologie di applicazioni

1. **Streaming di contenuti memorizzati** (audio, video):
   - **Streaming**: avvio della riproduzione prima del download completo.
   - **Stored (memorizzato sul server)**: trasmissione più veloce della riproduzione, richiede buffering lato client.
2. **Conversazione vocale/video su IP**:
   - Comunicazione interattiva tra utenti con bassa tolleranza al ritardo come Skype.
3. **Streaming live** (audio/video):
   - Trasmissione di eventi in diretta.

# Streaming di contenuti memorizzati

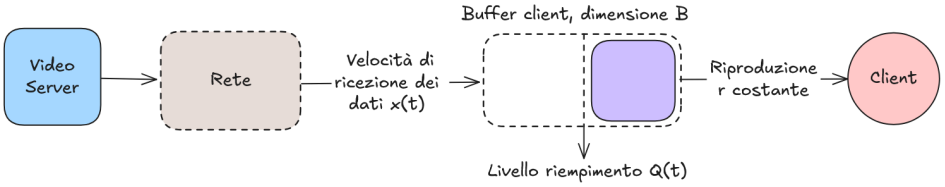
Il video viene memorizzato dal server come un insieme di chunk. Si vorrebbe un chunk al secondo per il client ma la realtà è che ci possono essere delay. Abbiamo diverse sfide:

- **Riproduzione continuo**: Una volta iniziata la riproduzione, non devono esserci interruzioni tra i chunk. Tuttavia, il **ritardo di rete è variabile (jitter)**, quindi il client deve **bufferizzare** per evitare interruzioni.
- **Interattività del client**: Il client deve poter eseguire azioni come **pausa, avanti, indietro, avanti veloce**.
- **Perdita e ritrasmissione**: I pacchetti possono essere persi e devono essere **ritrasmessi**.

> Il **jitter** è un indicatore della **variazione media nei tempi di consegna dei chunk** al client.

Quindi il client non può aspettarsi un chunk al momento giusto, ovvero quando termina l'altro serve un **buffer** e un **ritardo di riproduzione iniziale**.

1. Il client attende che il **buffer sia abbastanza pieno** prima di avviare la riproduzione.
2. **La riproduzione inizia a tempo** $t_p$.
3. Il **livello del buffer varia** in base alla velocità di ricezione dei dati variabile **x(t)**, mentre la velocità di **riproduzione r** è **costante**.



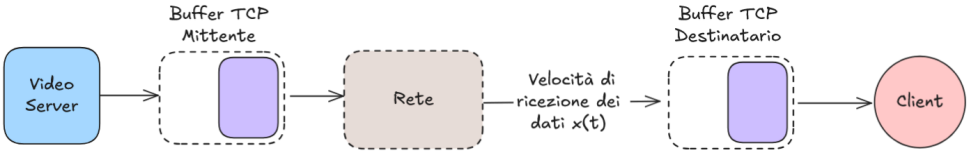

- Se **x medio è inferiore a r**, il buffer si svuota, causando **freeze del video** fino a quando il buffer non viene riempito di nuovo.
- Se **x medio è maggiore a r**, il buffer non si svuota, a condizione che il **ritardo di riproduzione iniziale** sia sufficiente per assorbire le variazioni di rete.

## UDP

Il **server invia i dati a una velocità appropriata per il client**. Viene introdotto un **piccolo playout delay** per **ridurre il jitter di rete**. La **correzione degli errori** (**error recovery**) è gestita **a livello applicativo**, se il tempo lo consente. Spesso i **firewall bloccano UDP**, rendendone difficile l'uso. Si può usare **TCP per la comunicazione bidirezionale**, ma **non scala bene** (più overhead e latenza).

## HTTP

Si scarica i video tramite **GET Request** con rate massimo TCP, la velocità di ricezione e quindi di riempimento dei buffer varia per il **congestion control** e **ritrasmissioni** di TCP. Un ritardo di riproduzione iniziale più grande rende la connessione migliore.



Per i due socket TCP c'è un **buffer per i chunks**. Quando quello del client è pieno impone un **limite** alla **velocità** di ricezione dei dati **indirettamente**, trasmette fin tanto che il chunk ricevuto non viene scartato, significa che non ha più spazio nel buffer.

Per saltare nei video da un punto a un altro, usiamo **l'header HTTP** "**byte-range**" per specificare i byte da ricevere. Se lo ripete nuovamente si invia un altra richiesta e il server si **scorda della precedente**.

Con TCP si **risolve il problema del jittering** ovvero la variazione di rete e i **rallentamenti non** sono un **problema** se i buffer sono grandi.

# DASH (Dynamic Adaptive Streaming over HTTP)

- **Lato server:** divide il file in chunk, ogni chunk è salvato e codificato in qualità diverse e ha un manifest file con gli URL dei diversi chunk.
- **Lato client**: misura periodicamente la qualità della trasmissione e tramite manifest sceglie i chunk con una qualità adatta alla trasmissione attuale. La logica risiede nel client che decide i chunk e determina: **quando** richiederlo, **quale** qualità e da **dove**.

# VoIP (Voice over IP)

Per funzionare server un **basso delay** per garantire una conversazione fluida. La sessione VoIP **inizia** con la **pubblicizzazione dell'IP, della porta e dell'algoritmo di encoding**. Abbiamo:

- **Generazione dei pacchetti audio**: La voce alterna **momenti di parlato e di silenzio, i** pacchetti vengono generati **solo quando si parla**. Si creano **chunk da 20ms** di audio.
- **Struttura dei pacchetti**: Ogni chunk audio viene **arricchito con un header del livello applicativo**. Il pacchetto viene poi **incapsulato in UDP o TCP**.
- **Invio dei segmenti**: L'applicazione **manda un segmento ogni 20ms** durante il parlato.

## VoIP: perdita di pacchetti e delay

1. **Network Loss**: Pacchetti persi a causa della **congestione della rete**.
2. **Delay Loss**: I pacchetti **arrivano troppo tardi** per essere riprodotti e vengono scartati.
3. **Loss Tolerance**: La voce può tollerare **una perdita tra l'1% e il 10% dei pacchetti.**

## Jitter e ritardo variabile

Il **jitter** è la misura della **variazione del tempo di arrivo dei pacchetti**. È dato dalla **differenza tra i tempi di trasmissione** dei pacchetti consecutivi. Un jitter elevato può causare **interruzioni e distorsioni audio** se non viene compensato.

## Riproduzione a delay fisso

Il ricevitore tenta di riprodurre ogni chunk **esattamente dopo q ms** dal momento in cui è stato generato. Un chunk ha un **timestamp t**, verrà riprodotto a **t+q**, se arriva dopo questo calcolo viene scartato. Dobbiamo fare un tradeoff con q dato che più è grande meno pacchetti si perdono ma più è piccolo più l'esperienza è fluida per entrambi gli utenti.

## Riproduzione a delay adattivo

Vogliamo ridurre il **ritardo di riproduzione** (cioè quanto tempo si aspetta prima di riprodurre un pacchetto audio) e la **perdita percepita** di pacchetti (che a volte non sono persi, ma solo arrivati troppo tardi). Il delay stimato è:

$$d_i = (1 - \alpha) \cdot d_{i-1} + \alpha \cdot (r_i - t_i)$$

Si può anche stimare la variazione media del delay ovvero il jitter.

$$v_i = (1 - \beta) \cdot v_{i-1} + \beta \cdot \mid r_i - t_i - d_i \mid$$

Quindi il ricevitore decide quando trasmettere il pacchetto:

$$playout\ time = t_i + d_i + k \cdot v_i$$

Ma come si capisce se un pacchetto è il primo del talk sprut?

1. **Senza perdite**: Se il ricevitore vede che **tra due pacchetti ricevuti consecutivi** c'è un **gap > 20 ms** (ad esempio: il primo ha timestamp 100 ms e il successivo 200 ms), **vuol dire che c'è stato silenzio,** quindi il secondo è **l'inizio di un nuovo talk spurt**.
2. **Con perdite:** Se ci sono **buchi nei numeri di sequenza**, si confrontano: il **timestamp del pacchetto ricevuto** e il **sequence number** per vedere se ci sono "buchi". Il **gap temporale è grande** (es. > 20 ms), **e non ci sono pacchetti ricevuti nel mezzo** allora si **assume che sia l'inizio di un nuovo talk spurt**.

## VoiP recupero di pacchetti persi

Metodo classico è che il ricevitore invia un **NACK** per richiedere la ritrasmissione di pacchetti persi, ma non è attuabile dato che abbiamo troppo ritardo nella comunicazione. Si utilizza **FEC (Forward Error Correction)** è un metodo in cui il trasmettitore invia **bit di ridondanza** insieme ai dati originali, in modo che il ricevitore possa **ricostruire i pacchetti persi senza richiedere una ritrasmissione**.

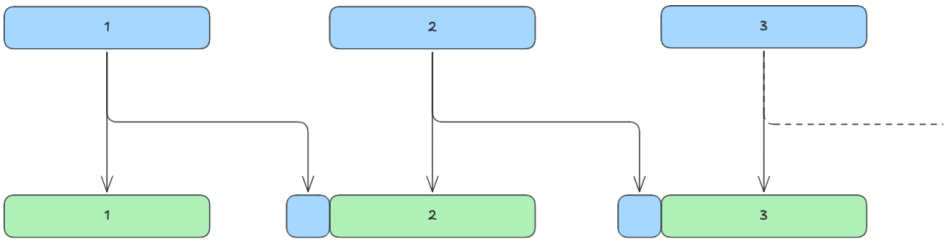
## FEC semplice

L'idea base è **aggiungere ridondanza** per permettere il recupero dei pacchetti persi.

- **Per ogni gruppo di N chunk** si crea un **chunk ridondante** basato su XOR dei chunk originali.
- Si inviano N+1 chunk, **aumentando la banda del** $\frac{1}{N}$.
- Se si **perde al massimo un chunk**, si può **ricostruire il dato mancante**.
- Questo metodo **funziona bene con piccoli playout delay**, ma ha overhead di banda.

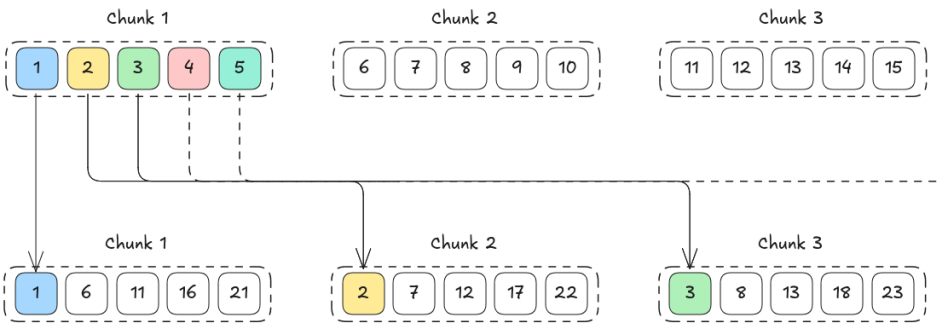
## FEC schema alternativo

Un'altra tecnica consiste nel **trasmettere un secondo flusso audio ridondante** a qualità inferiore, se il pacchetto principale si perde, si usa il **pacchetto di backup** con qualità minore.




## Interleaving per coprire le perdite

Gli **audio chunk vengono suddivisi in unità più piccole,** ogni pacchetto trasporta **dati di più chunk**. Se un pacchetto si perde, si possono **ricostruire parti dell'audio originale** dai pacchetti successivi.




Nel nostro esempio se il Chunk 2 fosse perso, noi possiamo comunque ricostruirlo, non avremo il pezzo numero 7 ma non è un grosso problema.

# VoiP Skype

Skype utilizza un'architettura **ibrida P2P** con i seguenti elementi principali:

- **Client (SC - Skype Client)**: Sono gli utenti normali che usano Skype per comunicare.
- **Supernodi (SN)**: Sono nodi speciali che aiutano a instradare il traffico, gestire NAT e trovare altri utenti.
- **Overlay Network**: La rete che connette i SN e permette di localizzare i client e i login server.
- **Login Server**: Gestisce l'autenticazione e mantiene traccia degli utenti online.

## Operazioni di un Client Skype

1. **Connessione alla rete Skype:** Il client si connette a un **Supernodo (SN)** tramite **TCP**. L'IP del SN viene memorizzato in cache per connessioni future.
2. **Login e autenticazione:** L'utente inserisce username e password. Il login avviene tramite il **Login Server**, che verifica l'identità dell'utente.
3. **Risoluzione dell'IP del destinatario:** Il client ottiene l'IP della persona da chiamare. Se l'IP non è noto, lo ottiene tramite il **SN** o la **Overlay Network**. Se il client ha già chiamato il destinatario, può usare l'IP memorizzato nella cache.
4. **Avvio della chiamata:** Se possibile, la chiamata avviene **direttamente** tra i due client.

## Peer come Relay

Quando due utenti **A e B** usano Skype e **sono entrambi dietro NAT**, non possono stabilire una connessione diretta perché:

- Un peer esterno **non può avviare una connessione** verso un peer dietro NAT.
- Solo il peer interno (dietro NAT) può **iniziare** una connessione verso l'esterno.

Quindi sia **A che B** mantengono una **connessione attiva** con i propri **Supernodi (SN), q**uando A vuole connettersi a B, A dice al suo SN di voler contattare B. Il **SN di A si collega al SN di B**. Il SN di B **usa la connessione già aperta** con B per inoltrare i dati.

# RTP (Real-Time Protocol)

RTP **non trasmette da solo**, ma **definisce la struttura dei pacchetti** che contengono i dati audio/video, aggiungendo informazioni utili per la sincronizzazione e la ricostruzione del flusso multimediale:

- **Payload type identification** → Indica il tipo di contenuto
- **Sequence number** → Aiuta a rilevare pacchetti persi e riordinare quelli ricevuti.

- **Timestamp** → Indica quando un pacchetto deve essere riprodotto, utile per il playout.

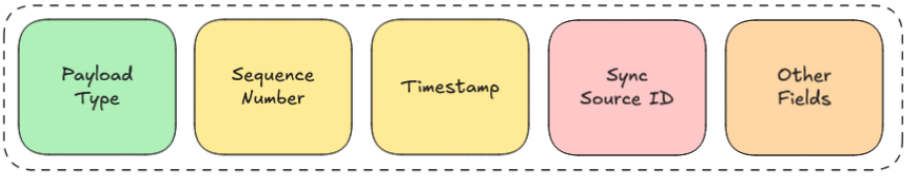
RTP viene **eseguito sugli end system,** i pacchetti sono **incapsulati in UDP**, non in TCP, per evitare i ritardi da ritrasmissione. **Due applicazioni VoIP che usano RTP possono comunicare tra loro**.

Le librerie RTP forniscono una **Transport Layer Interface** che estende UDP e consente di accedere a: **indirizzo IP** e **numero di porta**, **tipo di payload**, **numero di sequenza** e **timestamp**.

## RTP e QoS (Quality of Service)

RTP **non garantisce alcuna qualità di servizio** (QoS), i **router non gestiscono direttamente RTP**, vedono solo pacchetti UDP normali e forniscono solo un servizio **best-effort**.

## RTP Header




1. **Payload Type (7 bit):** Indica il tipo di **encoding audio o video**. Il ricevitore usa questo campo per sapere **come decodificare** il contenuto.
2. **Sequence Number (16 bit):** Viene **incrementato di 1** per ogni pacchetto RTP inviato. Serve per rilevare **perdite di pacchetti** e **ordinare** correttamente i pacchetti ricevuti.
3. **Timestamp Field (32 bit):** Rappresenta **l'istante di campionamento del primo byte** del pacchetto RTP.
4. **SSRC (Synchronization Source - 32 bit):** Identifica **univocamente la sorgente** del flusso RTP all'interno della sessione.

## Real-Time Control Protocol (RTCP)

RTCP funziona **insieme a RTP** e viene usato da ogni partecipante per inviare **periodicamente pacchetti di controllo** agli altri. **Report del sender e/o del receiver**, e serve per creare le statistiche utili nell'applicazione usata come: numero di pacchetti **inviati/ricevuti**, pacchetti **persi** o **jitter**.

## RTCP: Multicast e più mittenti

Questa parte si riferisce a scenari in cui **ci sono più partecipanti**, ad esempio in **videoconferenze o streaming live**. Una sessione RTP/RTCP **usa tipicamente un solo indirizzo multicast e** tutti i pacchetti RTP e RTCP della stessa sessione vengono **inviati allo stesso indirizzo multicast**.

- RTP e RTCP si distinguono: **tramite il numero di porta.**
- Per evitare che il traffico RTCP cresca troppo: r**idurre la frequenza dei pacchetti RTCP**.