# Chapter 9
Multimedia
Networking

A note on the use of these Powerpoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR
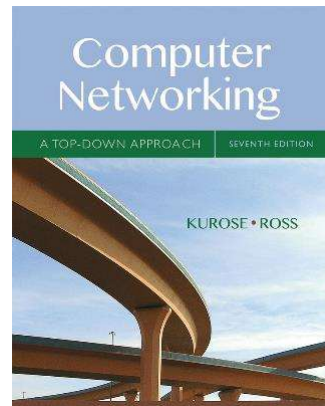
Computer Networking: A Top Down Approach

7th edition
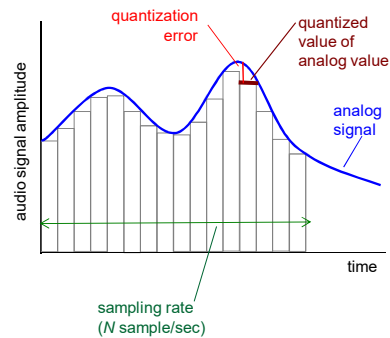Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

1

Multimedia: audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., $2^8=256$ possible quantized values
  - each quantized value represented by bits, e.g., 8 bits for 256 values
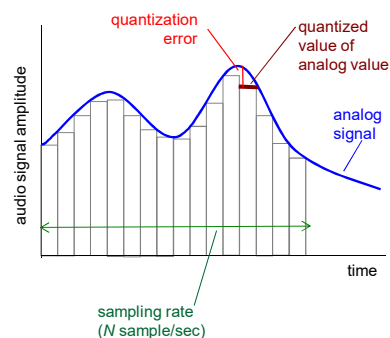

quantization error

quantized value of analog value

analog signal

audio signal amplitude

time

sampling rate (*N* sample/sec)

Multimedia: audio

- example: 8,000 samples/sec, 256 quantized values: 64,000 bps
- receiver converts bits back to analog signal:
  - some quality reduction

example rates
- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 kbps and up

quantization error

quantized value of analog value

analog signal

audio signal amplitude

time

sampling rate (*N* sample/sec)

Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple) and number of repeated values (*N)
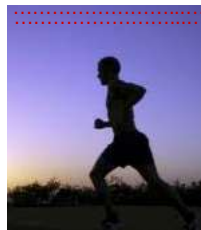
frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i
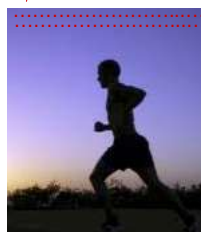
frame *i+1*

Multimedia Networking 9-5

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple) and number of repeated values (*N)

frame *i*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Multimedia Networking 9-6

Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - *streaming:* can begin playout before downloading entire file
  - *stored (at server):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- *conversational* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype
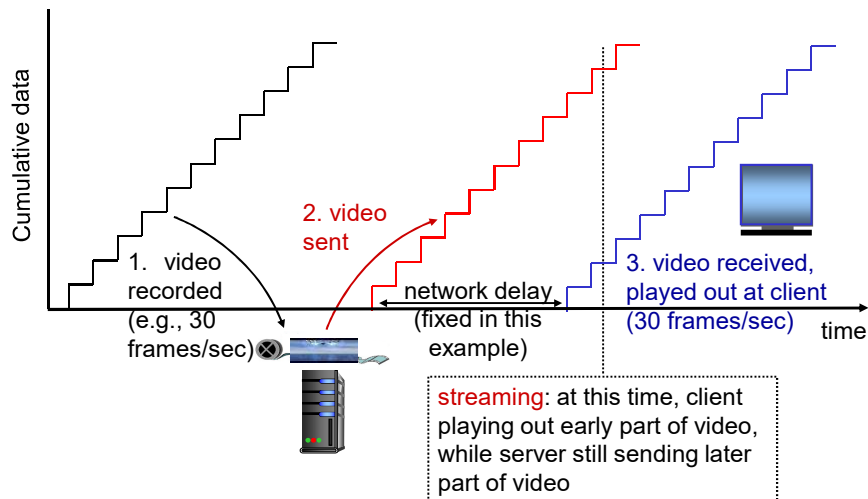- *streaming live* audio, video
  - e.g., live sporting event (futbol)

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

Streaming stored video:

1. video recorded (e.g., 30 frames/sec)

2. video sent

3. video received, played out at client (30 frames/sec)

network delay (fixed in this example)

Cumulative data

time

streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming stored video: challenges

- continuous playout constraint: once client playout begins, playback must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
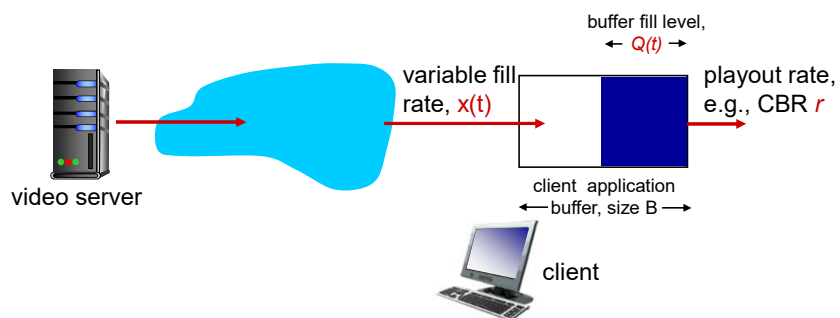 - video packets may be lost, retransmitted

Streaming stored video: revisited
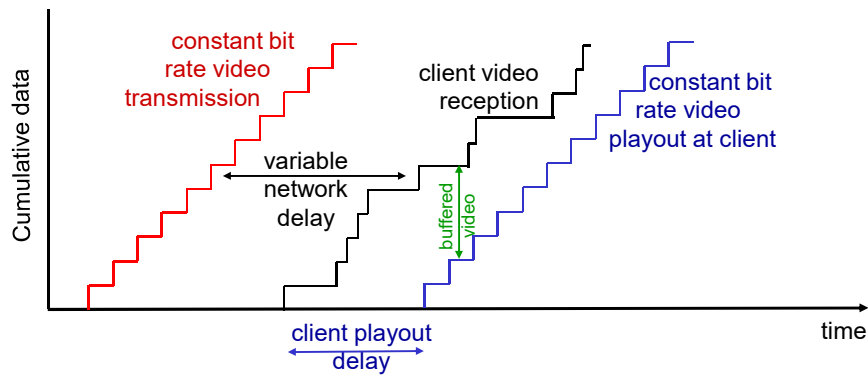
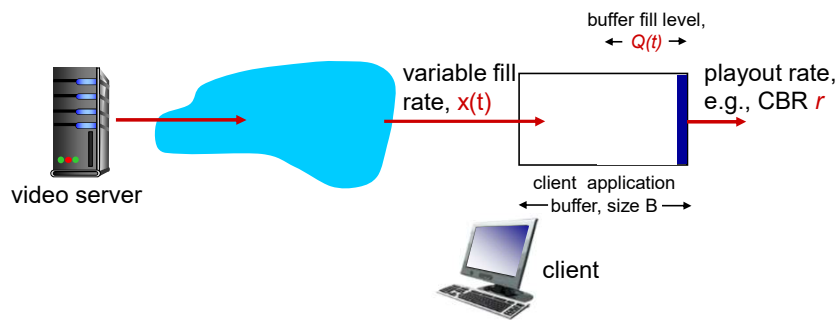- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

Client-side buffering, playout

buffer fill level,
$\leftarrow Q(t) \rightarrow$

variable fill rate, $x(t)$

playout rate, e.g., CBR r
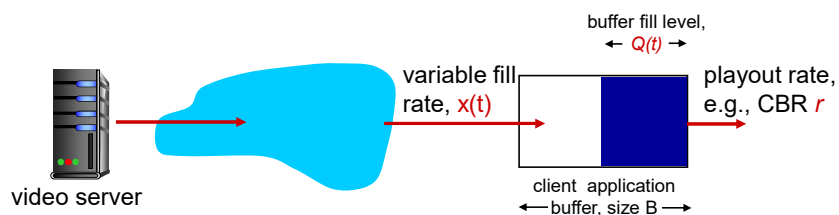
video server

client application
$\leftarrow$ buffer, size B $\rightarrow$

client

1. Initial fill of buffer until playout begins at t_p
2. playout begins at t_p,
3. buffer fill level varies over time as fill rate $x(t)$ varies and playout rate r is constant

Client-side buffering, playout

buffer fill level,
$\leftarrow Q(t) \rightarrow$

variable fill rate, $x(t)$

playout rate, e.g., CBR r

video server

client application
$\leftarrow$ buffer, size B $\rightarrow$

playout buffering: average fill rate ($\bar{x}$), playout rate (r):

- $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in $x(t)$
  - *initial playout delay tradeoff*: buffer starvation less likely with larger delay, but larger delay until user begins watching
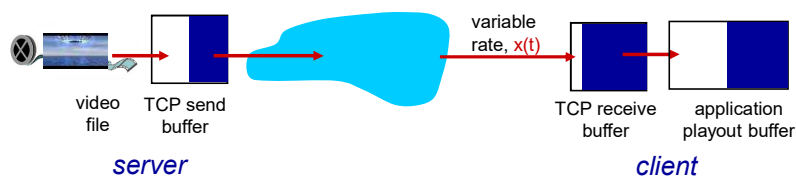
Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate
  - transmission rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, time permitting
- RTP [RFC 2326]: multimedia payload types
- UDP may *not* go through firewalls

Streaming multimedia: HTTP

- multimedia file retrieved via HTTP GET
- send at maximum possible rate under TCP

video file | TCP send buffer | variable rate, $x(t)$ | TCP receive buffer | application playout buffer

server   *client*

- fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

Voice-over-IP (VoIP)

- *VoIP end-end-delay requirement*: needed to maintain "conversational" aspect
  - higher delays noticeable, impair interactivity
  - < 150 msec: good
  - > 400 msec bad
  - includes application-level (packetization, playout), network delays
- *session initialization:* how does callee advertise IP address, port number, encoding algorithms?
- *value-added services:* call forwarding, screening, recording
- *emergency services:* 911

VoIP characteristics

- speaker's audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
  - pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP or TCP segment
- application sends segment into socket every 20 msec during talkspurt

VoIP: packet loss, delay

- *network loss:* IP datagram lost due to network congestion (router buffer overflow)
- *delay loss:* IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400 ms
- *loss tolerance:* depending on voice encoding, loss concealment, packet loss rates between 1% and 10% can be tolerated

Delay jitter
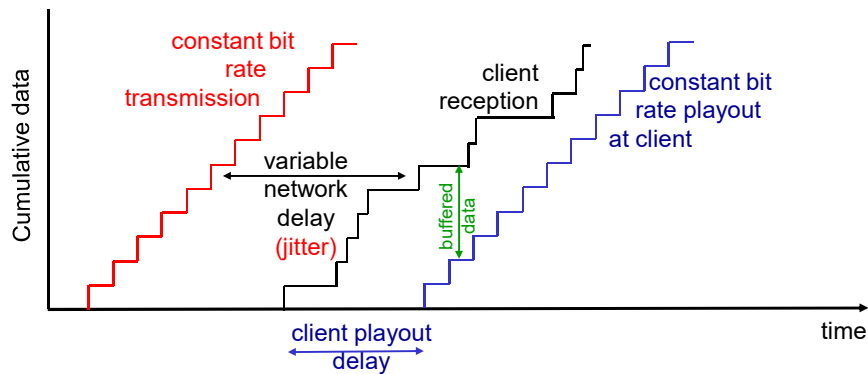
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

VoIP: fixed playout delay

- receiver attempts to playout each chunk exactly q msecs after chunk was generated.
  - chunk has time stamp t: play out chunk at $t+q$
  - chunk arrives after $t+q$: data arrives too late for playout: data "lost"
- tradeoff in choosing q:
  - *large q:* less packet loss
  - *small q:* better interactive experience

VoIP: fixed playout delay

- sender generates packets every 20 msec during talk spurt.
- first packet received at time r
- first playout schedule: begins at p
- second playout schedule: begins at p'

packets

packets
generated

packets
received

loss

playout schedule
p' - r

playout schedule
p - r

time

r

p

p'
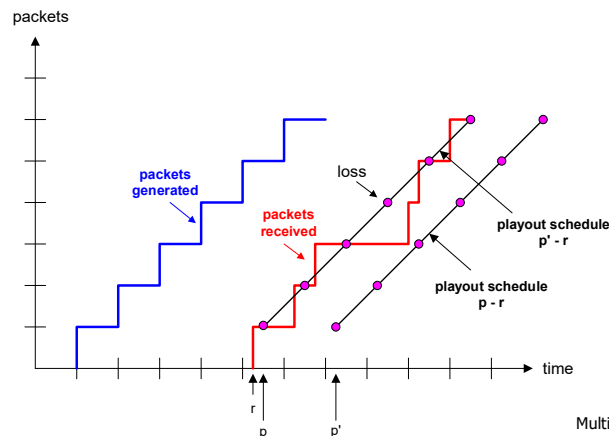
VoiP: recovery from packet loss (1)

Challenge: recover from packet loss given small tolerable delay between original transmission and playout
- each ACK/NAK takes ~ one RTT
- alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission (recall two-dimensional parity in Ch. 5)
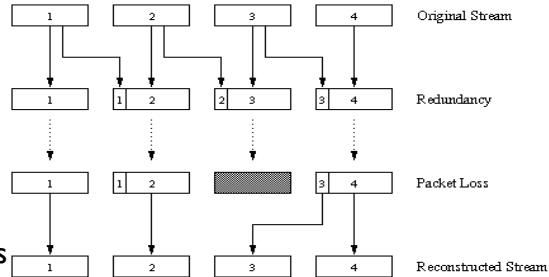
simple FEC
- for every group of n chunks, create redundant chunk by exclusive OR-ing n original chunks
- send $n+1$ chunks, increasing bandwidth by factor $1/n$
- can reconstruct original n chunks if at most one lost chunk from $n+1$ chunks, with playout delay
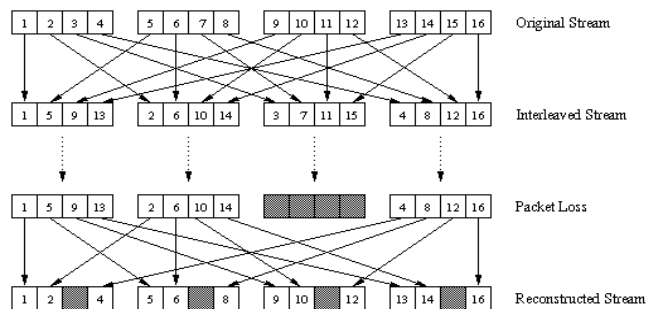
VoiP: recovery from packet loss (2)

another FEC scheme:

- "piggyback lower quality stream"
- send lower resolution audio stream as redundant information
- e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps
- non-consecutive loss: receiver can conceal loss
- generalization: can also append (n-1)st and (n-2)nd low-bit rate chunk

Original Stream

Redundancy

Packet Loss

Reconstructed Stream

VoiP: recovery from packet loss (3)

Original Stream

Interleaved Stream

Packet Loss

Reconstructed Stream

interleaving to conceal loss:

- audio chunks divided into smaller units, e.g. four 5 msec units per 20 msec audio chunk
- packet contains small units from different chunks

- if packet lost, still have *most* of every original chunk
- no redundancy overhead, but increases playout delay

Voice-over-IP: Skype

- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - clients: Skype peers connect directly to each other for VoIP call
  - super nodes (SN): Skype peers with special functions
 - overlay network: among SNs to locate SCs
  - login server

Skype clients (SC)

Skype login server

supernode (SN)

supernode overlay network

---
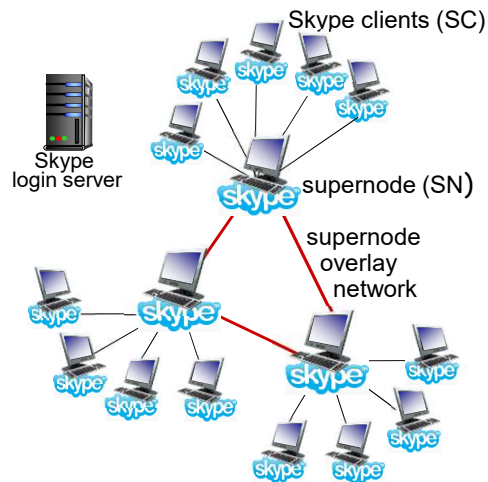
P2P voice-over-IP: Skype

Skype client operation:

1. joins Skype network by contacting SN (IP address cached) using TCP
2. logs-in (username, password) to centralized Skype login server
3. obtains IP address for callee from SN, SN overlay
   - or client buddy list
4. initiate call directly to callee

Skype login server
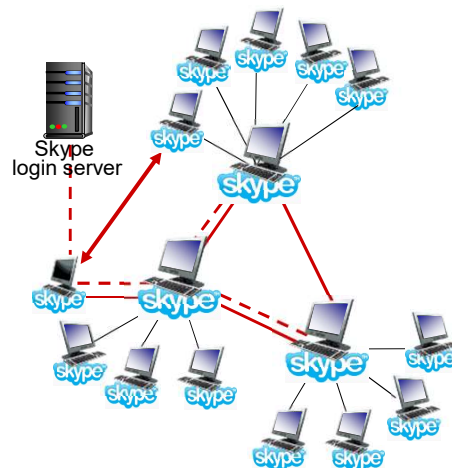
Skype: peers as relays

- *problem:* both Alice, Bob are behind "NATs"
  - NAT prevents outside peer from initiating connection to insider peer
  - inside peer *can* initiate connection to outside

- relay solution: Alice, Bob maintain open connection to their SNs
  - Alice signals her SN to connect to Bob
  - Alice's SN connects to Bob's SN
  - Bob's SN connects to Bob over open connection Bob initially initiated to his SN
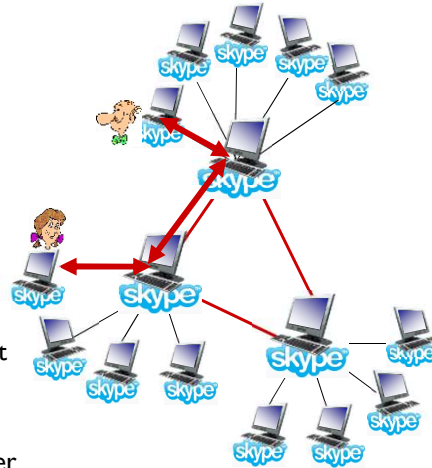
Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications: RTP

9.5 network support for multimedia

Real-Time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RFC 3550
- RTP packet provides
  - payload type identification
  - packet sequence numbering
  - time stamping

- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- interoperability: if two VoIP applications run RTP, they may be able to work together

RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:
- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping

Application
RTP
UDP
IP
Data Link
Physical

transport layer (RTP + UDP)

16
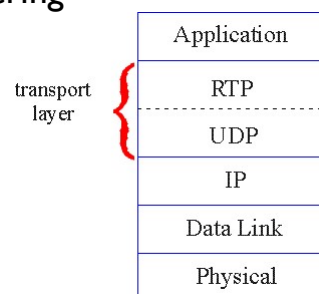
RTP example

example: sending 64 kbps PCM-encoded voice over RTP

- application collects encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk
- audio chunk + RTP header form RTP packet, which is encapsulated in UDP segment

- RTP header indicates type of audio encoding in each packet
  - sender can change encoding during conference
- RTP header also contains sequence numbers, timestamps

RTP and QoS

- RTP does *not* provide any mechanism to ensure timely data delivery or other QoS guarantees
- RTP encapsulation only seen at end systems (*not* by intermediate routers)
  - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

payload type (7 bits): indicates type of encoding currently being used. If sender changes encoding during call, sender informs receiver via payload type field

> Payload type 0: PCM mu-law, 64 kbps
> Payload type 3: GSM, 13 kbps
> Payload type 7: LPC, 2.4 kbps
> Payload type 26: Motion JPEG
> Payload type 31: H.261
> Payload type 33: MPEG2 video

sequence # (16 bits): increment by one for each RTP packet sent

- ❖ detect packet loss, restore packet sequence

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

- *timestamp field (32 bits long):* sampling instant of first byte in this RTP data packet
  - for audio, timestamp clock increments by one for each sampling period (e.g., each 125 usecs for 8 KHz sampling clock)
  - if application generates chunks of 160 encoded samples, timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.

- *SSRC field (32 bits long):* identifies source of RTP stream. Each stream in RTP session has distinct SSRC