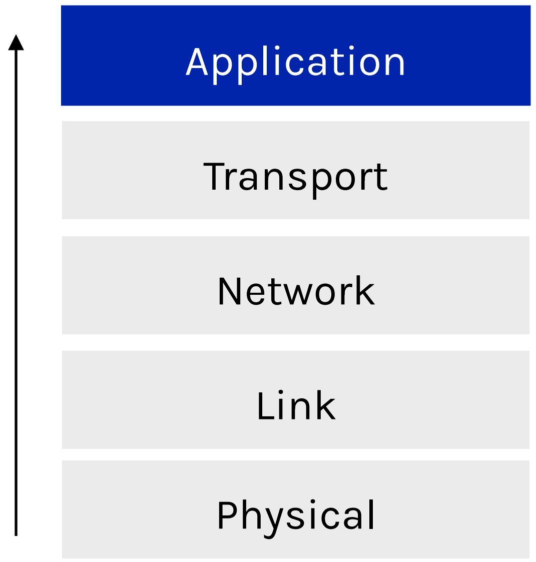
PADERBORN UNIVERSITY

Computer Networks (WS23/24)

L11: Video Streaming

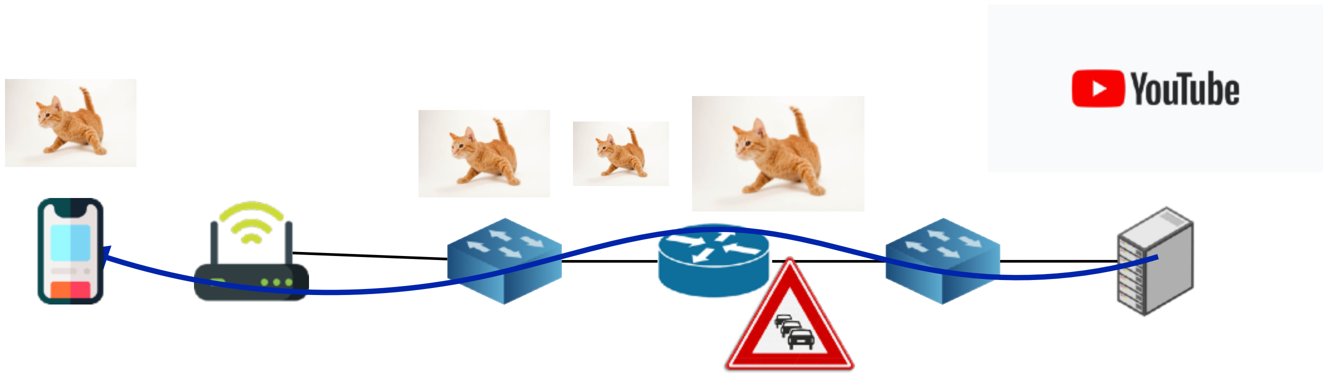
Prof. Dr. Lin Wang
Computer Networks Group
Department of Computer Science
Paderborn University

Learning objectives



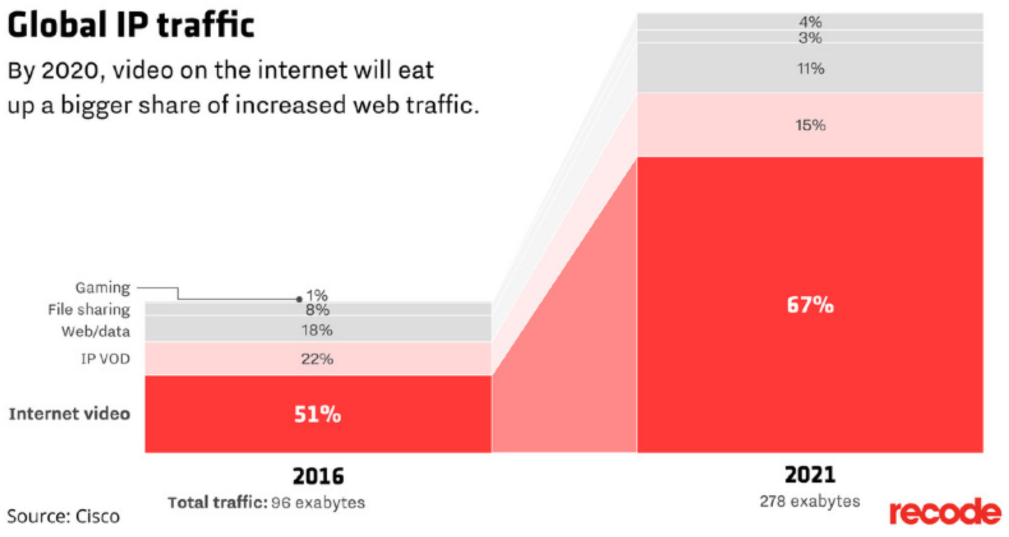
Part 2

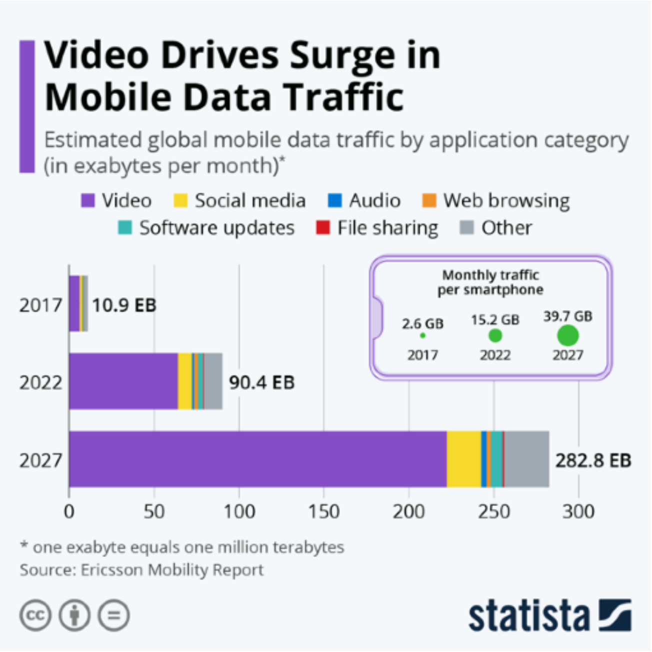
- Video streaming
- Video streaming protocols
- Bitrate adaptation algorithms
- Advanced topics: Netflix video serving, video analytics



Video Streaming Basics

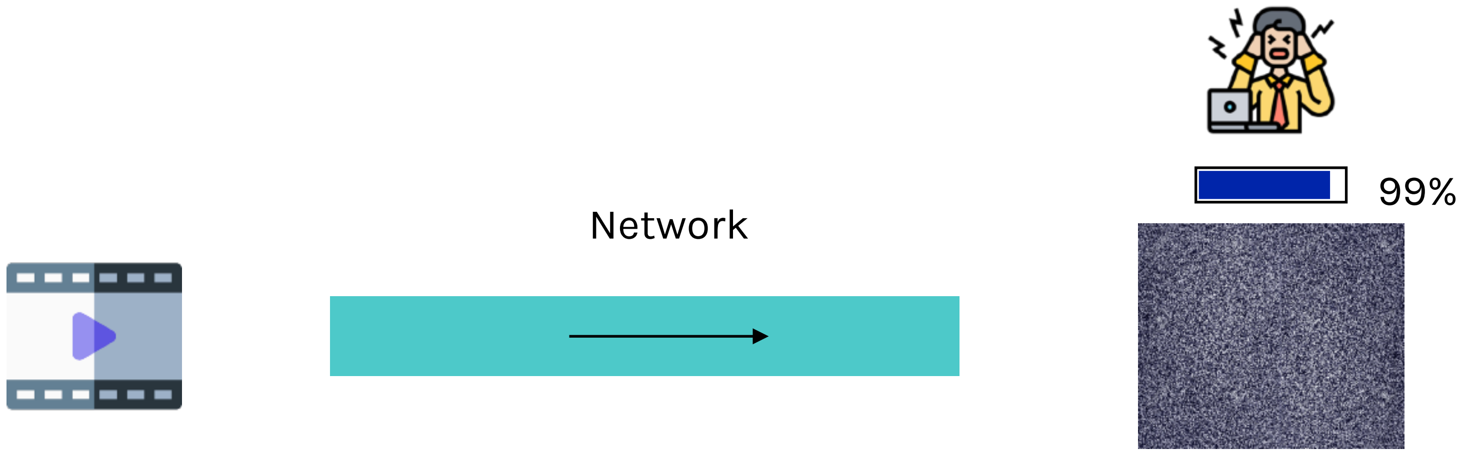
Video content has been dominating the Internet





Video traffic is dominant nowadays: By 2027, Ericsson expects video content to account for almost 80% of mobile traffic, which is projected to triple once more in the next five years

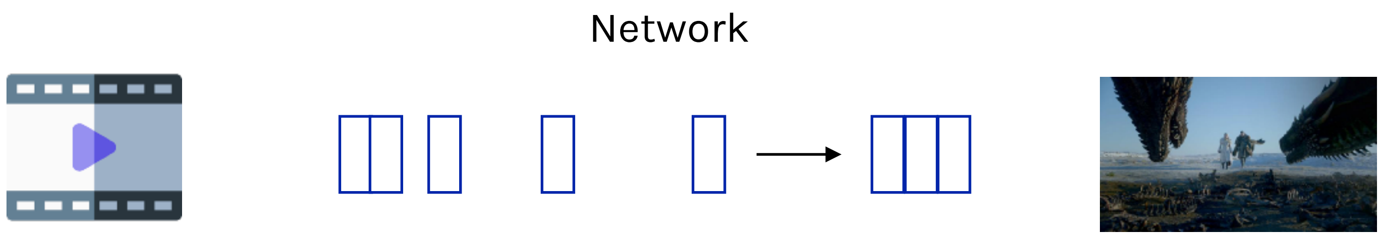
Pre-streaming era



Download the whole video file (e.g., FTP) and play it when the download is finished

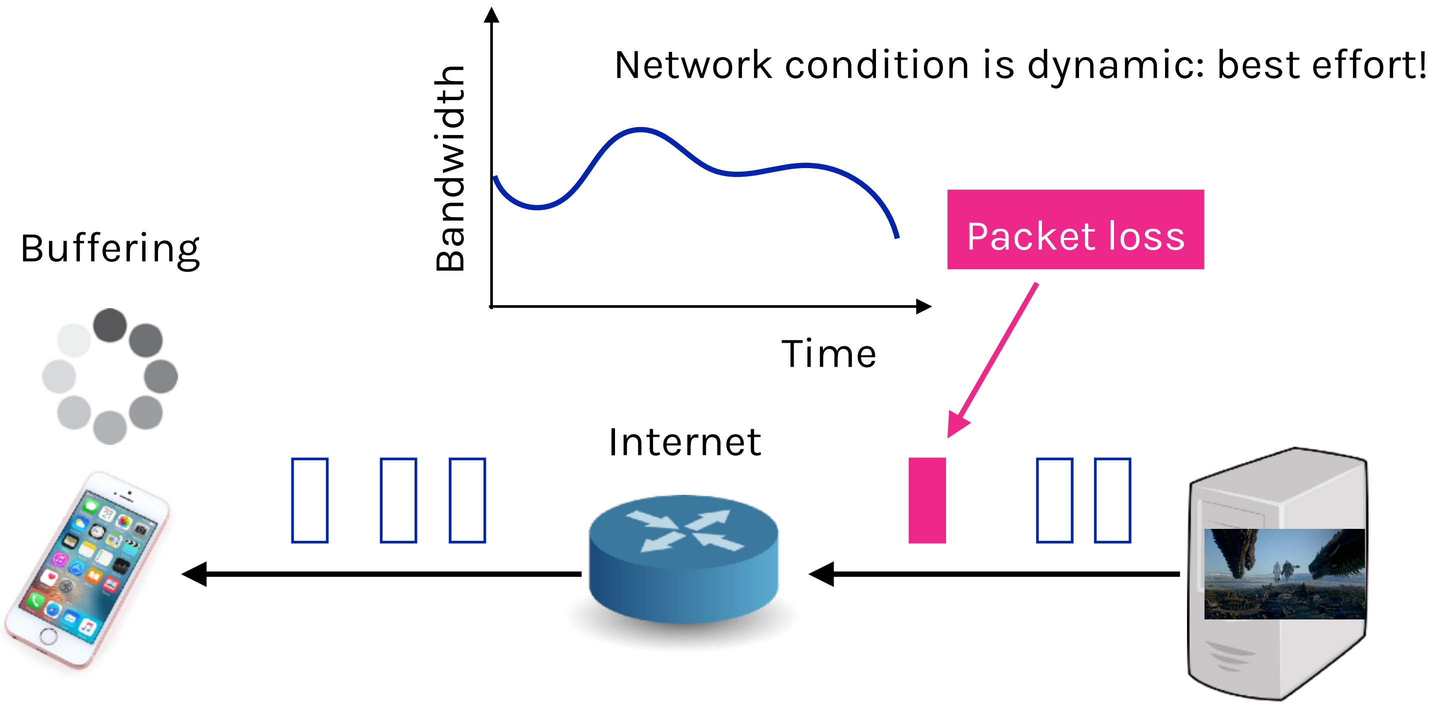
Problem: long waiting time and susceptible to glitches!

Streaming era



Chunk the video into small segments and stream from any segment

Challenges in video streaming



How to address/mitigate this issue?

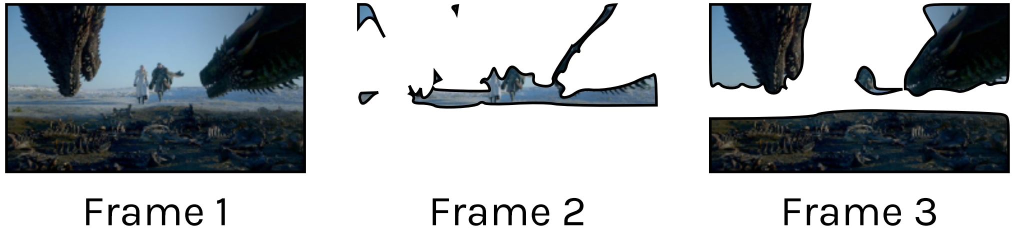
Video compression

Reduce the data volume to be transmitted while keeping the video quality

Techniques:

- Frame-level compression: resize/encode the image
- Video-level compression: encode the images across time (calculating deltas)

Frame-level compression



Video-level compression

Frame-level compression

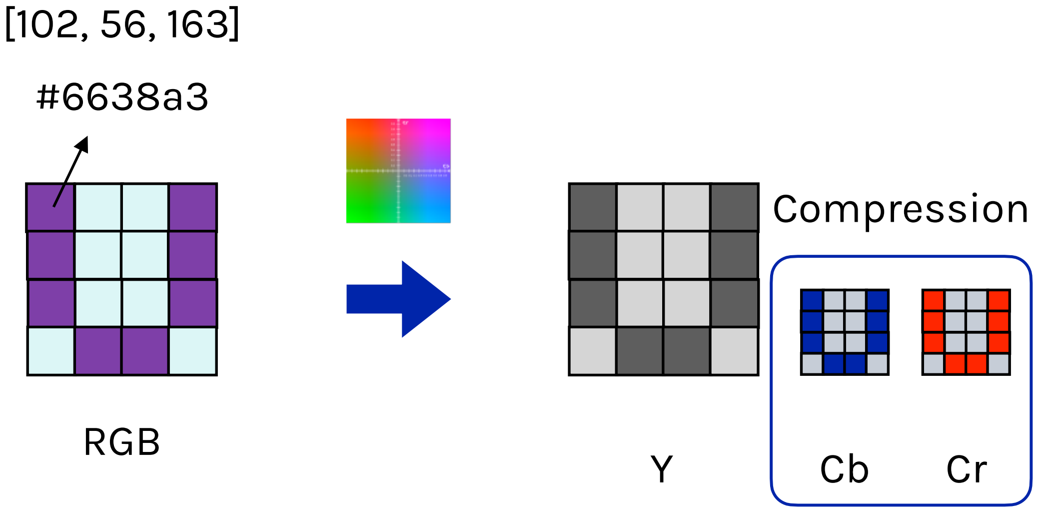
JPEG compression

- Changes RGB to YC_bC_r
- Y: luminance, C_bC_r are chrominance
  (blue and red relative to the green color)

Why this change?

- Human eyes are less sensitive to chrominance than to luminance

JPEG reduces sizes of C_b and C_r: quantization by 4



What is the compression ratio?

Frame-level compression

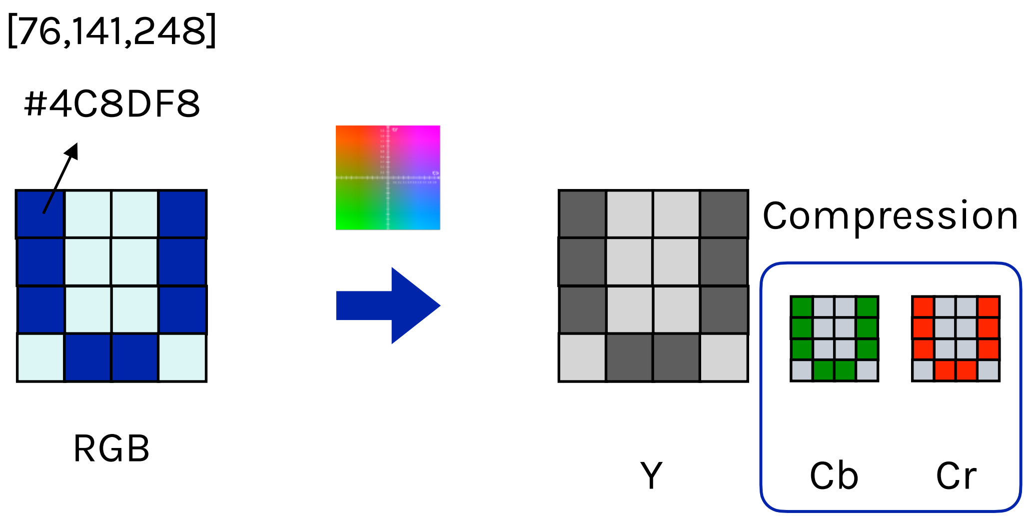
JPEG compression

- Changes RGB to YC_bC_r
- Y: luminance, C_bC_r are chrominance (blue and red relative to the green color)

Why this change?

- Human eyes are less sensitive to chrominance than to luminance

JPEG reduces sizes of C_b and C_r: quantization by 4

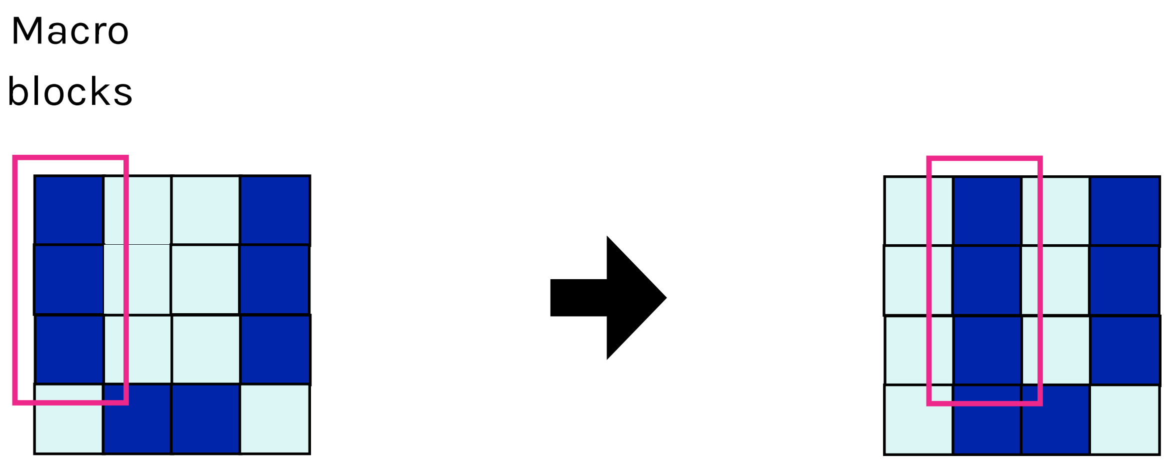


Compression ratio calculation:

- Size before compression: $16 \times 3 = 48$ bytes
- Size after compression: $16 + 4 + 4 = 24$ bytes
- Compression ratio: $48/24 = 2$

Video-level compression

Remove temporal redundancy by keeping track of the relative differences (deltas) between frames

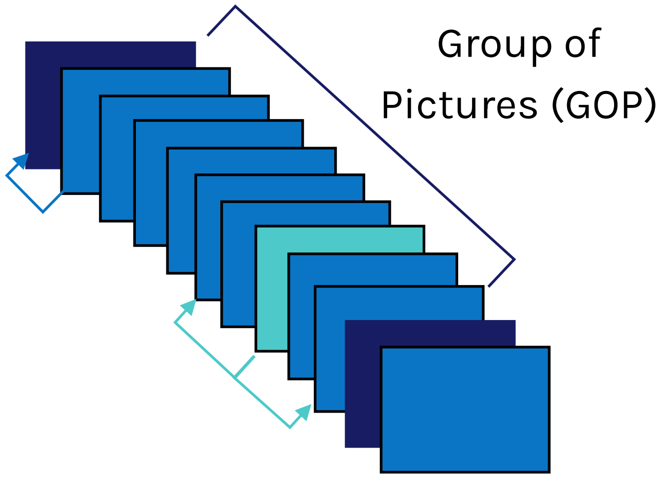


Compressibility highly depends on the content

https://en.wikipedia.org/wiki/Macroblock

Video-level compression

Remove temporal redundancy by keeping track of the relative differences (deltas) between frames



I (intra-coded) frame: self-contained, e.g., JPEG

P (predictive) frame: looks back to I and P frames for prediction

B (bidirectional) frame: looks forward and backward to other frames

I frames are the largest, **P** frames are medium-size, and **B** frames are the smallest

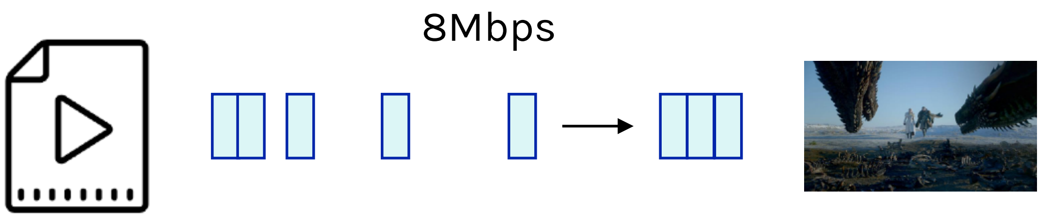
Bitrate

Measures the data size per unit time

- Amount of data used to encode video (or audio) per second, e.g., Mbps, Kbps

Bitrate affects both the file size and the quality of the video

- Affect the required bandwidth when streaming the video over the network



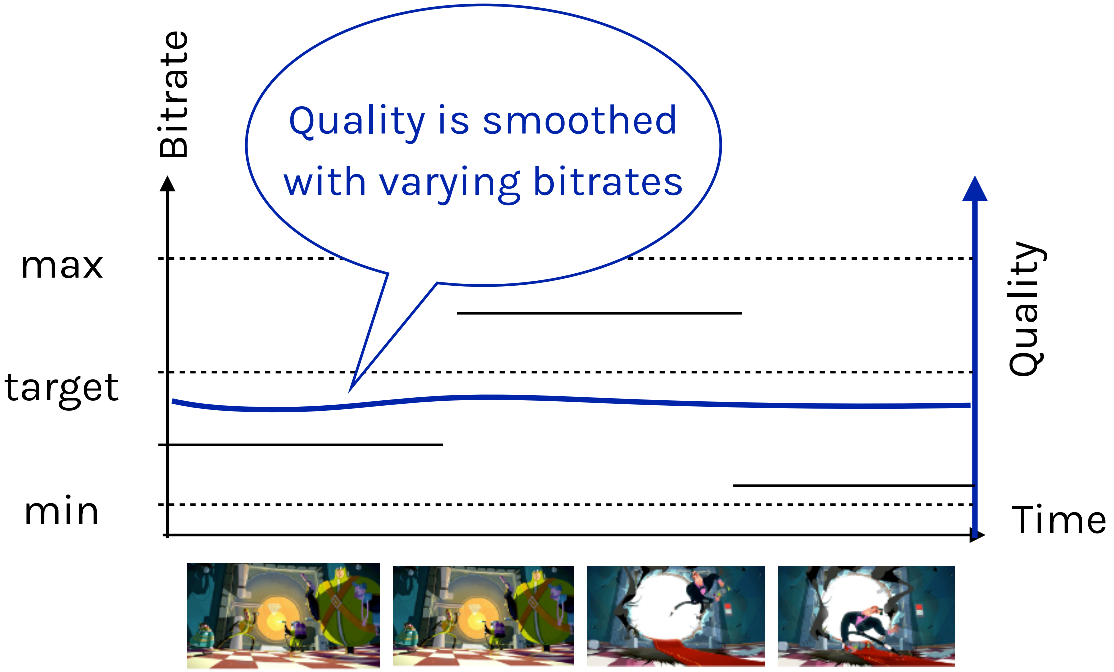
Live demo: https://reference.dashif.org/dash.js/latest/samples/dash-if-reference-player/index.html

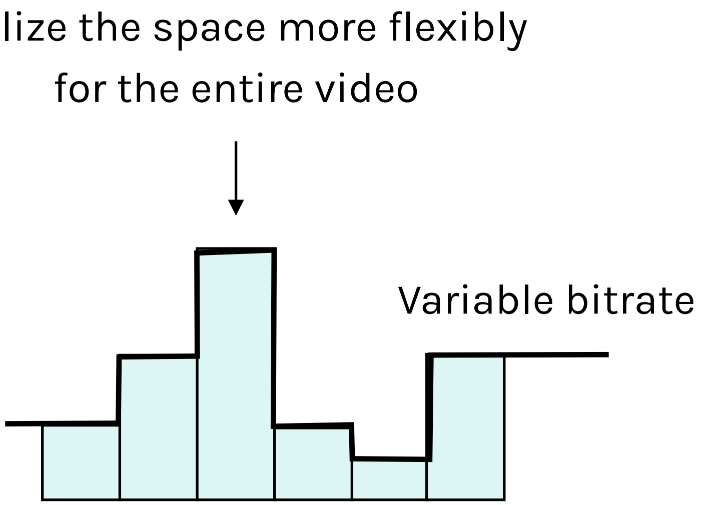
Variable bitrate (VBR)

Encode video with varying bitrates

- Smooth out the quality
- Higher bitrate for more complex segments, not friendly with streaming over a network

VBR algorithms for encoder and decoder are more complex and typically require support from the hardware

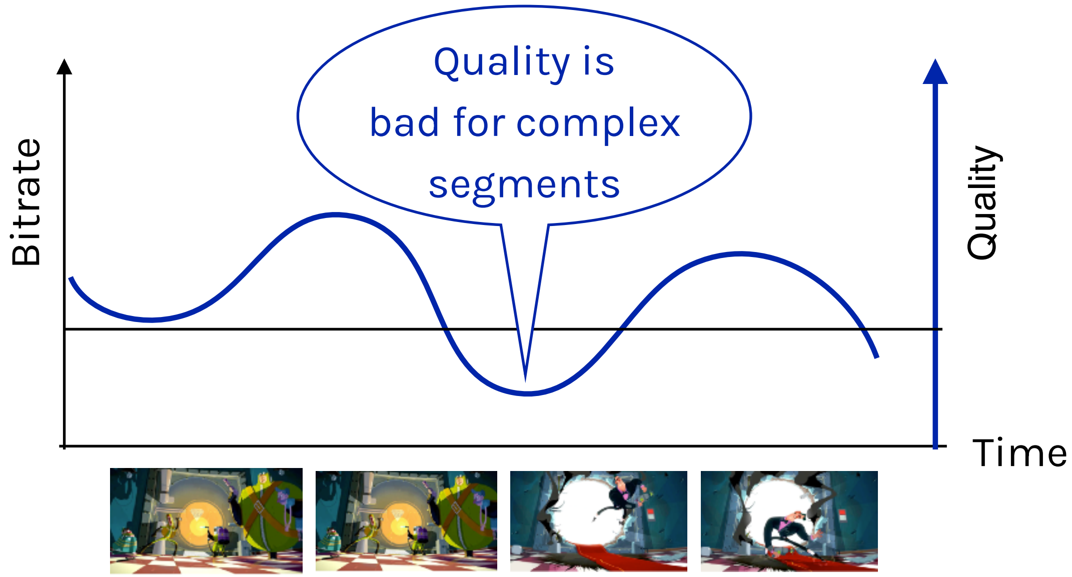


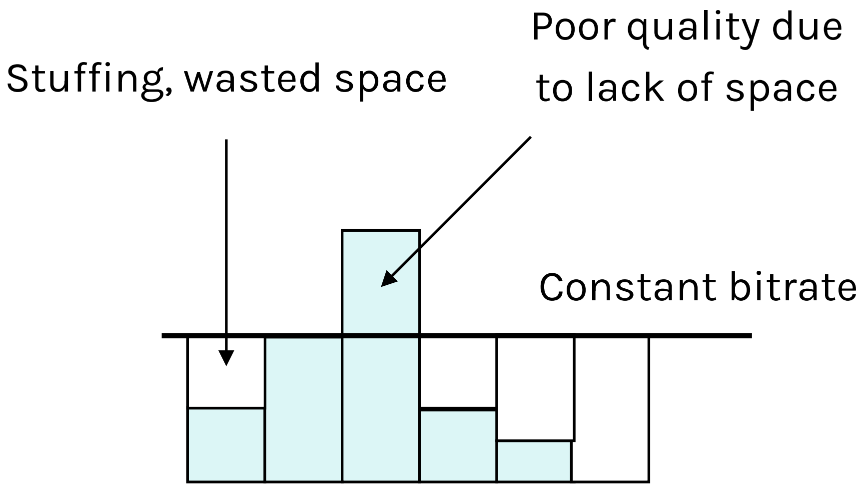


Constant bitrate (CBR)

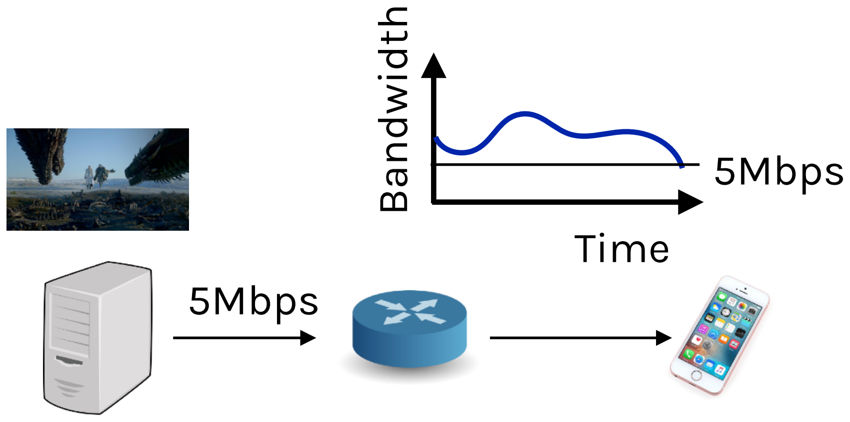
Compress video with a constant bitrate

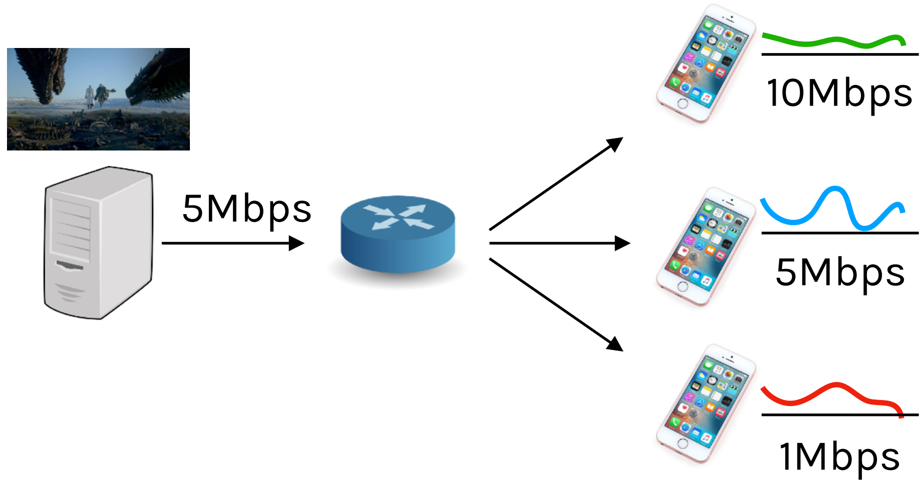
- **Constant bitrate** → constant compression ratio → **varying quality**
- In H.264, quality is worse when the motion is higher due to the larger deltas





Video streaming with CBR

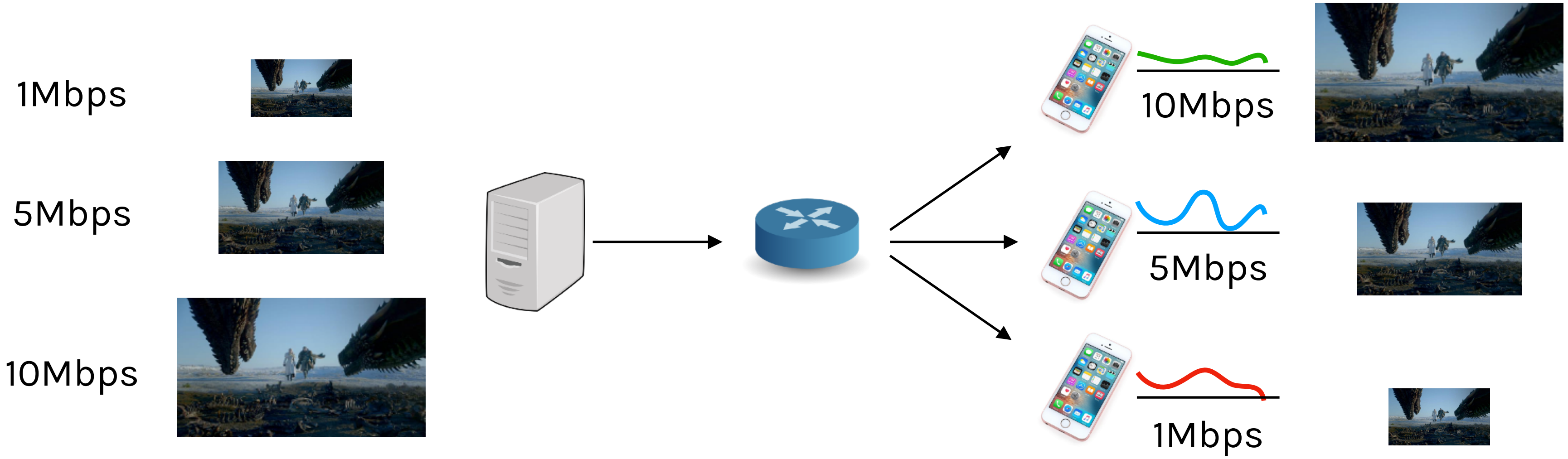




For a single user, CBR is sufficient, though not perfect

CBR is not efficient when multiple users with different bandwidth availabilities are present

CBR improvement

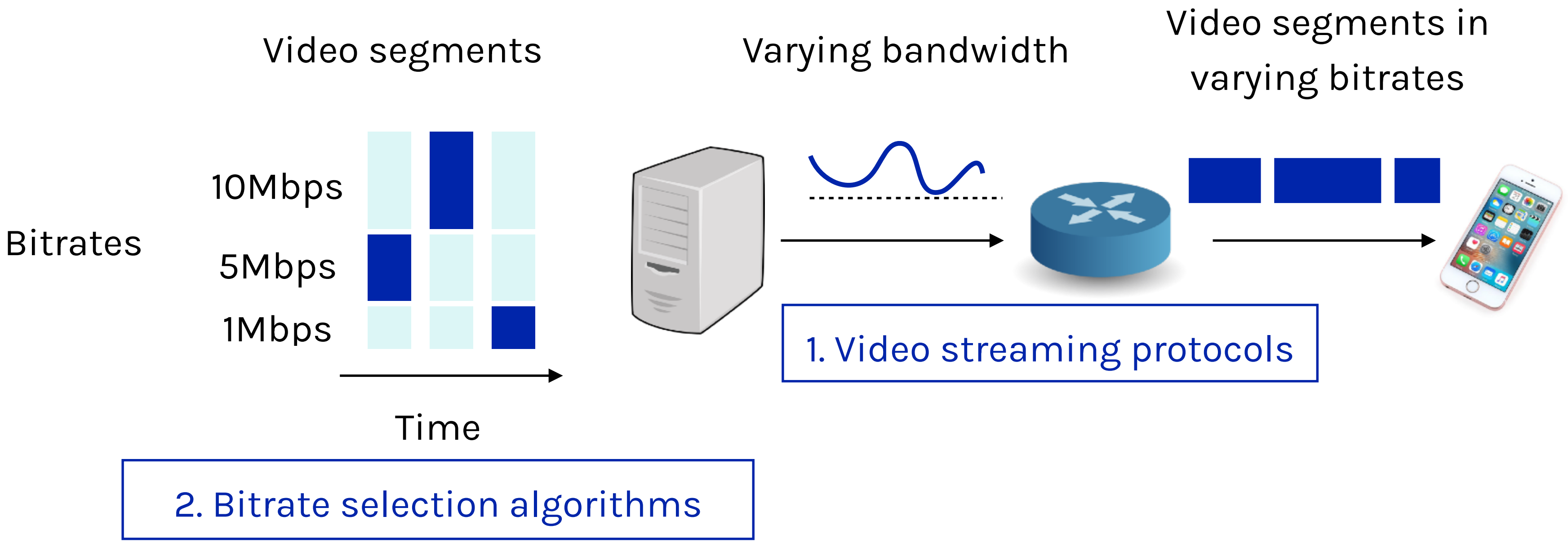


Encode the video with different CBRs at the streaming server and choose a suitable CBR based on the real-time bandwidth availability

Adaptive bitrate (ABR)

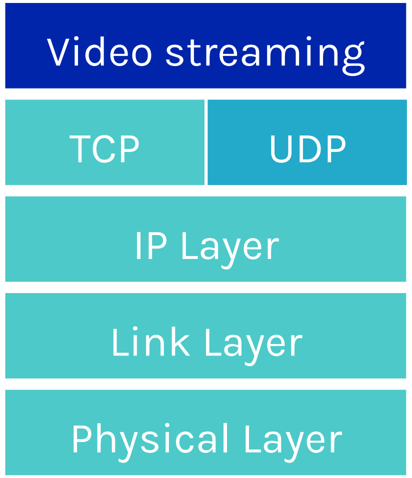
Main idea:

- **Chop the video into small segments** and encode the segments with different bitrates
- **Adaptively select the bitrate for each segment** in streaming for each user

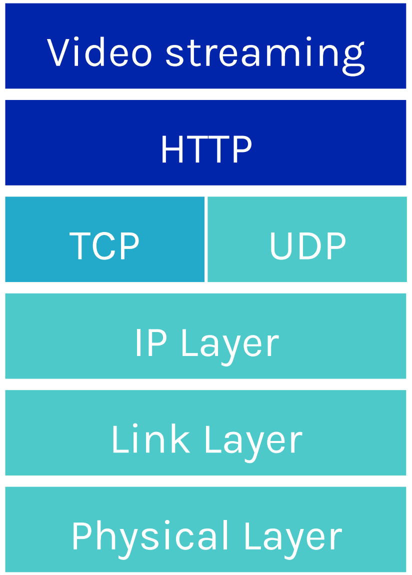


Video Streaming Protocols

Video streaming protocols



Majority video streaming protocols are based on UDP in favor of timeliness instead of reliability



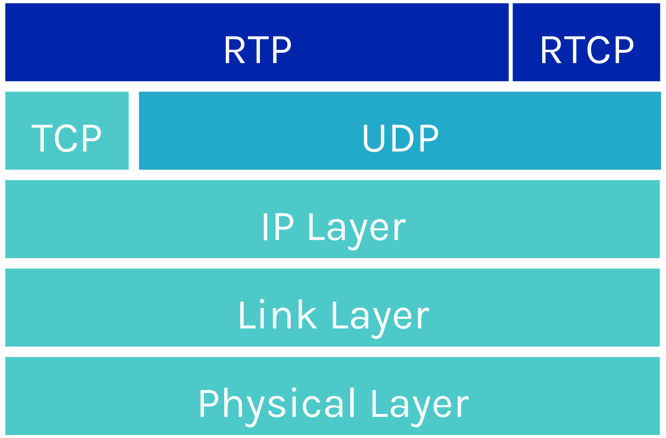
Modern video streaming protocols are based on HTTP

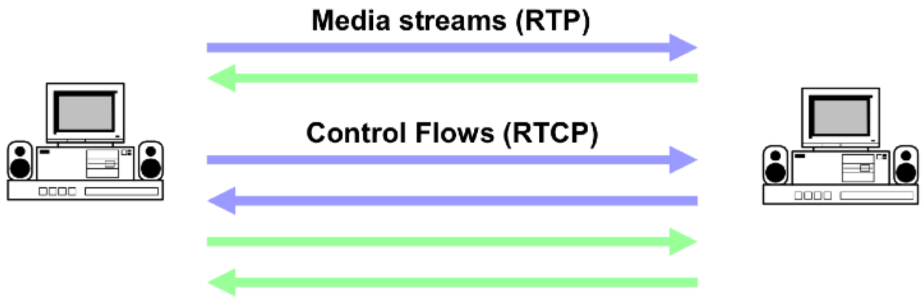
Real-time Transport Protocol (RTP)

RFC 1889 RFC 3550

Based on UDP

- Primary standard for audio/video transport in IP networks, widely used for **real-time multimedia applications** such as voice over IP, audio over IP, **WebRTC** (uses SRTP), and IP television
- Comes with a **control protocol, RTCP**, for QoS feedback and synchronization between media streams, account for around 5% of total bandwidth usage





RTP packet header

V=2	P	X	CC	M	PT	Sequence number
Timestamp						
Synchronization source (SSRC) identifier						
Contributing source (CSRC) identifiers ⋮						
Extension header						
RTP payload						

- **Sequence number (16 bits):** used for packet loss detection or packet reordering, initially randomized
- **Timestamp (32 bits):** used by the receiver to play back the received samples at appropriate time and interval (e.g., use a clock of 90 kHz for a video stream)
- **SSRC (32 bits):** uniquely identify the source of a stream
- **CSRC (32 bits):** enumerate contributing sources to a stream which has been generated from multiple sources

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.1.1	10.2.2.2	RTP	121	PT=DynamicRTP-Type-111, SSRC=0xC2B13255, Seq=19591, Time=2760404098
2	0.000037	10.2.2.2	10.1.1.1	RTP	121	PT=DynamicRTP-Type-111, SSRC=0xB5770A56, Seq=4305, Time=4131840
3	0.020622	10.1.1.1	10.2.2.2	RTP	131	PT=DynamicRTP-Type-111, SSRC=0xC2B13255, Seq=19592, Time=2760405058
4	0.020653	10.2.2.2	10.1.1.1	RTP	131	PT=DynamicRTP-Type-111, SSRC=0xB5770A56, Seq=4306, Time=4132800
5	0.025986	10.1.1.1	10.2.2.2	RTP	1190	PT=DynamicRTP-Type-96, SSRC=0x69E8BDC, Seq=24102, Time=3068471093
6	0.026109	10.1.1.1	10.2.2.2	RTP	1190	PT=DynamicRTP-Type-96, SSRC=0x69E8BDC, Seq=24103, Time=3068471093
7	0.026153	10.1.1.1	10.2.2.2	RTP	1190	PT=DynamicRTP-Type-96, SSRC=0x69E8BDC, Seq=24104, Time=3068471093
8	0.026290	10.1.1.1	10.2.2.2	RTP	1190	PT=DynamicRTP-Type-96, SSRC=0x69E8BDC, Seq=24105, Time=3068471093

Control in RTP: RTCP

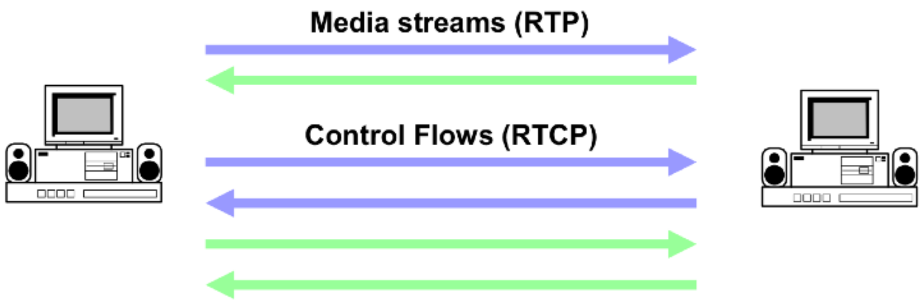
RTP/RTCP has no support for ABR!

Receiver constantly measure transmission quality

- Delay, jitter, packet loss, RTT

Regular control information exchange between senders and receivers

- Feedback to sender (receiver report)
- Feed forward to recipients (sender report)



Allow applications to adapt to current QoS

- Limiting a flow or using a different codec

Limited overhead: a small fraction, e.g., 5% max. of total bandwidth per RTP session

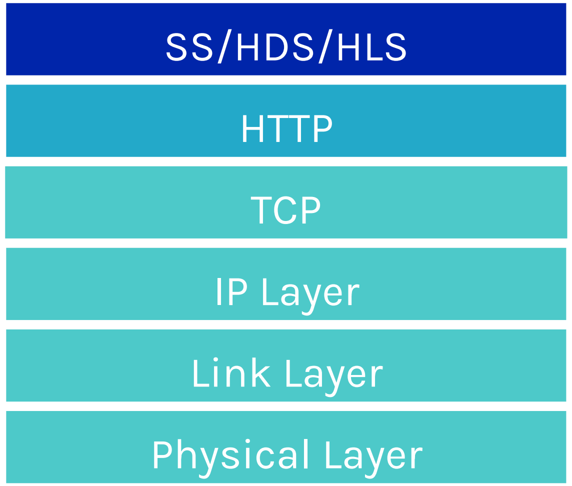
Video streaming protocols based on HTTP

Three major players

- Microsoft Smooth Streaming
- Adobe HTTP Dynamic Streaming (HDS)
- Apple HTTP Live Streaming (HLS)

Each has a proprietary format and its own ecosystem

Bad for the industry such as CDN providers like Akamai since every functionality has to be implemented three times

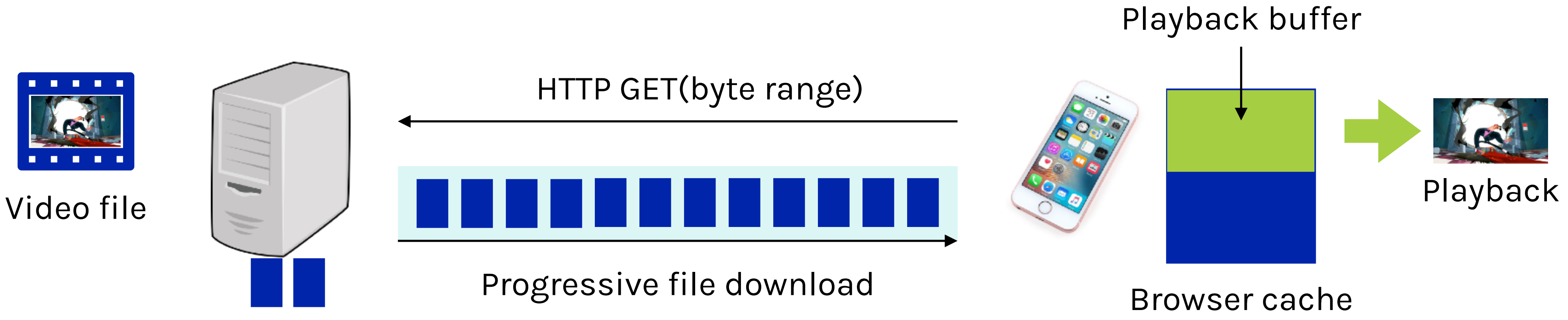


Why HTTP?

HTTP 1.1+ supports progressive download

- Prevalent form of web-based media delivery for video share sites
- Progressive = playback begins while download is in progress (byte range request)

Compatible with middleboxes (e.g., firewalls) on the Internet



Different HTTP-based protocols

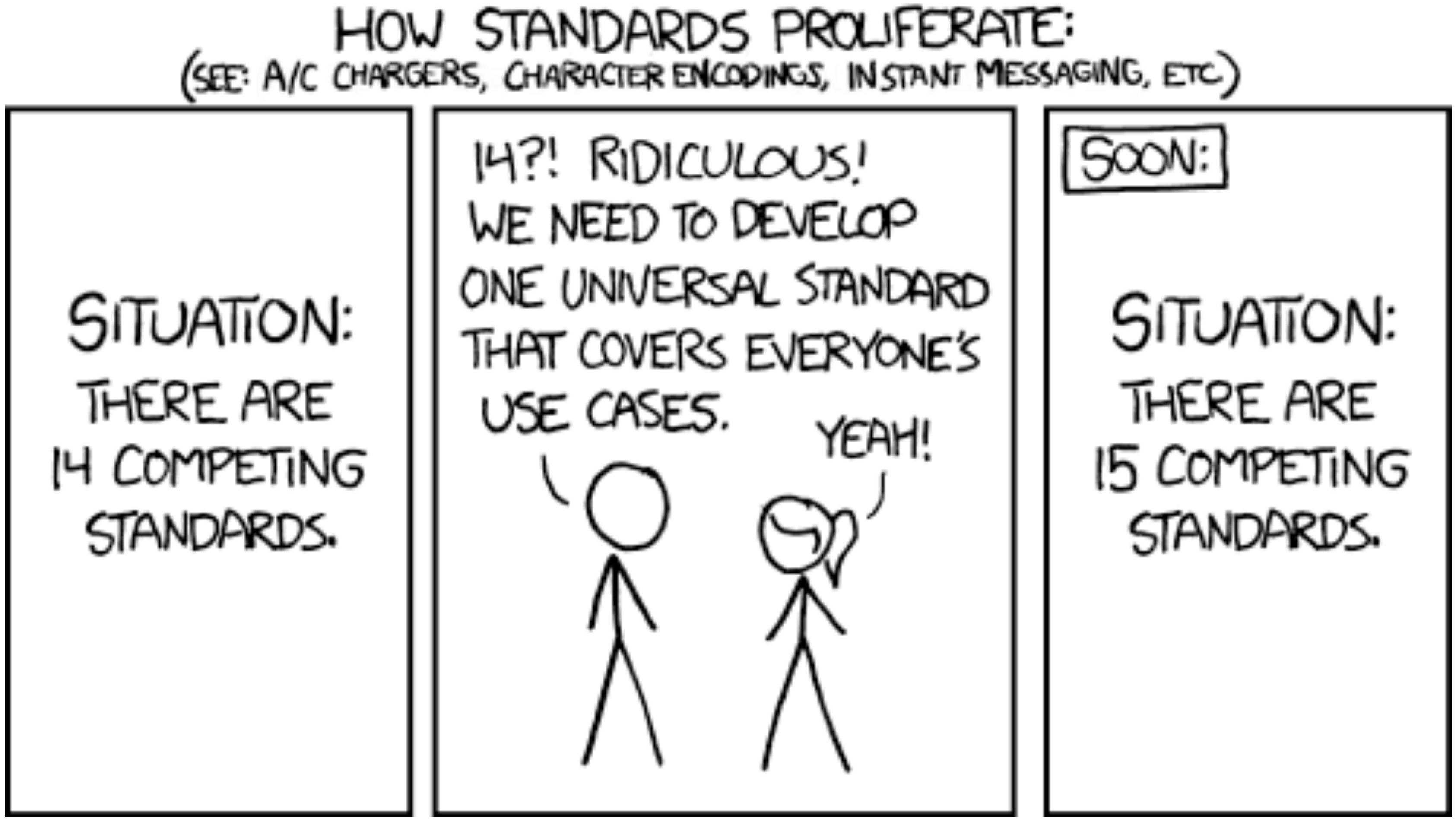
“We've spent the past five years delivering a variety of adaptive video formats—SmoothHD, HDNI, HLS and HDS—all of which are 80 percent the same but 100 percent incompatible.” - Will Law (Akamai) 2011

Microsoft Smooth Streaming

Adobe HTTP Dynamic Streaming (HDS)

Apple HTTP Live Streaming (HLS)

Let’s try to unify them and make the life of content providers and CDNs easier


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

Yet another standard: MPEG-DASH

Dynamic adaptive streaming over HTTP (DASH) is an ISO standard for the adaptive delivery of segmented content

- Blending existing formats into a new format

MPEG (moving pictures experts group)

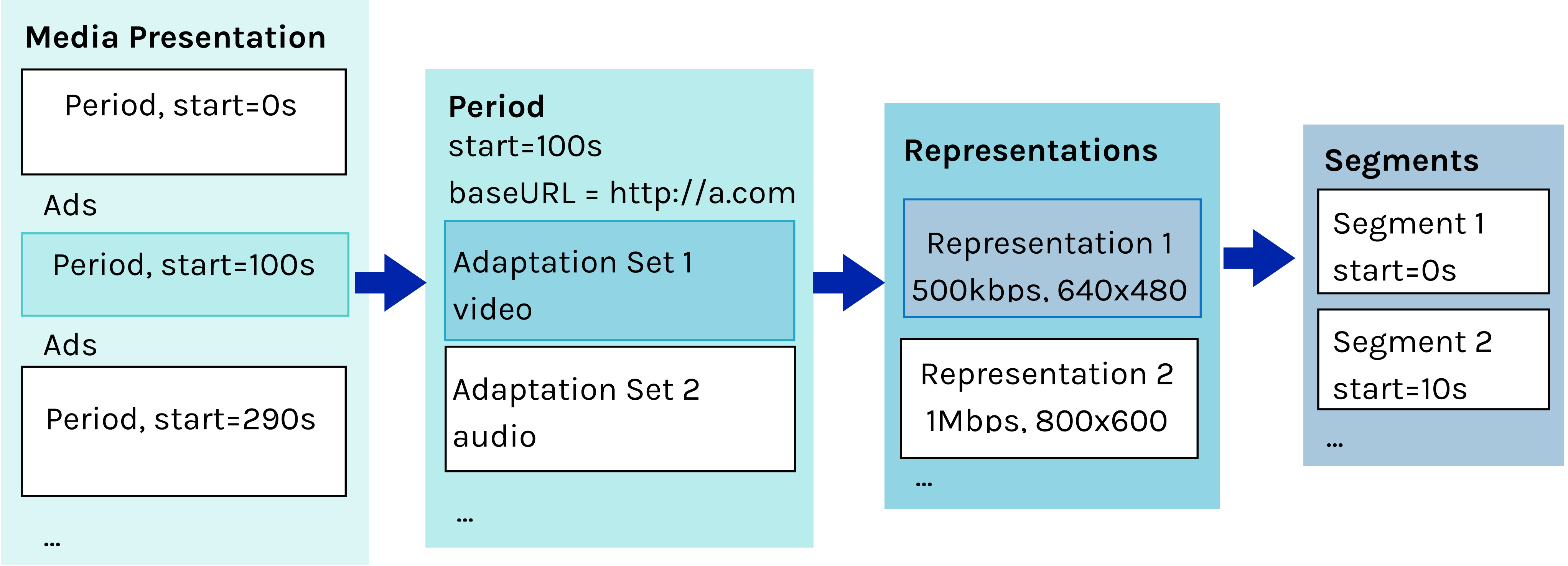
- Standardized MP3, MP4

Standardization work from 2010-2012

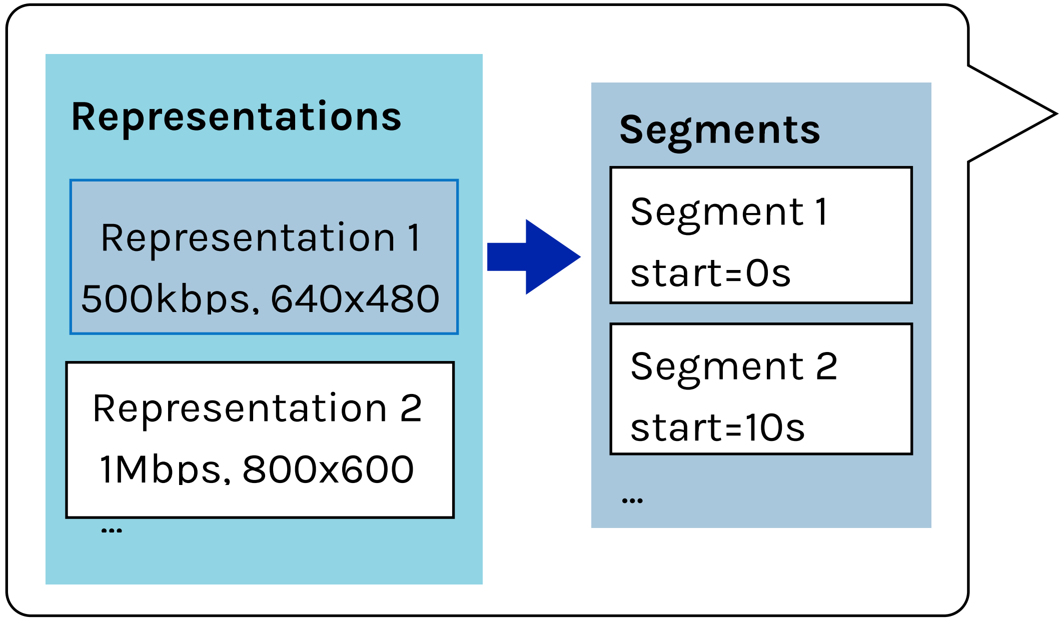
Note: DASH is not a protocol (implementation specific decisions are left out)

DASH: data model

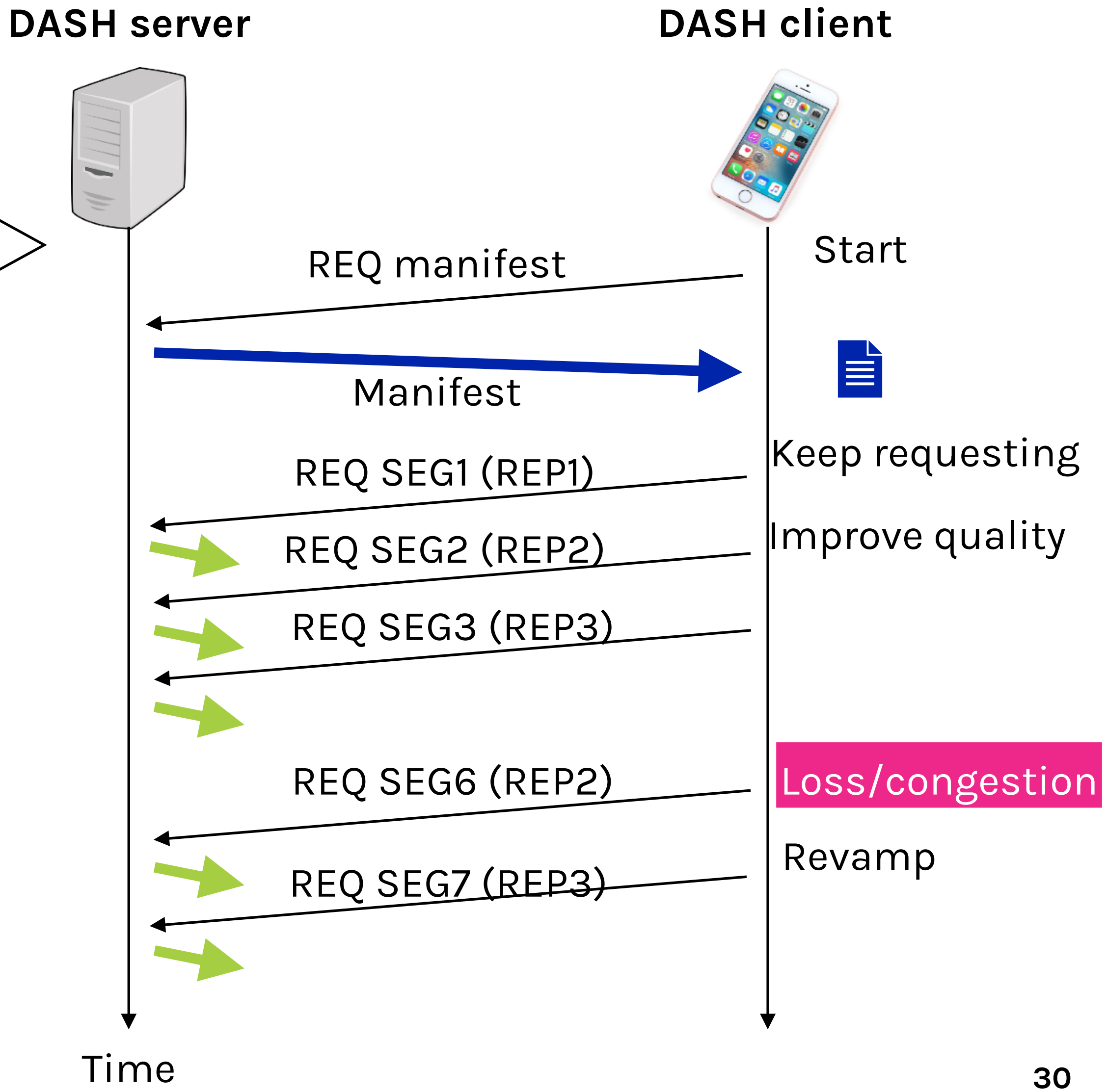
MDP (media presentation description) describes accessible segments and corresponding timing, ensuring interoperability



DASH workflow

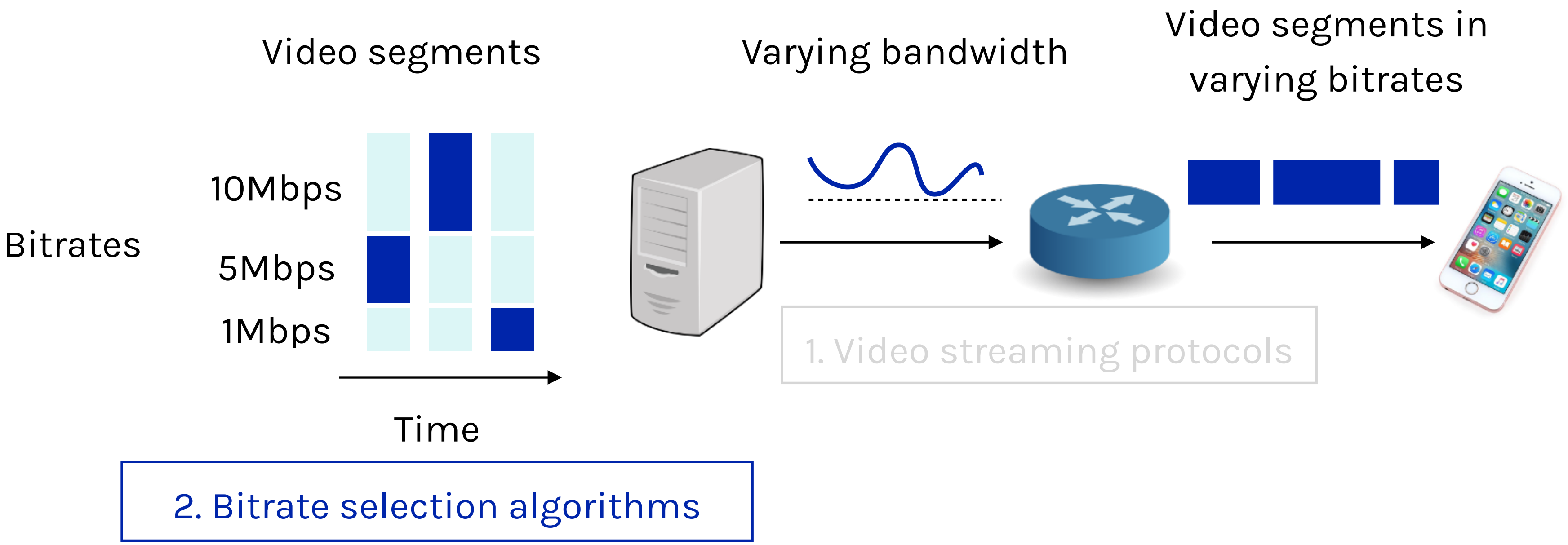


Video file is encoded using the MDP data model described with a **manifest** file



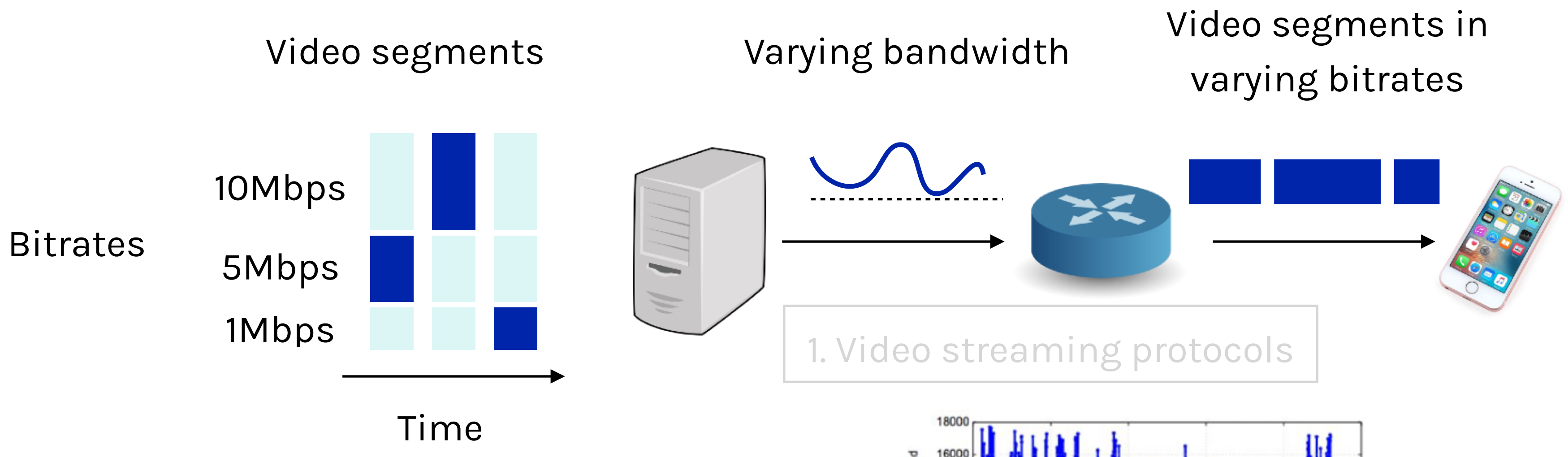
Bitrate Selection Algorithms

Bitrate selection in ABR



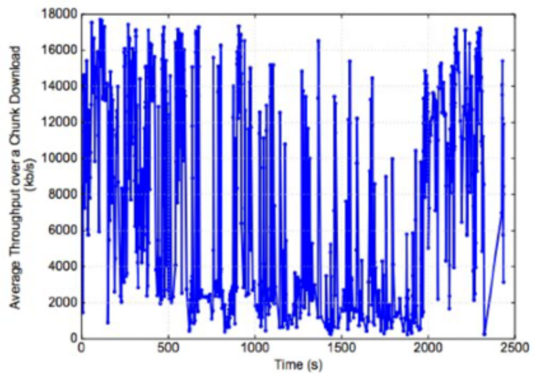
How would you design a bitrate selection algorithm?

Bitrate selection in ABR



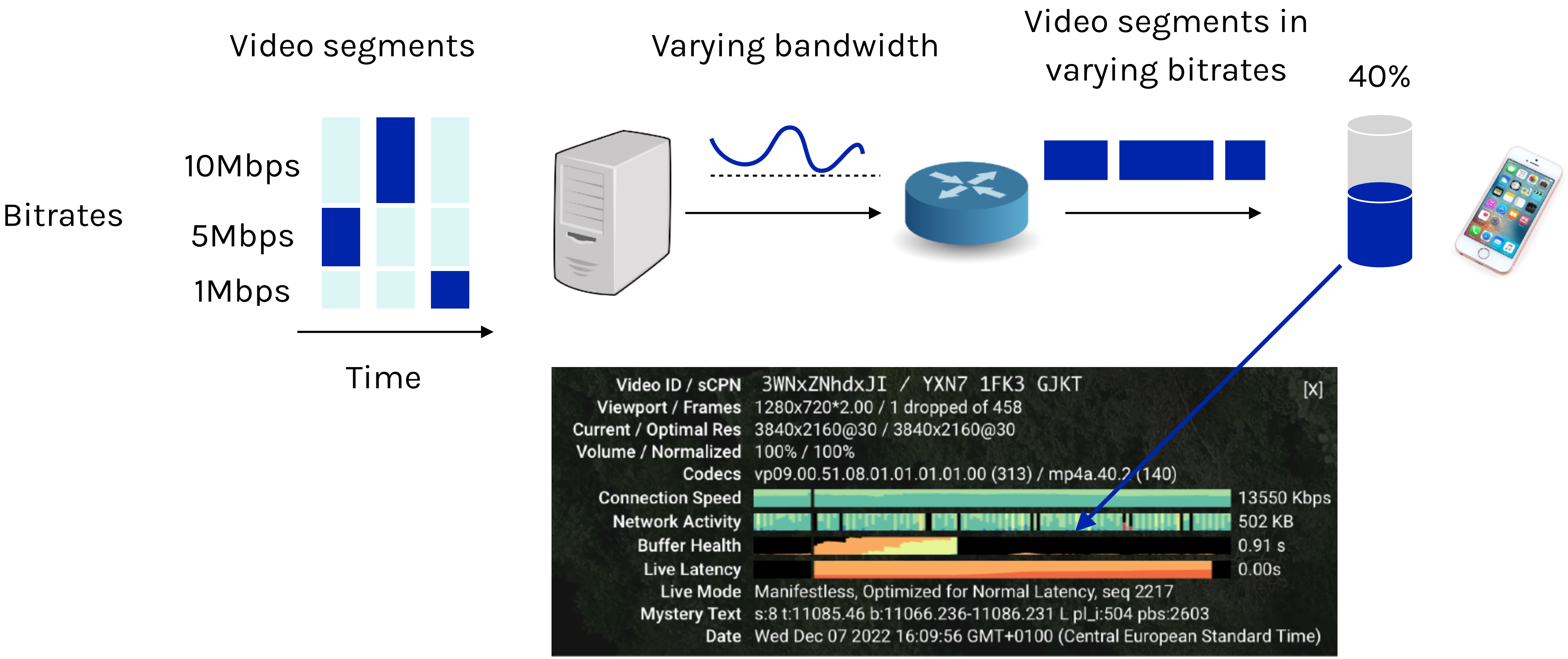
2. Bitrate selection algorithms

The most straightforward approach is to perform bandwidth estimation



Challenge: bandwidth variation can be very high!

Bitrate selection in ABR



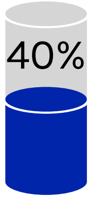
Make ABR decisions based on the buffer occupancy at the client

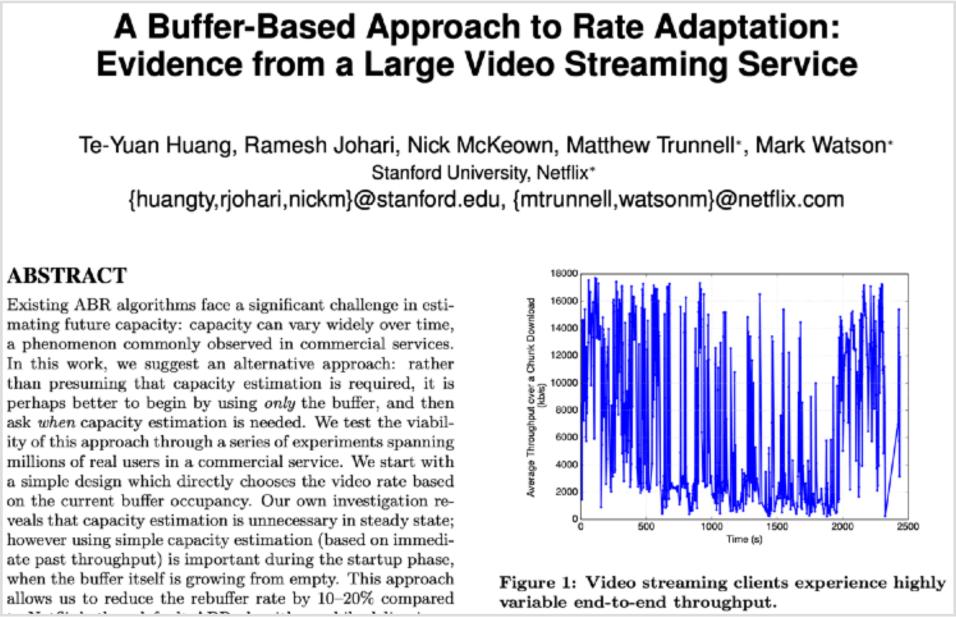
ABR algorithm: buffer-based

Main motivation

- Avoid bandwidth estimation
- Buffer occupancy contains implicit information about the bandwidth

BBA (buffer-based algorithm): pick the bitrate based on a function of buffer occupancy

bitrate $= f($  $)$



A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service

Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell*, Mark Watson*
Stanford University, Netflix*
{huangty,rjohari,nickm}@stanford.edu, {mtrunnell,watsonm}@netflix.com

ABSTRACT

Existing ABR algorithms face a significant challenge in estimating future capacity: capacity can vary widely over time, a phenomenon commonly observed in commercial services. In this work, we suggest an alternative approach: rather than presuming that capacity estimation is required, it is perhaps better to begin by using *only* the buffer, and then ask *when* capacity estimation is needed. We test the viability of this approach through a series of experiments spanning millions of real users in a commercial service. We start with a simple design which directly chooses the video rate based on the current buffer occupancy. Our own investigation reveals that capacity estimation is unnecessary in steady state; however using simple capacity estimation (based on immediate past throughput) is important during the startup phase, when the buffer itself is growing from empty. This approach allows us to reduce the rebuffer rate by 10–20% compared

Figure 1: Video streaming clients experience highly variable end-to-end throughput.

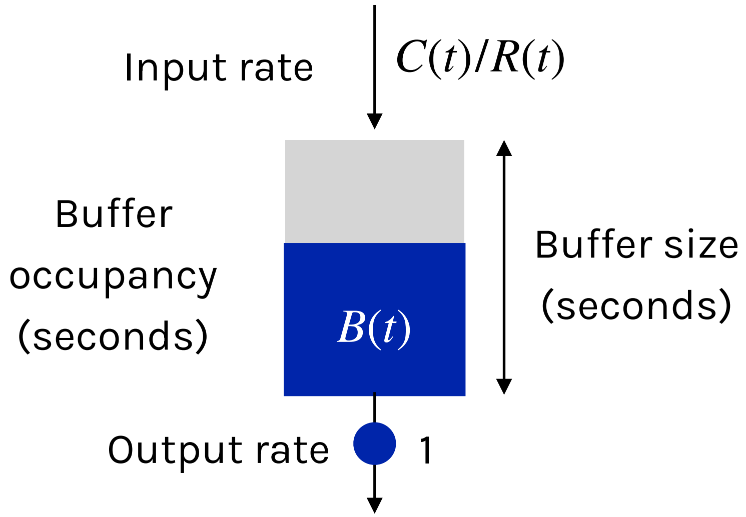
BBA: system model

$C(t)/R(t) > 1$: **buffer $B(t)$ grows**

- Arrival rate is higher than 1 second of video per second
- At a certain point, it is safe to increase $R(t)$ to improve the streaming quality

$C(t)/R(t) < 1$: **buffer $B(t)$ drains**

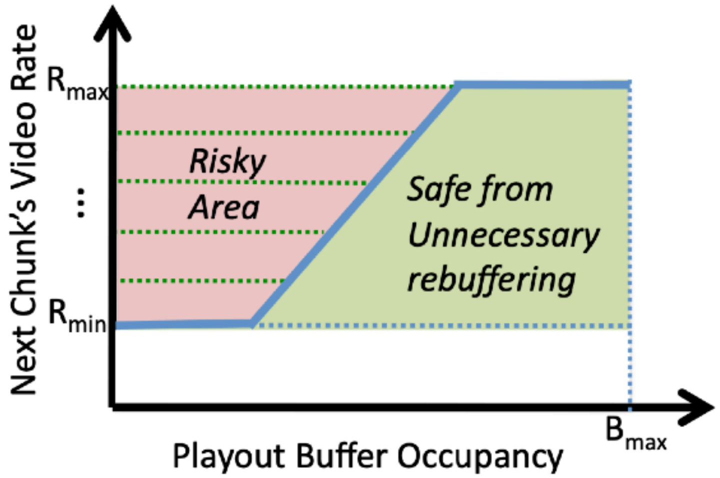
- Arrival rate is smaller than 1 second of video per second
- The chosen rate $R(t)$ is too high
- Buffer will be depleted and “rebuffering” happens



We use the unit of video seconds: representing how many seconds of video we can fetch/buffer

$\boxed{\text{Question: find a good function } R(t) = f(B(t))}$

BBA: theoretical analysis



Assumptions: infinitesimal segment size, continuous bit rate, videos are CBR coded, videos are infinitely long

Goal 1: no unnecessary rebuffering

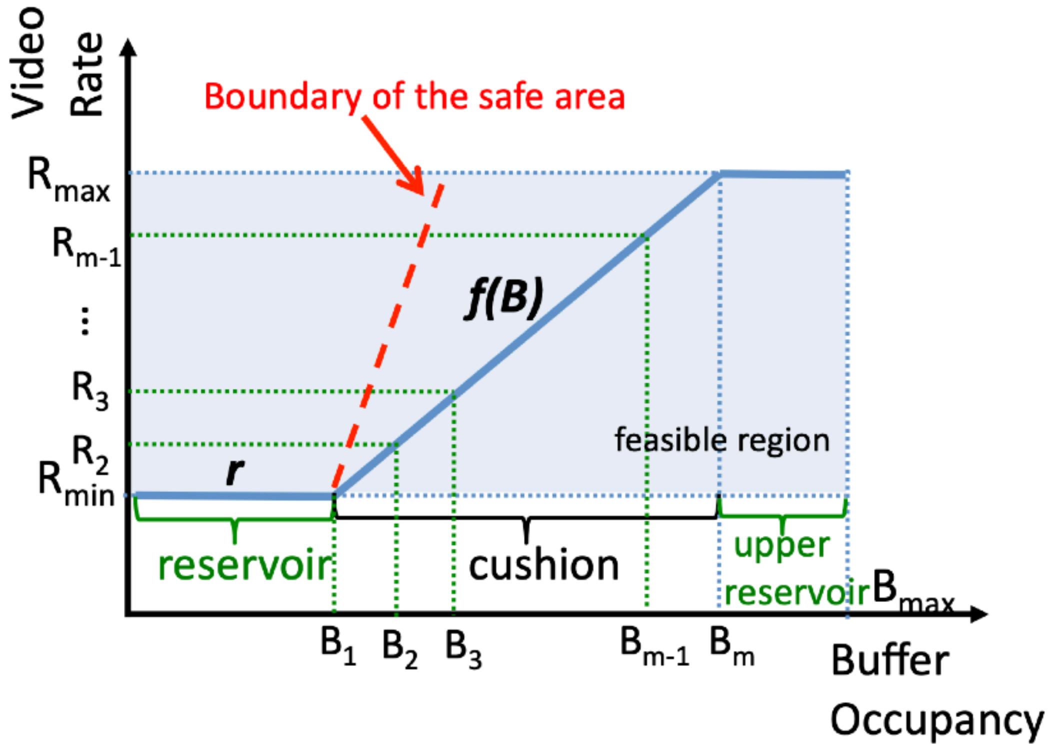
- As long as $C(t) > R_{min}$ for all t and we adapt $f(B) \rightarrow R_{min}$ as $B \rightarrow 0$, we will never unnecessarily rebuffer because the buffer will start to grow before it runs dry

Goal 2: average video rate maximization

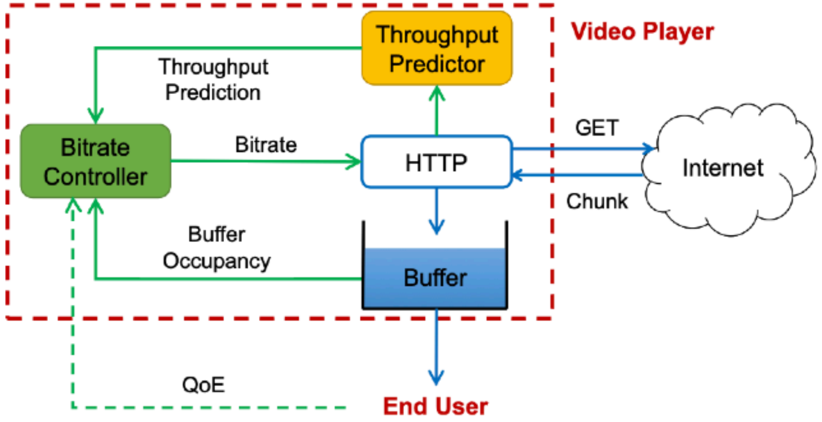
- As long as $f(B)$ is increasing and eventually reaches R_{max}, the average video rate will match the average capacity when $R_{min} < C(t) < R_{max}$ for all $t > 0$

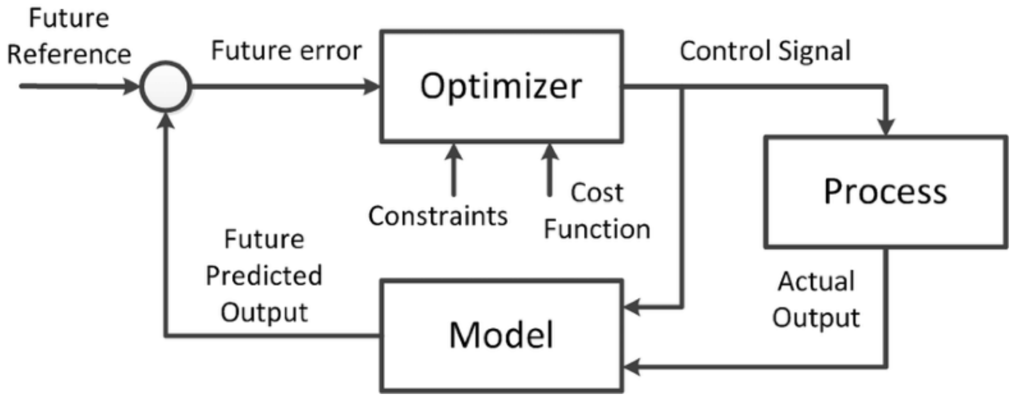
BBA in practice

Assumptions do not always hold in practice, we need to be more conservative



ABR algorithm: control theory based

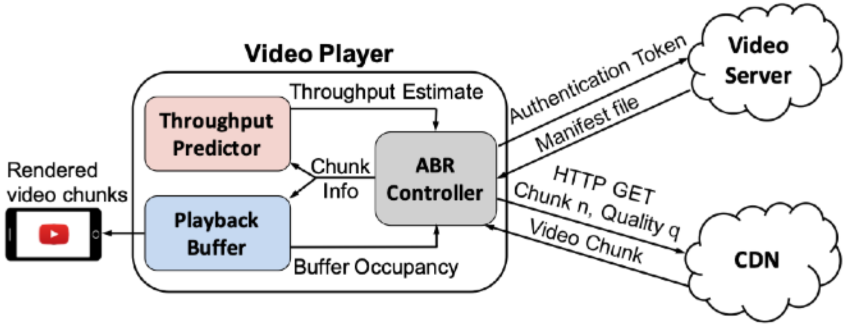


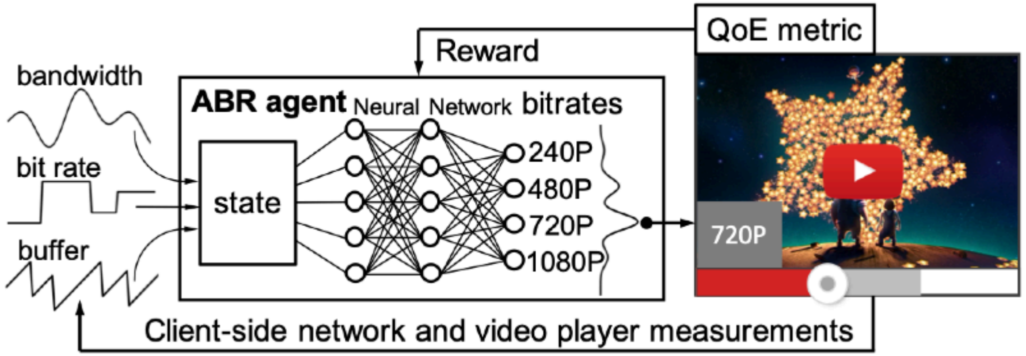


Model the ABR control problem as Markov processes and apply control theory

https://dl.acm.org/doi/pdf/10.1145/2785956.2787486

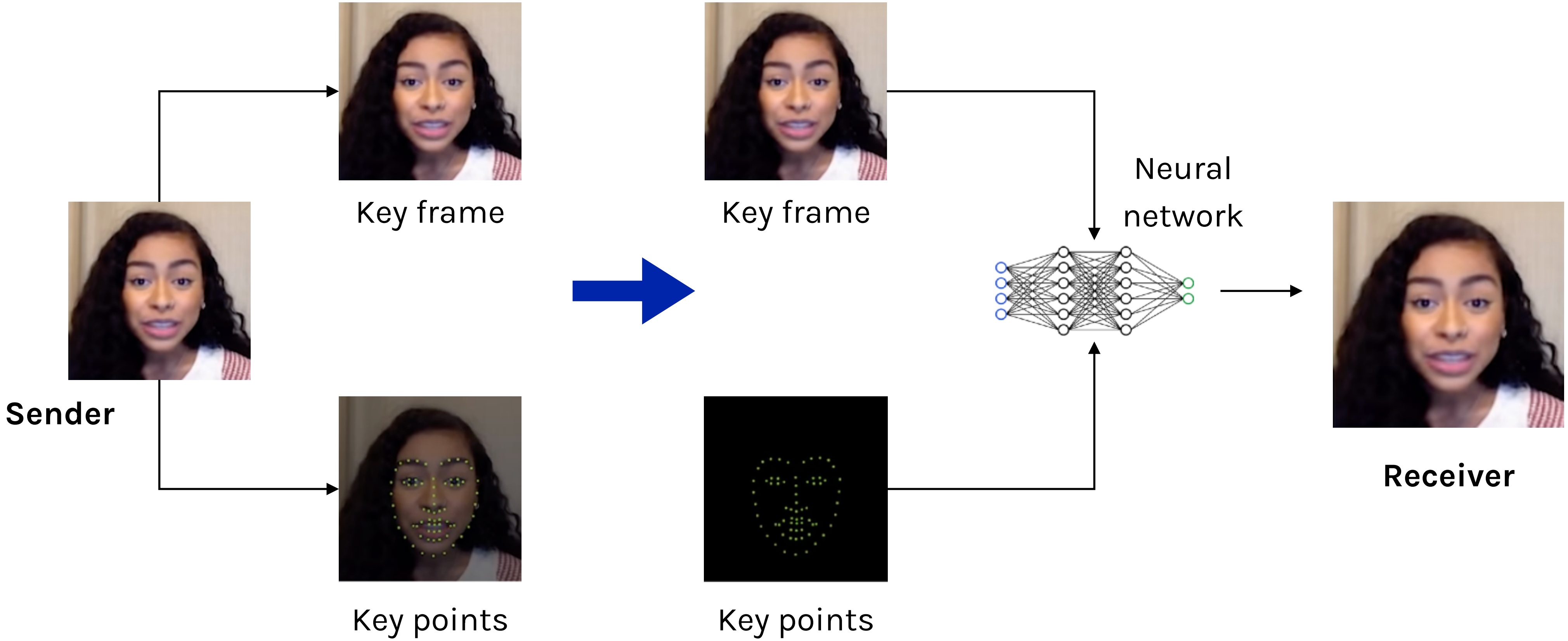
ABR algorithm: deep reinforcement learning based





Model the ABR control problem as a **Markov Decision Process** and apply deep reinforcement learning

Replacing video codecs with machine learning



https://www.youtube.com/watch?v=NqmMnjJ6GEg

Netflix Video Serving

Netflix video serving

Workloads

- Serve only static media files
- Pre-encoded for all codecs/bitrates

Video serving stack: FreeBSD-current, NGINX web server (HTTP)

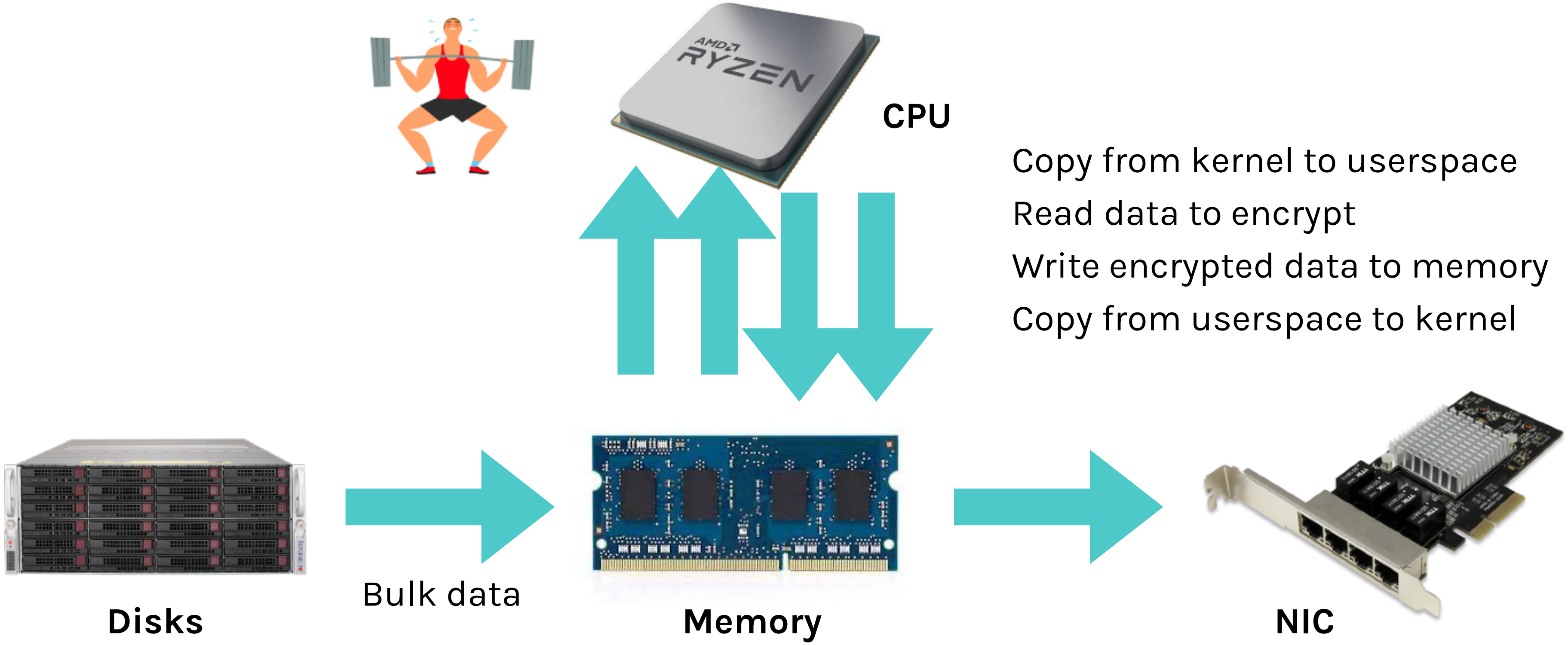
- Video served via asynchronous **sendfile()** and encrypted using kTLS (offloaded to NICs)
- Since 2020, 200Gbps of TLS encrypted traffic from a server, aiming for 400+ Gbps now

Sendfile() directs the kernel to send data from a file descriptor to a TCP socket

- Eliminates the need to copy data into or out of the kernel

Based on the content of a recent talk from Drew Gallatin (Netflix)

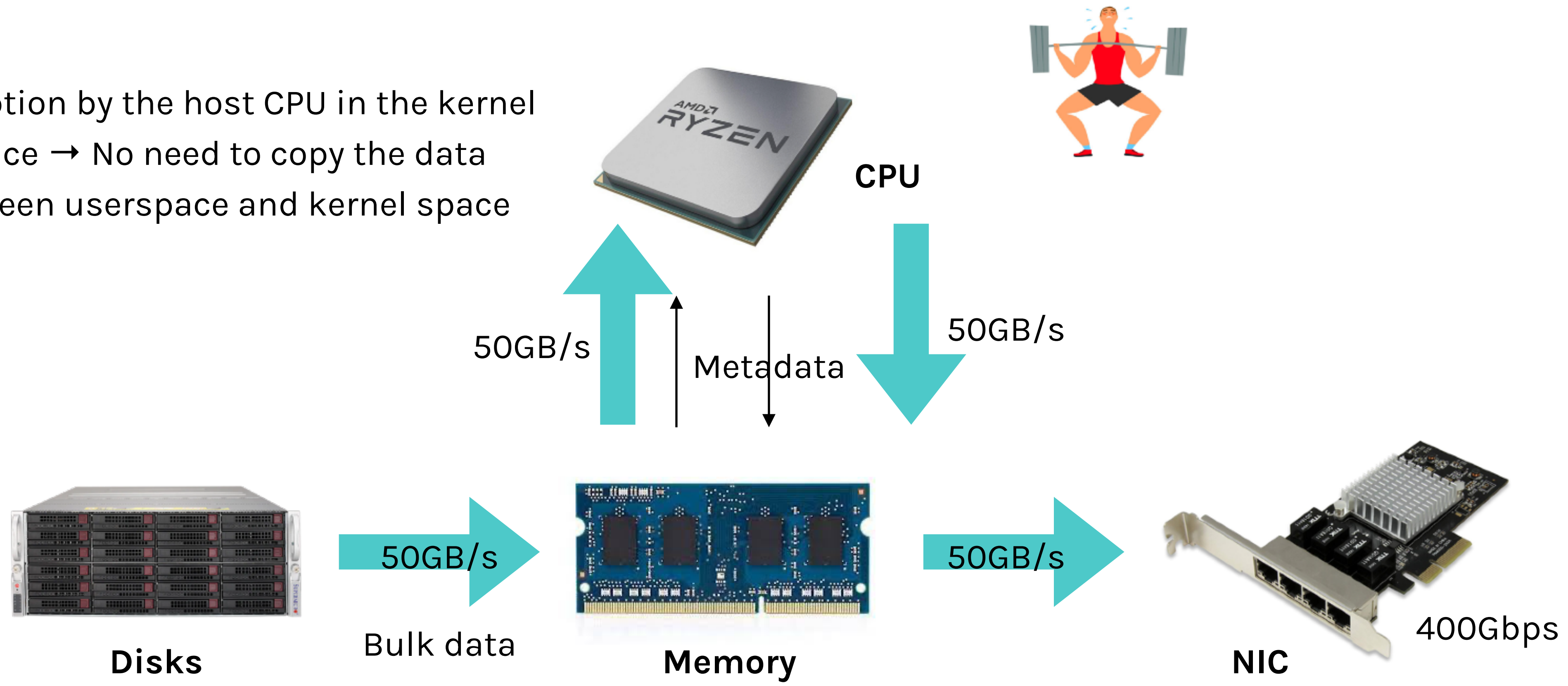
Netflix video serving data flow



Memory bandwidth is a bottleneck

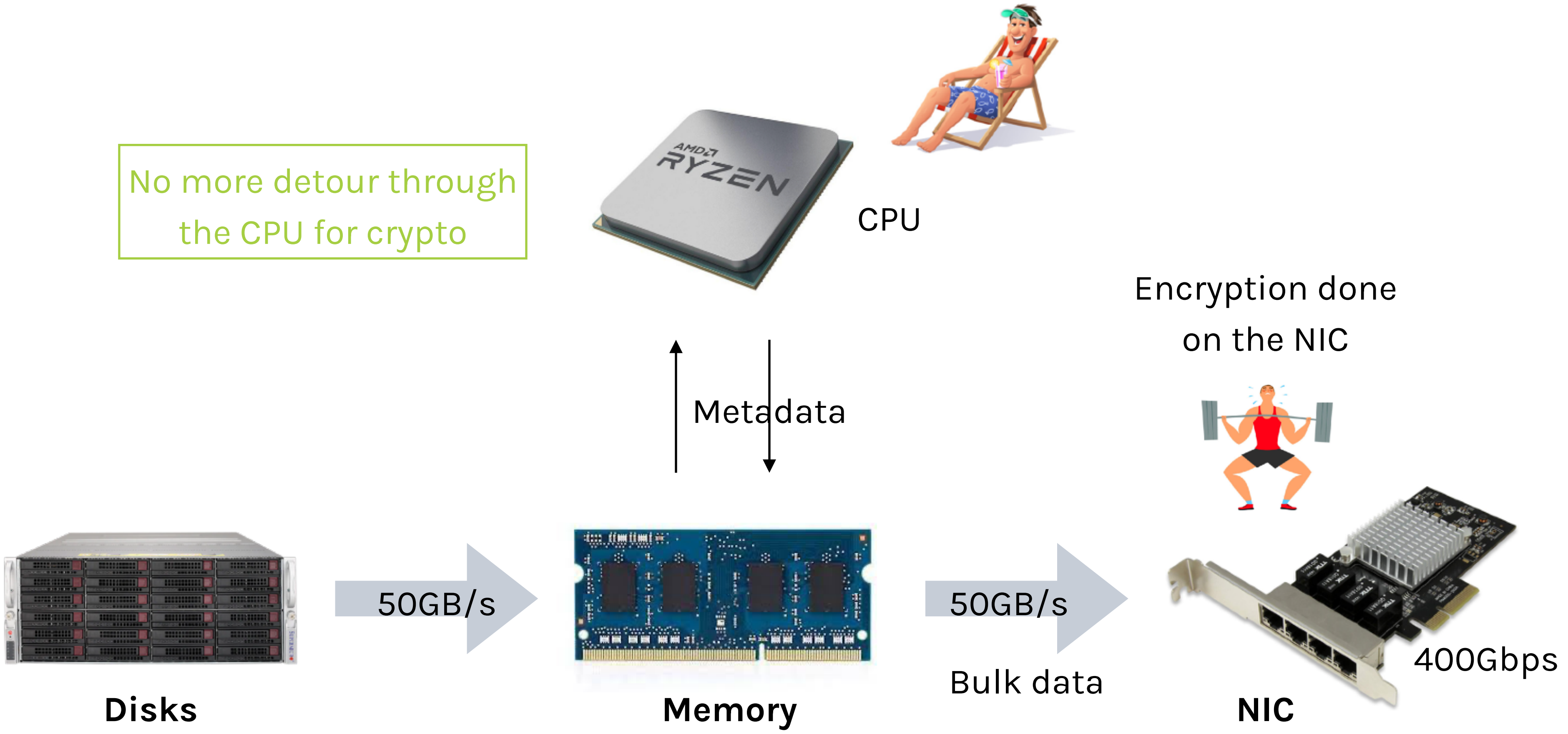
Netflix video serving data flow

Encryption by the host CPU in the kernel space → No need to copy the data between userspace and kernel space



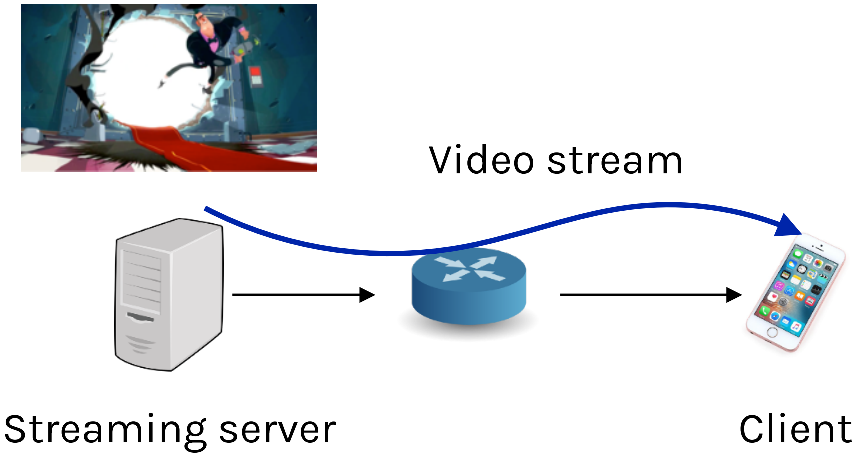
achieve 400Gbps → require 200GB/s memory bandwidth → Challenging

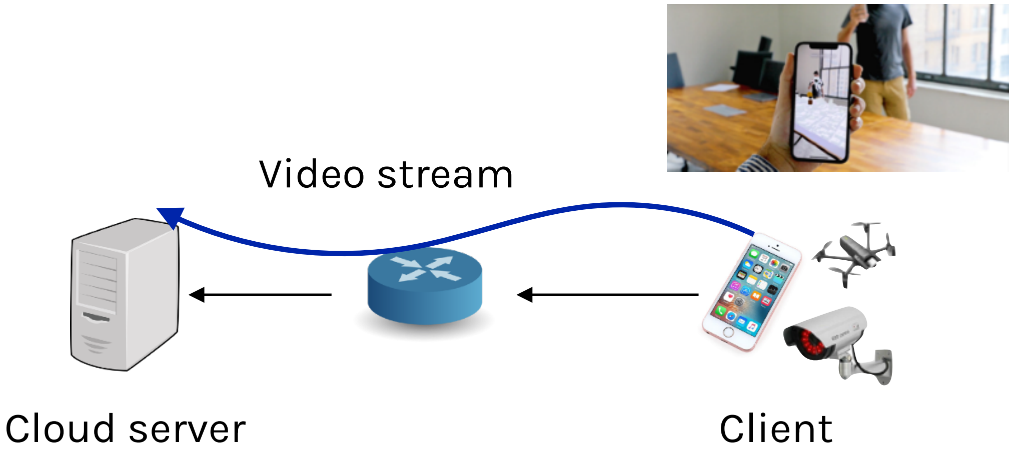
Netflix video serving data flow



Video Analytics

Video streaming vs. video stream analytics





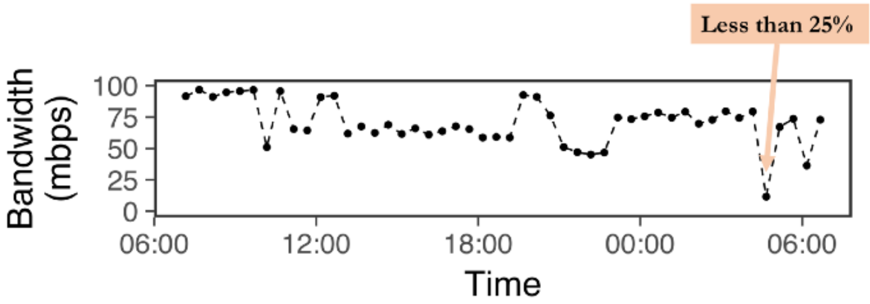
Video streaming

Video stream analytics

Challenges in video stream analytics

Large volume of traffic needs to be sent across the wide area network (WAN)

WAN has scarce, expensive, and variable bandwidth



Applications have quality of service requirements which are complex to optimize

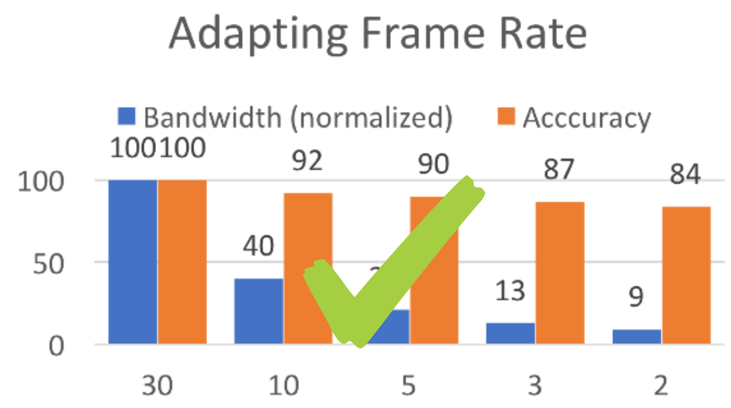
- Unlike video streaming where quality of experience is relatively well-defined
- Video analytics rely on **deep learning** models and the analytics accuracy has a nonlinear relationship with the quality metrics (resolution, frame rate, latency)

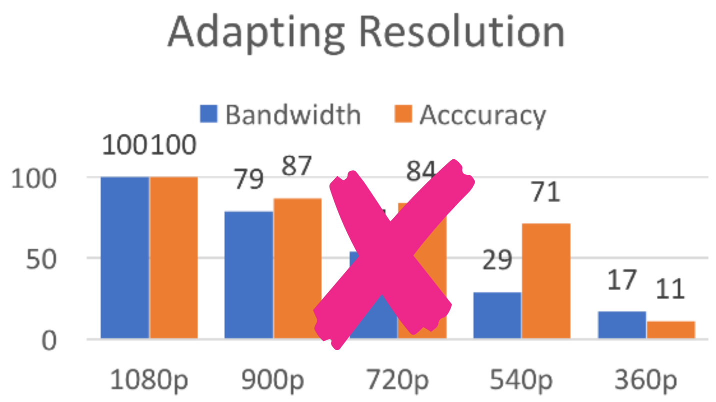
Application-specific optimization

Scenario 1: a monitoring application that counts pedestrians on a busy street

t=0s, small target in far-field views

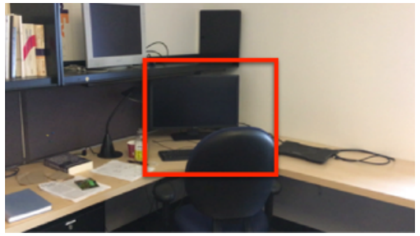
t=1s, small difference



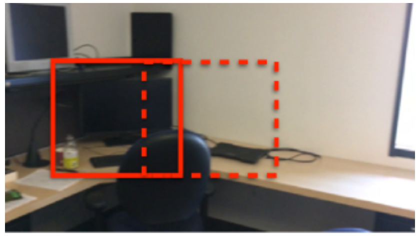


Application-specific optimization

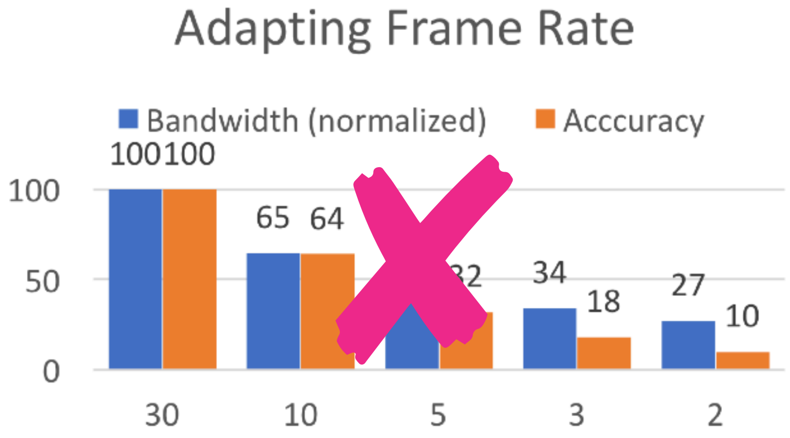
Scenario 2: an AR application that detects objects on a mobile phone

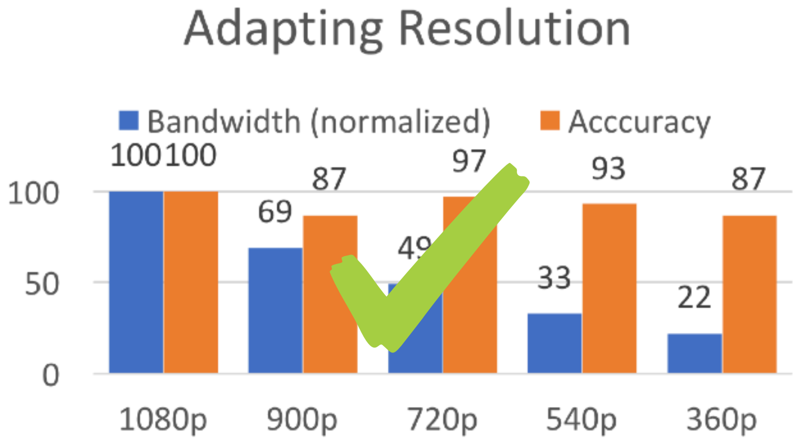


t=0s, nearby and large target



t=1s, large difference due to camera movement

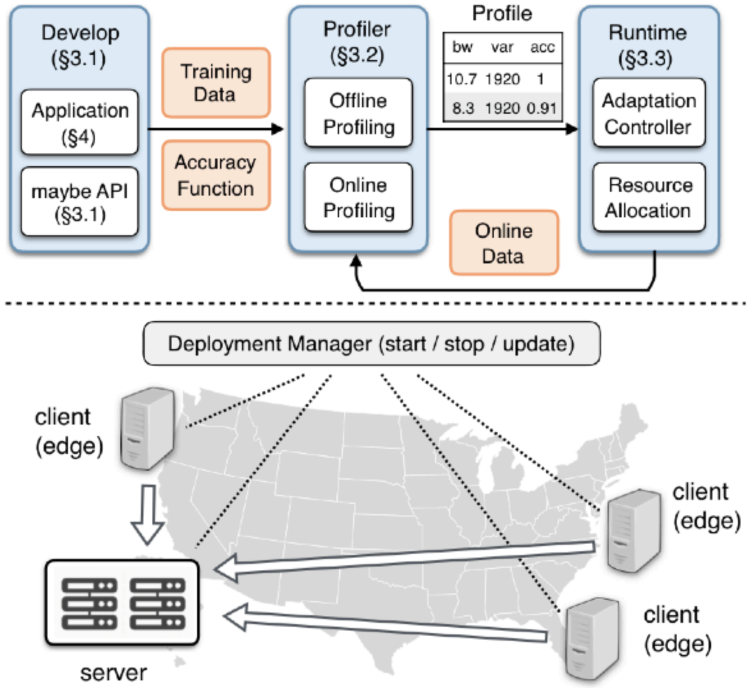




A general framework: AWStream

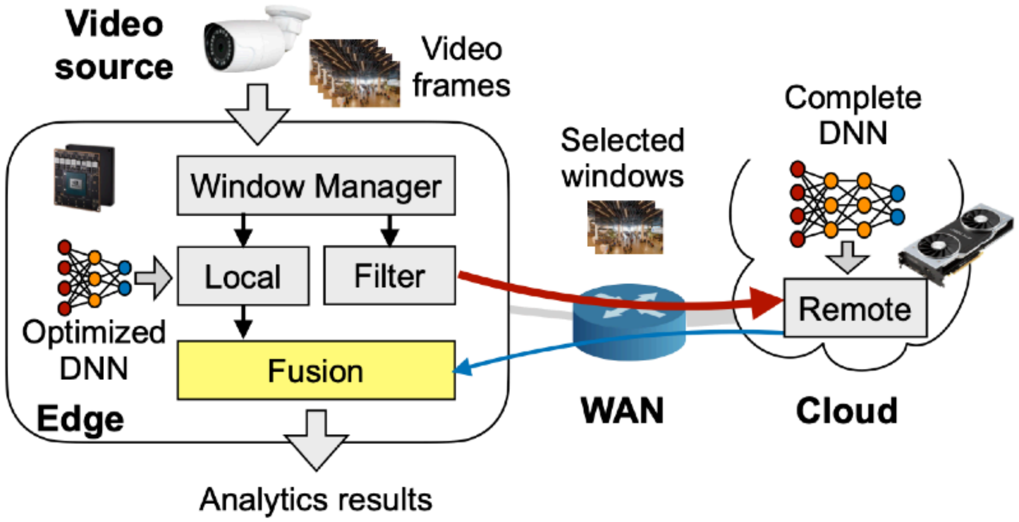
Systematic and quantitative adaptation

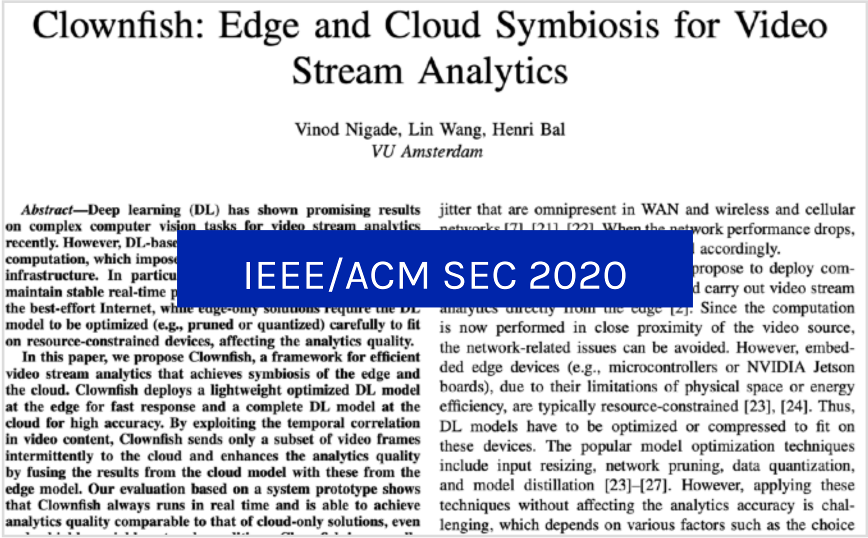
- New **programming abstractions** to express adaptation
- **Automatic** data-driven **profiling**
- **Runtime adaptation** balancing the different goals



Clownfish: real-time video stream analytics

Combine a local fast processing and a remote accurate processing





Clownfish: Edge and Cloud Symbiosis for Video Stream Analytics

Vinod Nigade, Lin Wang, Henri Bal
VU Amsterdam

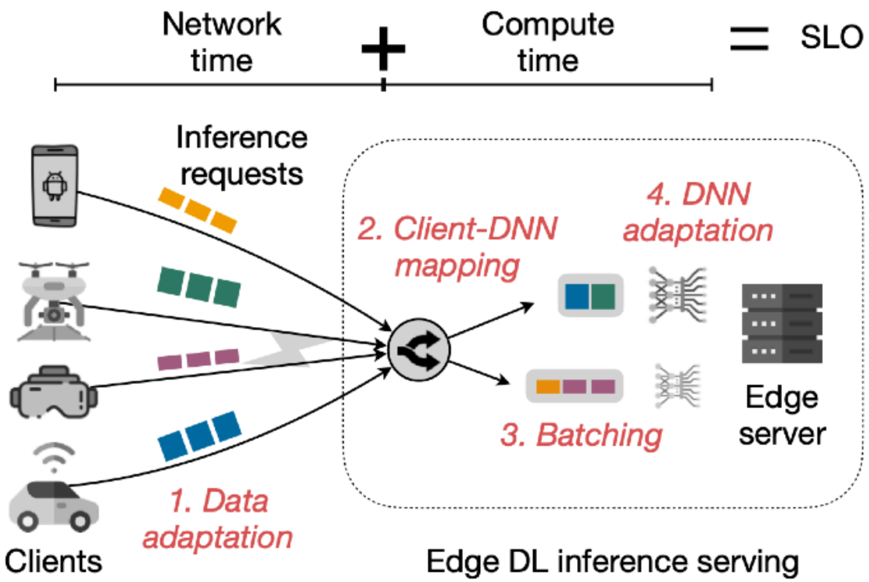
IEEE/ACM SEC 2020

***Abstract*—Deep learning (DL) has shown promising results on complex computer vision tasks for video stream analytics recently. However, DL-base computation, which impose infrastructure. In particu maintain stable real-time p the best-effort Internet, model to be optimized (e.g., pruned or quantized) carefully to fit on resource-constrained devices, affecting the analytics quality.**

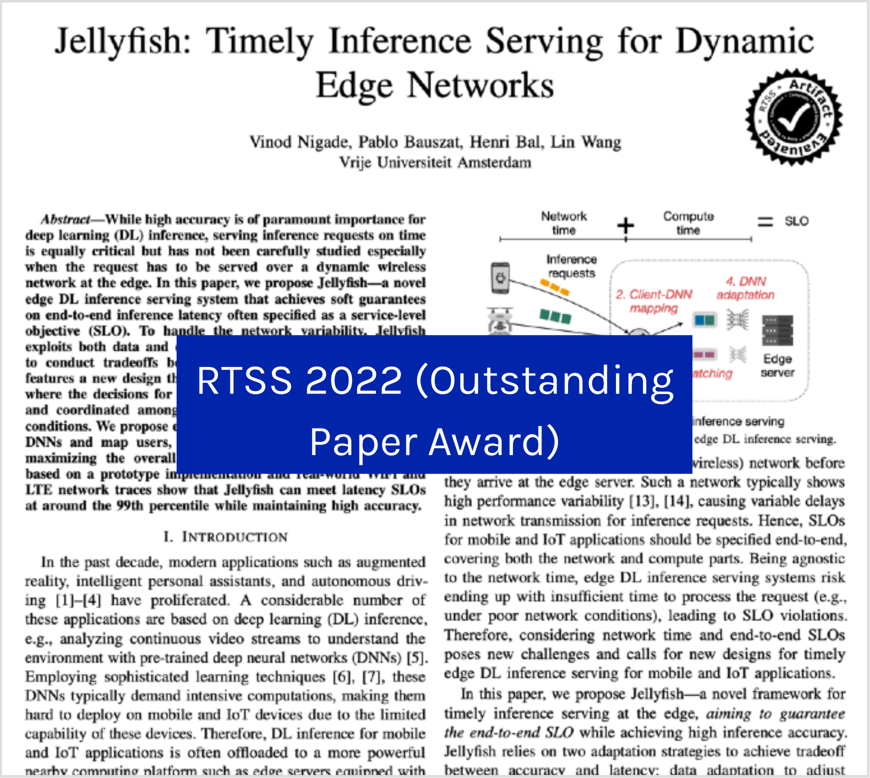
In this paper, we propose Clownfish, a framework for efficient video stream analytics that achieves symbiosis of the edge and the cloud. Clownfish deploys a lightweight optimized DL model at the edge for fast response and a complete DL model at the cloud for high accuracy. By exploiting the temporal correlation in video content, Clownfish sends only a subset of video frames intermittently to the cloud and enhances the analytics quality by fusing the results from the cloud model with these from the edge model. Our evaluation based on a system prototype shows that Clownfish always runs in real time and is able to achieve analytics quality comparable to that of cloud-only solutions, even

jitter that are omnipresent in WAN and wireless and cellular networks [7], [21], [22]. When the network performance drops, accordingly. propose to deploy com- d carry out video stream analytics directly from the edge [2]. Since the computation is now performed in close proximity of the video source, the network-related issues can be avoided. However, embedded edge devices (e.g., microcontrollers or NVIDIA Jetson boards), due to their limitations of physical space or energy efficiency, are typically resource-constrained [23], [24]. Thus, DL models have to be optimized or compressed to fit on these devices. The popular model optimization techniques include input resizing, network pruning, data quantization, and model distillation [23]–[27]. However, applying these techniques without affecting the analytics accuracy is challenging, which depends on various factors such as the choice

Jellyfish: real-time video stream analytics



End-to-end real-time guarantee



Jellyfish: Timely Inference Serving for Dynamic Edge Networks

Vinod Nigade, Pablo Bauszat, Henri Bal, Lin Wang
Vrije Universiteit Amsterdam

Abstract—While high accuracy is of paramount importance for deep learning (DL) inference, serving inference requests on time is equally critical but has not been carefully studied especially when the request has to be served over a dynamic wireless network at the edge. In this paper, we propose Jellyfish—a novel edge DL inference serving system that achieves soft guarantees on end-to-end inference latency often specified as a service-level objective (SLO). To handle the network variability, Jellyfish exploits both data and
to conduct tradeoffs b
features a new design th
where the decisions for
and coordinated among
conditions. We propose
DNNs and map users,
maximizing the overall
based on a prototype implementation and real-world WiFi and LTE network traces show that Jellyfish can meet latency SLOs at around the 99th percentile while maintaining high accuracy.

I. Introduction

In the past decade, modern applications such as augmented reality, intelligent personal assistants, and autonomous driving [1]–[4] have proliferated. A considerable number of these applications are based on deep learning (DL) inference, e.g., analyzing continuous video streams to understand the environment with pre-trained deep neural networks (DNNs) [5]. Employing sophisticated learning techniques [6], [7], these DNNs typically demand intensive computations, making them hard to deploy on mobile and IoT devices due to the limited capability of these devices. Therefore, DL inference for mobile and IoT applications is often offloaded to a more powerful nearby computing platform such as edge servers equipped with

inference serving
edge DL inference serving.

wireless) network before they arrive at the edge server. Such a network typically shows high performance variability [13], [14], causing variable delays in network transmission for inference requests. Hence, SLOs for mobile and IoT applications should be specified end-to-end, covering both the network and compute parts. Being agnostic to the network time, edge DL inference serving systems risk ending up with insufficient time to process the request (e.g., under poor network conditions), leading to SLO violations. Therefore, considering network time and end-to-end SLOs poses new challenges and calls for new designs for timely edge DL inference serving for mobile and IoT applications.

In this paper, we propose Jellyfish—a novel framework for timely inference serving at the edge, *aiming to guarantee the end-to-end SLO* while achieving high inference accuracy. Jellyfish relies on two adaptation strategies to achieve tradeoff between accuracy and latency: data adaptation to adjust

RTSS 2022 (Outstanding Paper Award)

Summary

Video streaming

- Video compression
- Bitrate, VBR, CBR
- ABR

Video streaming protocols

- RTP
- HTTP-based
- DASH

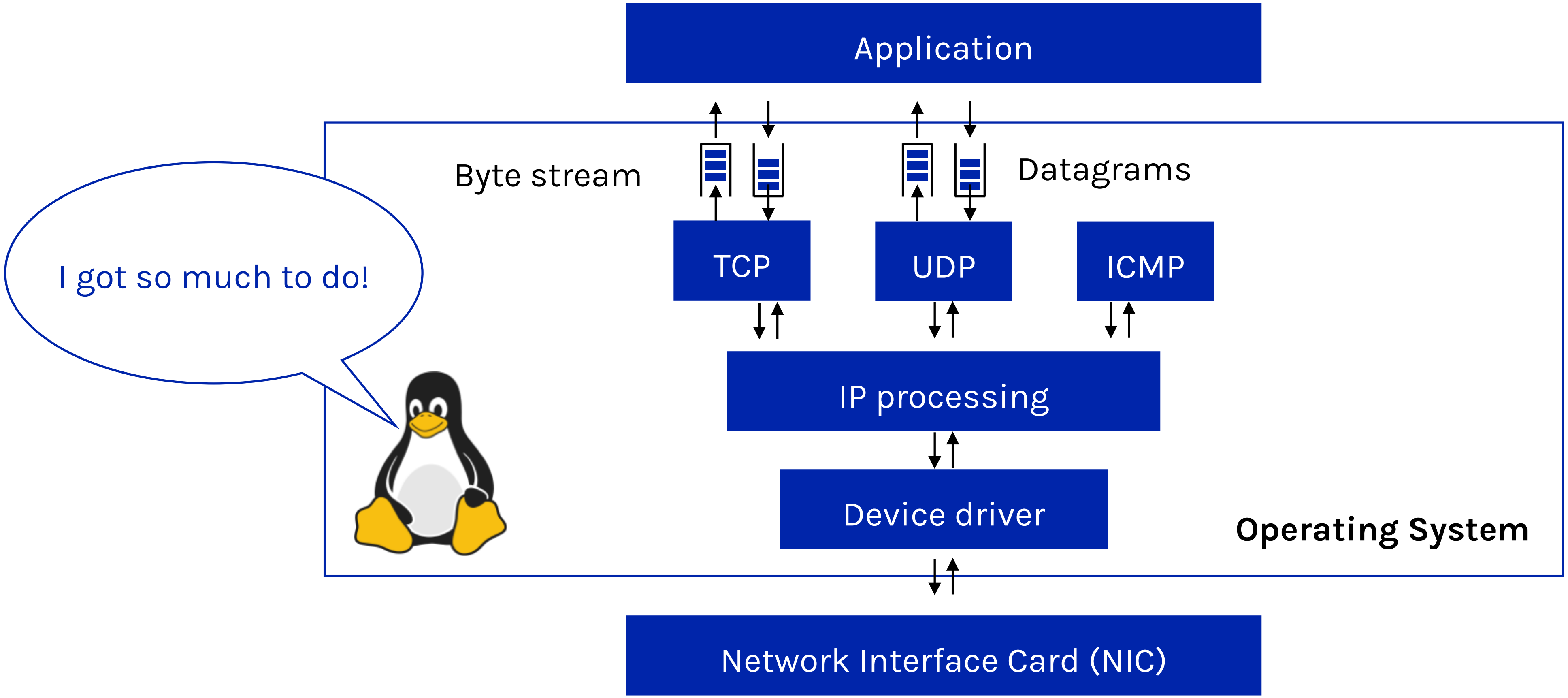
Bitrate selection algorithms

- Rate-based
- Buffer-based: BBA
- Others: control theory, learning

Advanced topics

- Netflix video serving
- Video analytics

Next time: Linux networking stack



Further reading material

Andrew S. Tanenbaum, David J. Wetherall. Computer Networks (5th edition).

- Section 7.4: Streaming Audio and Video

Junchen Jiang, Vyas Sekar, Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. ACM CoNEXT, 2012.

Marios Kanakis, Ramin Khalili, Lin Wang. Machine Learning for Computer Systems and Networking: A Survey. ACM Computing Surveys, vol. 44(4), pp. 1-36, 2022.

- Section 9: Adaptive Video Streaming