CS 352
Video Streaming (Part 2)

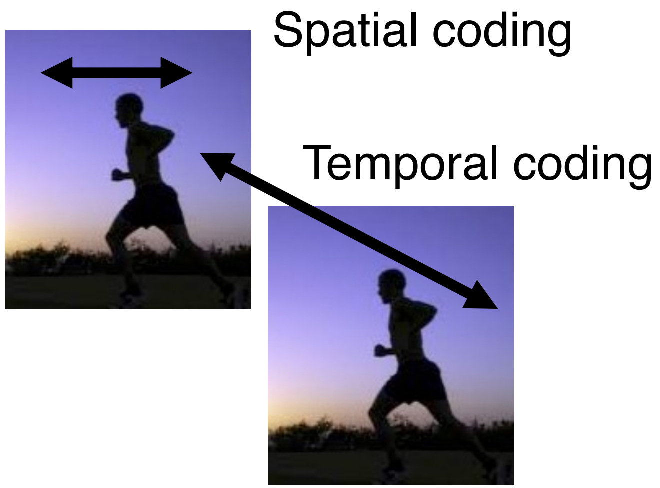
Lecture 9

http://www.cs.rutgers.edu/~sn624/352-F22

Srinivas Narayana

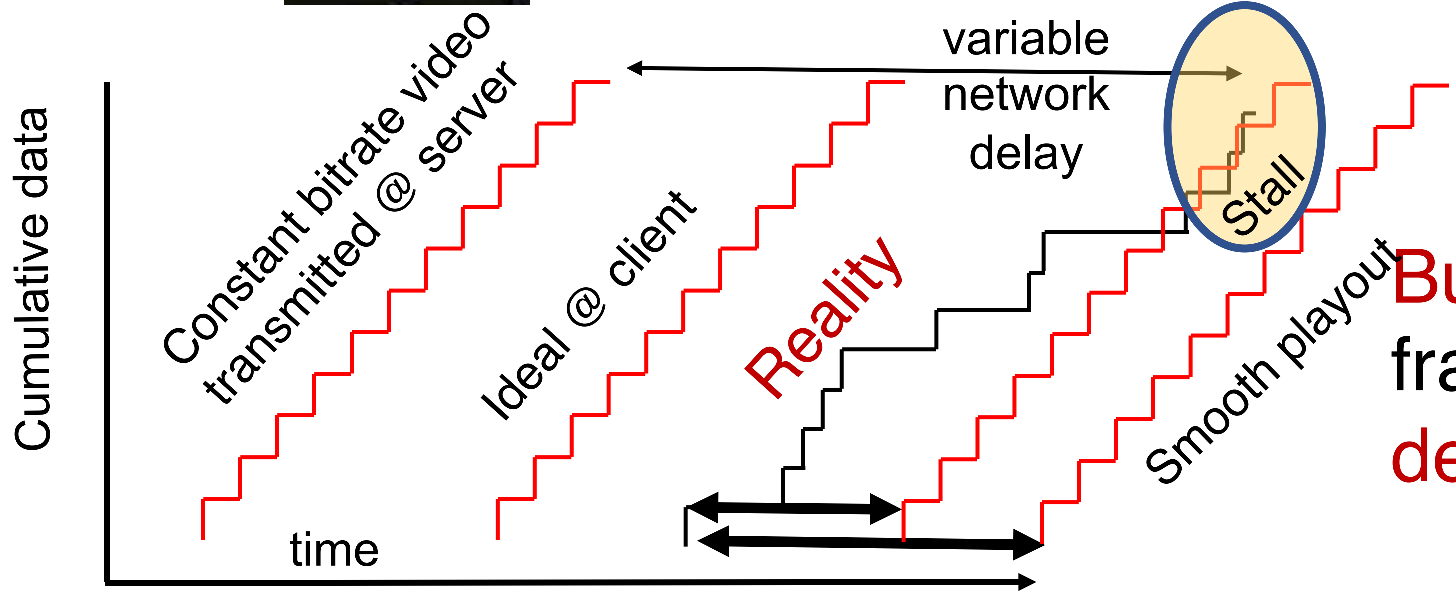
RUTGERS
UNIVERSITY | NEW BRUNSWICK

Quick recap of concepts



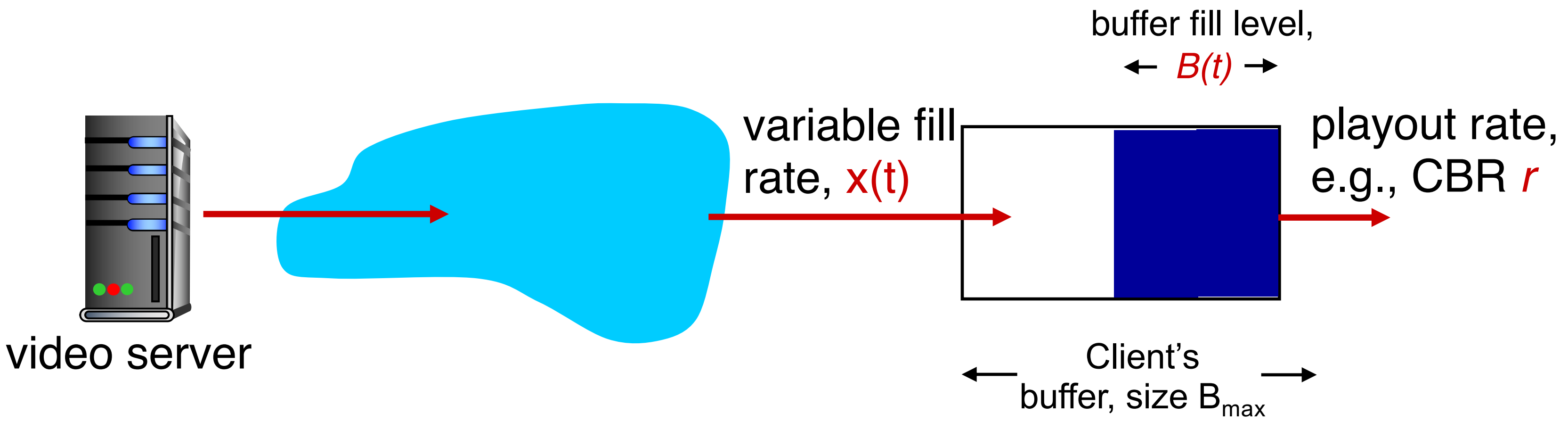
Video Bitrate

Bits played out per second (can vary over video's lifetime)



Buffer at the client to hold frames initially until playout delay t_p

Client-side buffering, playout

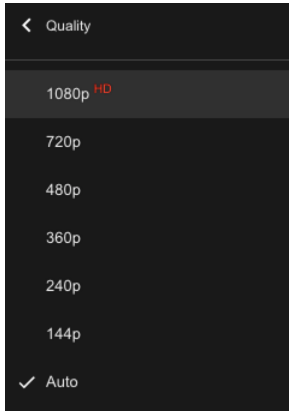


playout buffering: average fill rate ($\bar{x}$), playout rate (r):

- is $\bar{x} < r$ or $\bar{x} > r$ for a given network connection?
- It is hard to predict this in general!
  - Best effort network suffers long queues, paths with low bandwidth, …
- How to set playout rate r?
  - Too low a bit-rate r: video has poorer quality than needed
  - Too high a bit-rate r: buffer might empty out. Stall/rebuffering!

Adaptive bit–rate video

- Motivation: Want to provide high quality video experience, without stalls
- Observations:
  - Videos come in different qualities (average bit rates)
  - Versions of the video for different quality levels readily available
  - Different segments of video can be downloaded separately



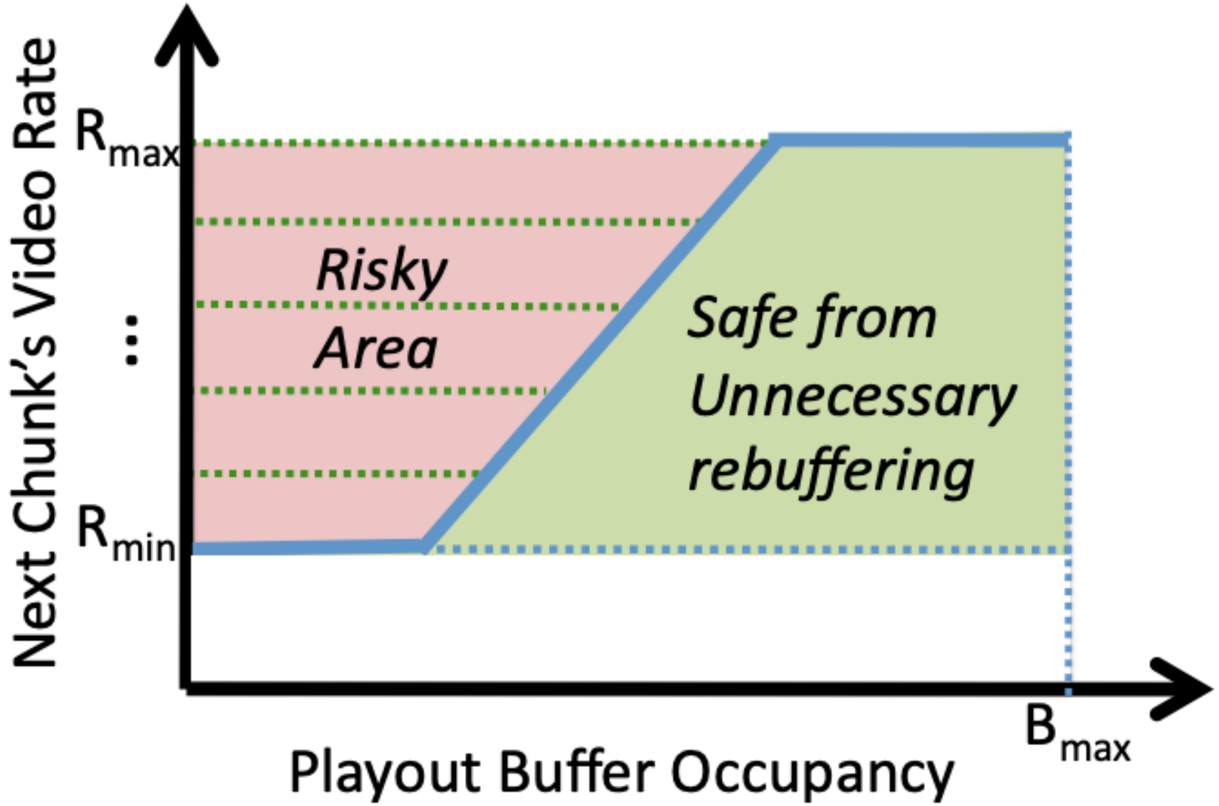
- Adapt bit rate per segment through collaboration between the video client (e.g., your browser) and the server (e.g., @ Netflix)
- Adaptive bit-rate (ABR) video: change the bit-rate (quality) of next video segment based on network and client conditions
- A typical strategy: Buffer-based rate adaptation

Buffer-based bit-rate adaptation

- Key idea: If there is a large stored buffer of video, optimize aggressively for video quality, i.e., high bit rates

- Else (i.e., buffer has low occupancy), avoid stalls by being conservative and ask for a lower quality (bit-rate)
  - Hope: lower bandwidth requirement of a lower quality stream is satisfiable more easily

Buffer-based bit-rate adaptation



A highly effective method to provide high video quality despite variable and intermittently poor network conditions.

Used by Netflix.

http://yuba.stanford.edu/~nickm/papers/sigcomm2014-video.pdf

A Buffer-Based Approach to Rate Adaptation

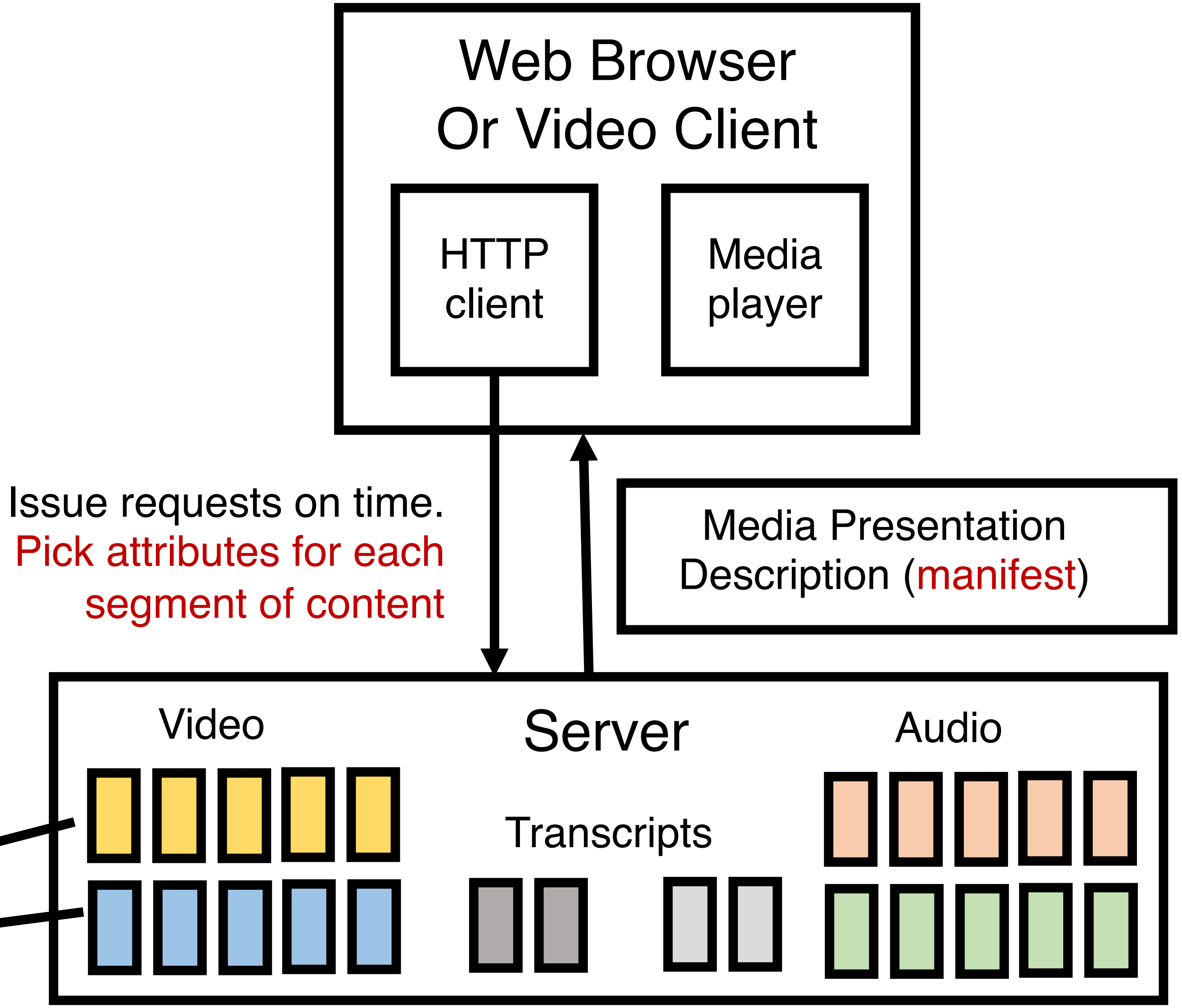
Dynamic Adaptive Streaming over HTTP (DASH)

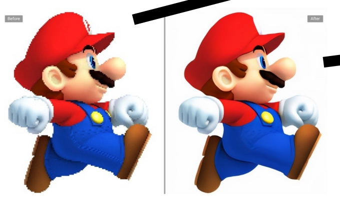
Streaming multimedia with DASH

- Dynamic Adaptive Streaming over HTTP
  - Used by Netflix and most popular video streaming services
- Adaptive: Perform video bit rate adaptation
  - It can be done on the client, or the server (with client feedback)
- Dynamic: Retrieve a single video from multiple sources
- The DASH video server is just a standard HTTP server
  - Provides video/audio content in multiple formats and encodings
- Leverage existing web-based infrastructure
  - DNS
  - CDNs!

DASH: Key ideas

- Content (video, audio, transcript, etc.) divided into segments (time)
- Algorithms to determine and request varying attributes (e.g., bitrate, language) for each segment
- Goal: ensure good quality of service, match user prefs, etc.





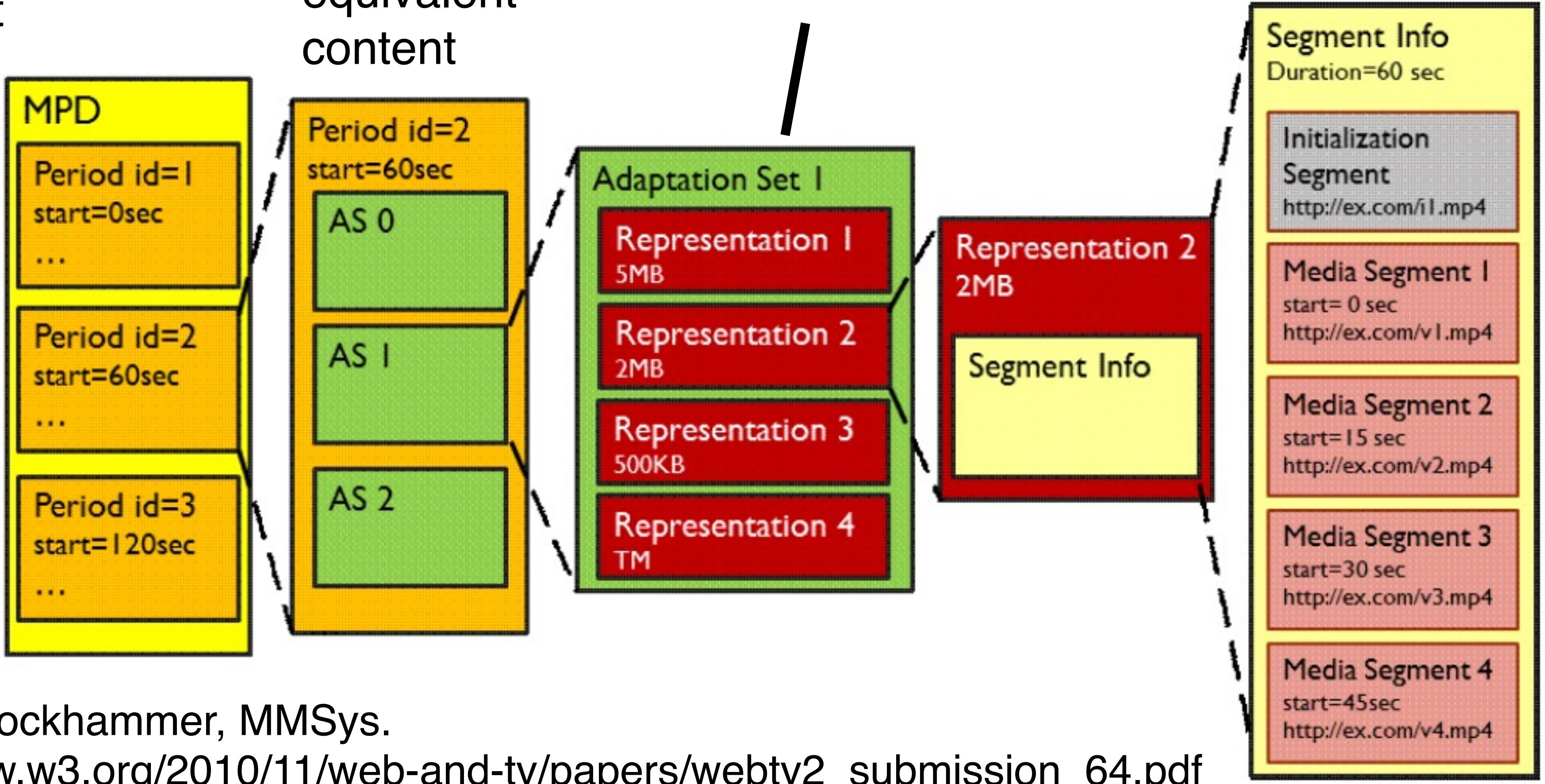
What does the manifest contain?

Periods: Durations of content

Adaptation set: functionally equivalent content

Representations: codecs, bit rates, etc.

Functionally equivalent: RSes of given AS
Functionally different: different ASes



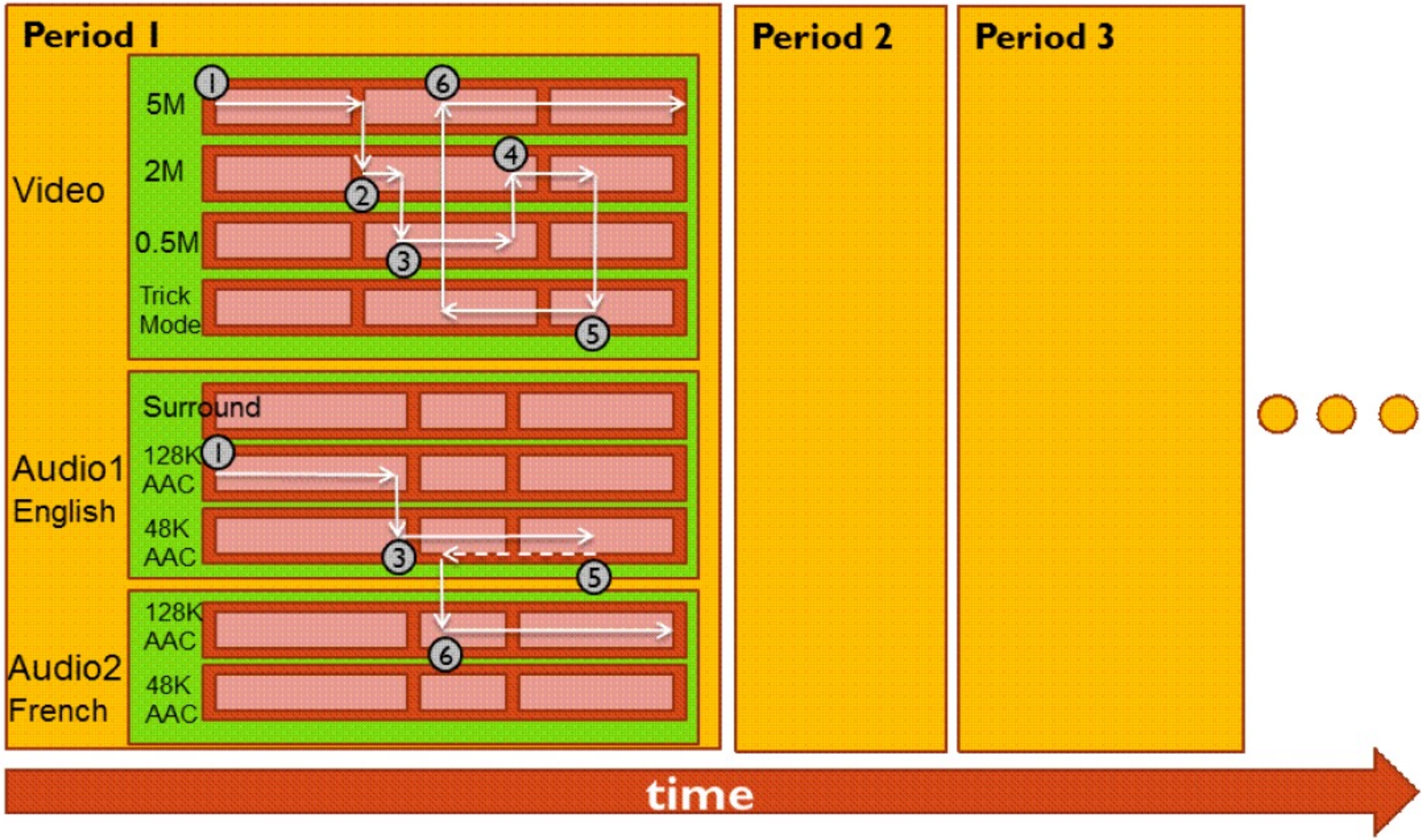
Multiple segments per representation

URL available for each segment

Byte ranges per segment (HTTP header for a range request)

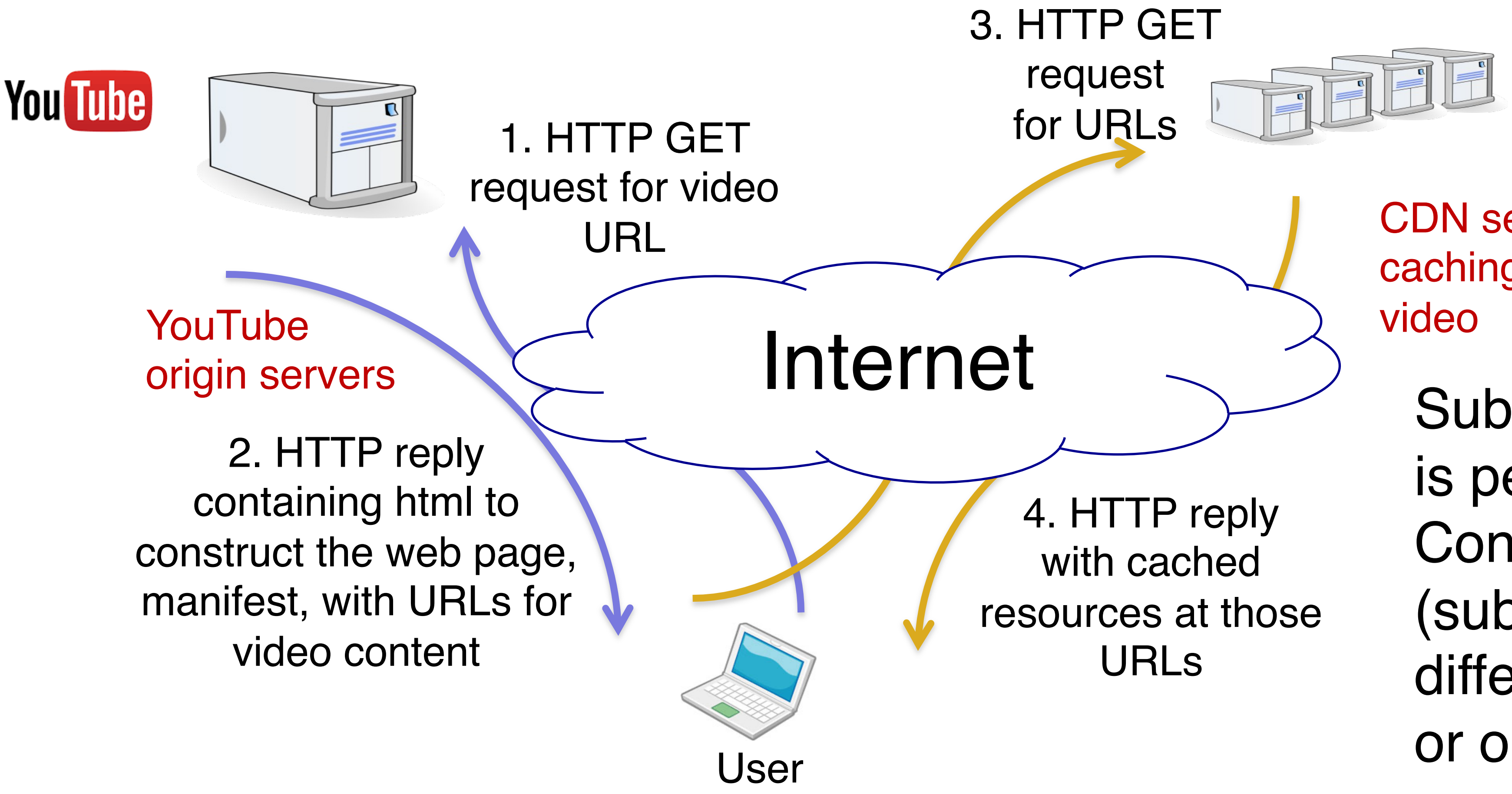
Source: Stockhammer, MMSys.
https://www.w3.org/2010/11/web-and-tv/papers/webtv2_submission_64.pdf

Dynamic changes in stream quality



Dynamic changes in stream location

- Just an HTTP request for an HTTP object



CDN DNS points user to best CDN server

Subtle: DNS granularity is per (sub)domain. Content from different (sub)domains can go to different CDN servers or origin

DASH reference player

- https://reference.dashif.org/dash.js/latest/samples/dash-if-reference-player/index.html

DASH Summary

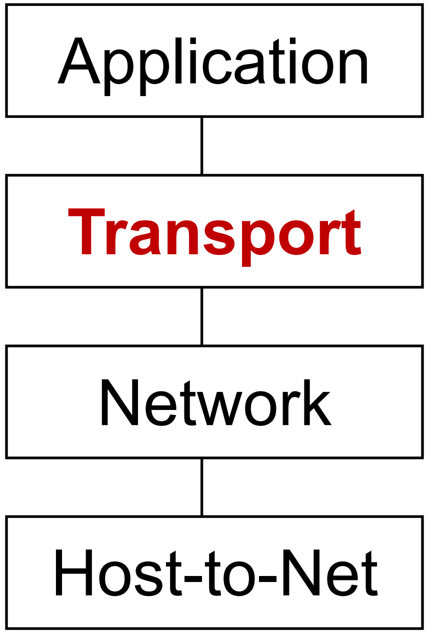
- Piggyback video on HTTP: widely used
- Enables independent HTTP requests per segment
  - Choose dynamic quality & preferences over time
  - Independent HTTP byte ranges
- Works well with CDNs
  - Fetch segments from locations other than the origin server
  - Fetch different segments from possibly different locations
- More resources on DASH
  - https://www.w3.org/2010/11/web-and-tv/papers/webtv2_submission_64.pdf
  - https://www.youtube.com/watch?v=xgowGnH5kUE

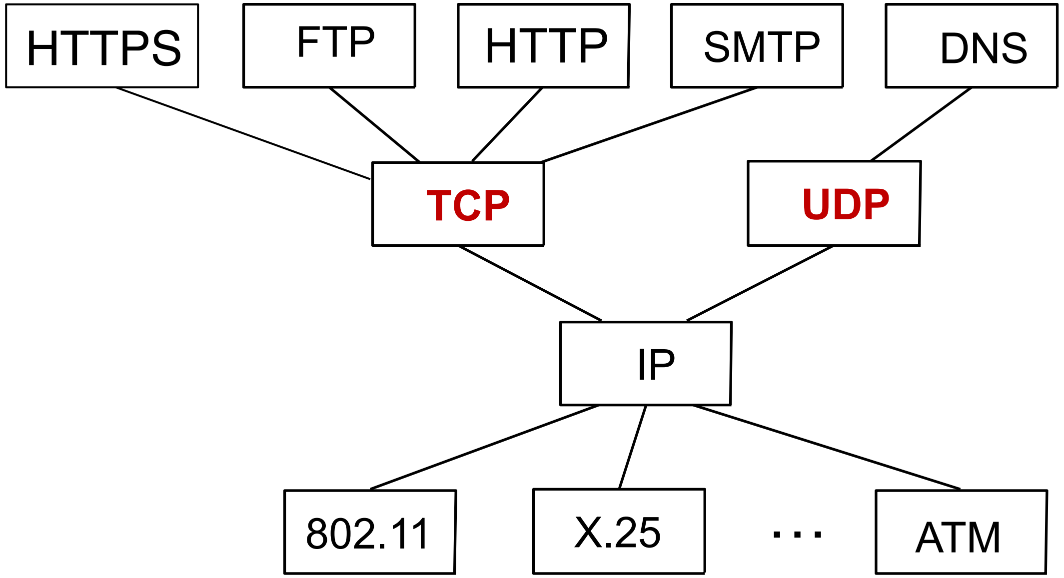
Application Layer: Wrap-up

- Name resolution, the web, mail, video
- Protocols built over the `socket()` abstraction
- Simple designs go a long way
  - Plain text protocols, header-based evolution, …
- Infrastructure for functionality, performance, …
  - Mail servers, CDNs, proxies, …
- Fit your apps to run on browsers: run almost anywhere (e.g. video)
- Apps are ultimately what users and most engineers care about
- BUT: if you don't understand what's under the hood, you risk bad design and poor performance for your Internet-facing applications

Transport

Transport

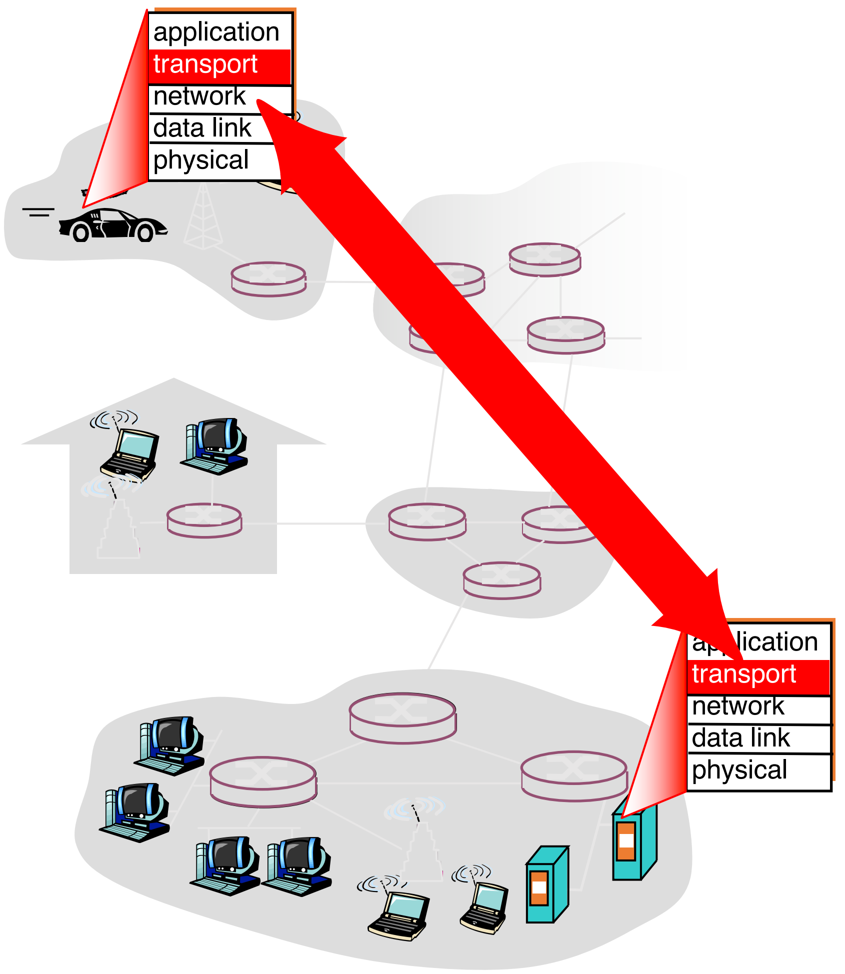

Application
Transport
Network
Host-to-Net


HTTPS
FTP
HTTP
SMTP
DNS
TCP
UDP
IP
802.11
X.25
…
ATM


Tp layer

Transport services and protocols

- Provide a communication abstraction between application processes
- Transport protocols run @ endpoints
  - send side: transport breaks app messages into segments, passes to network layer
  - recv side: reassembles segments into messages, passes to app layer
- Multiple transport protocols available to apps
  - Very popular in the Internet: TCP and UDP



Transport vs. network layer

- Network layer: abstraction to communicate between endpoints. Network layer provides best effort packet delivery to a remote endpoint.

- Transport layer: communication abstraction between processes. Delivers packets to the process.

Household analogy:

3 kids sending letters to 3 kids

- endpoints = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Alice and Bob who de/mux to in-house siblings
- network-layer protocol = postal service

Alice

Bob

Identifying a single conversation

- Application connections are identified by 4-tuple:

- Source IP address
- Source port
- Destination IP address
- Destination port

- In this analogy,

- Source address: the address of the first house
- Source port: name of a kid in the first house
- Destination address: the address of the second house
- Destination port: name of a kid in the second house

Demultiplexing Packets

Two popular transports

Transmission Control Protocol (TCP)

- Connection-based: the application remembers the other process talking to it.
- Suitable for longer-term, contextual data transfers, like HTTP, file transfers, etc.
- Guarantees: reliability, ordering, congestion control

User Datagram Protocol (UDP)

- Connectionless: app doesn't remember the last process or source that talked to it.
- Suitable for single req/resp flows, like DNS.
- Guarantees: basic error detection