Communication Networks

Prof. Laurent Vanbever

Communication Networks

Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
April 29 2019

Materials inspired from Scott Shenker, Jennifer Rexford, and Ankit Singla

Two weeks ago on
Communication Networks

Congestion Control

DNS

google.ch ⟷ 172.217.16.131

Congestion Control

DNS

Congestion control aims at solving three problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps…
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth "fairly" among flows, without overloading the network

Congestion control differs from flow control

both are provided by TCP though

Flow control	prevents one fast sender from overloading a slow receiver
Congestion control	prevents a set of senders from overloading the network

The sender adapts its sending rate based on these two windows

Receiving Window RWND	How many bytes can be sent without overflowing the receiver buffer? based on the receiver input
Congestion Window CWND	How many bytes can be sent without overflowing the routers? based on network conditions
Sender Window	minimum(CWND, RWND)

The 2 key mechanisms of Congestion Control

detecting congestion

reacting to congestion

The 2 key mechanisms of Congestion Control

detecting congestion

reacting to congestion

Detecting losses can be done using ACKs or timeouts, the two signal differ in their degree of severity

duplicated ACKs	mild congestion signal packets are still making it
timeout	severe congestion signal multiple consequent losses

The 2 key mechanisms of Congestion Control

detecting congestion

reacting to congestion

TCP approach is to gently increase when not congested and to rapidly decrease when congested

question: What increase/decrease function should we use?

it depends on the problem we are solving...

Congestion control aims at solving three problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps...
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth "fairly" among flows, without overloading the network

#1 bandwidth estimation: How to adjust the bandwidth of a single flow to the bottleneck bandwidth?

could be 1 Mbps or 1 Gbps...

Initially, you want to quickly get a first-order estimate of the available bandwidth

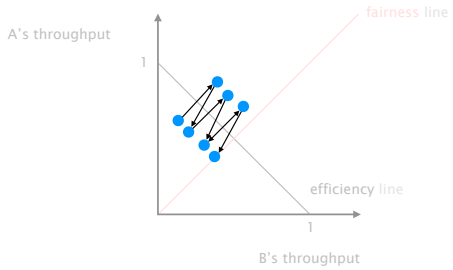
Intuition	Start slow but rapidly increase until a packet drop occurs	
Increase policy	cwnd = 1	initially
	cwnd += 1	upon receipt of an ACK

#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?

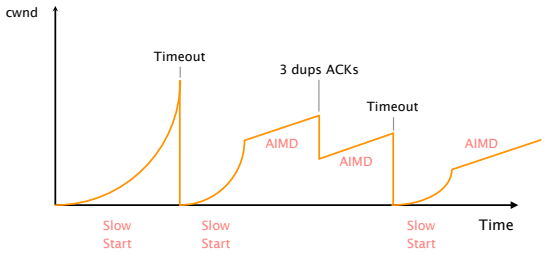
	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

#3	fairness	How to share bandwidth "fairly" among flows, without overloading the network

AIMD converge to fairness and efficiency,
it then fluctuates around the optimum (in a stable way)



Congestion control makes TCP throughput
look like a "sawtooth"



Congestion Control

DNS

google.ch ⟷ 172.217.16.131

The DNS system is a distributed database
which enables to resolve a name into an IP address

name	DNS ⟷	IP address
www.ethz.ch		129.132.19.216

To scale,
DNS adopt three intertwined hierarchies

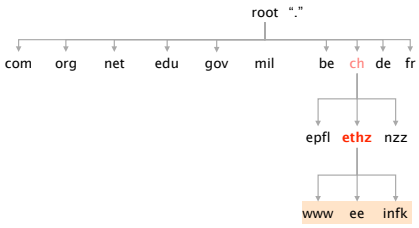
naming structure	addresses are hierarchical www.ee.ethz.ch
management	hierarchy of authority over names
infrastructure	hierarchy of DNS servers

naming structure

addresses are hierarchical

www.ee.ethz.ch

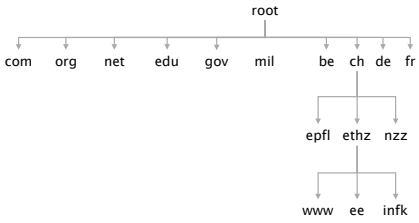
A name, *e.g.* ee.ethz.ch, represents
a leaf-to-root path in the hierarchy



management

hierarchy of authority
over names

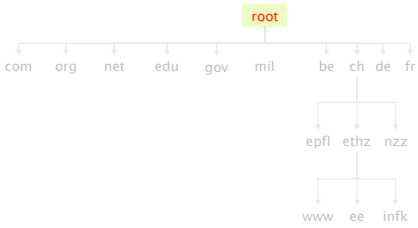
The DNS system is
hierarchically administered



infrastructure

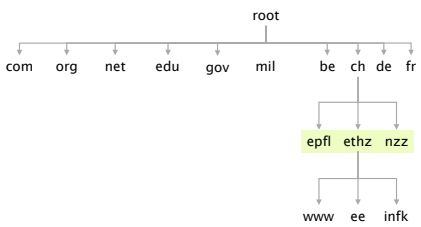
hierarchy of DNS servers

13 root servers (managed professionally)
serve as root (*)



(*) see http://www.root-servers.org/

The bottom (and bulk) of the hierarchy is
managed by Internet Service Provider or locally

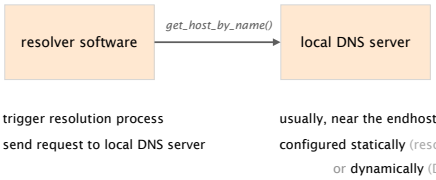


Every server knows the address of the root servers (*)
required for bootstrapping the systems

(*) see https://www.internic.net/domain/named.root

Each server knows the address of all children

Using DNS relies on two components

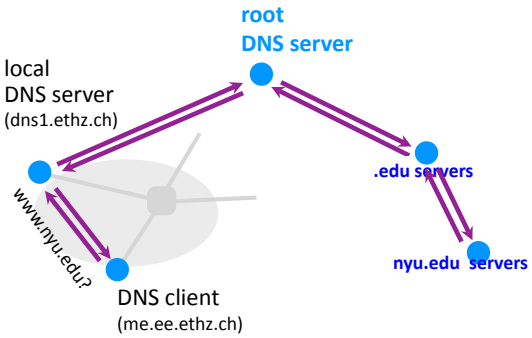


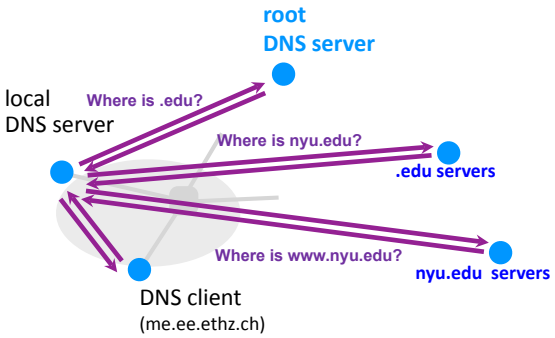
trigger resolution process
send request to local DNS server

usually, near the endhosts
configured statically (resolv.conf)
or dynamically (DHCP)

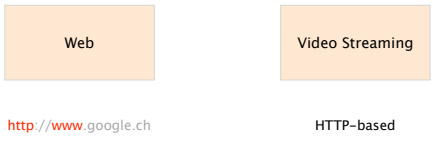
Records	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

DNS resolution can either be
recursive or iterative





This week on
Communication Networks



http://www.google.ch

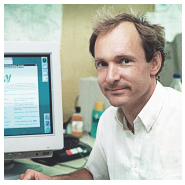
HTTP-based

Web

Video Streaming

http://www.google.ch

The Web as we know it was founded in ~1990,
by Tim Berners-Lee, physicist at CERN



Tim Berners-Lee Photo: CERN

His goal:
provide distributed access to data

The World Wide Web (WWW):
a distributed database of "pages"
linked together via the
Hypertext Transport Protocol (HTTP)

The Web was and still is so successful as
it enables everyone to self-publish content

Self-publishing on the Web is easy, independent & free
and accessible, to everyone

People weren't looking for technical perfection
little interest in collaborative or idealistic endeavor

People essentially want to make their mark
and find something neat…

The WWW is made of
three key components

Infrastructure	Content	Implementation
Clients/Browser	Objects files, pictures, videos, …	URL: name content
Servers	*organized* in	HTTP: transport content
Proxies	Web sites a collection of objects	

We'll focus on
its implementation

Infrastructure	Content	Implementation
Clients/Browser	Objects files, pictures, videos, …	URL: name content
Servers	*organized* in	HTTP: transport content
Proxies	Web sites a collection of objects	

Infrastructure	Content	Implementation
Clients/Browser	Objects files, pictures, videos, …	**URL: name content**
Servers	*organized* in	HTTP: transport content
Proxies	Web sites a collection of objects	

A Uniform Resource Locator (URL)
refers to an Internet ressource

protocol://hostname[:port]/directory_path/resource

protocol://hostname[:port]/directory_path/resource

HTTP(S)
FTP
SMTP…

protocol://hostname[:port]/directory_path/resource

DNS Name
IP address

default to protocol's standard
HTTP:80, HTTPs:443

protocol://hostname[:port]/directory_path/resource

protocol://hostname[:port]/directory_path/resource

identify the resource
on the destination

Infrastructure	Content	Implementation
Clients/Browser Servers Proxies	Objects files, pictures, videos, ... *organized* in Web sites a collection of objects	URL: name content **HTTP: transport content**

HTTP is a rather simple synchronous request/reply protocol

HTTP is layered over a bidirectional byte stream
almost always TCP

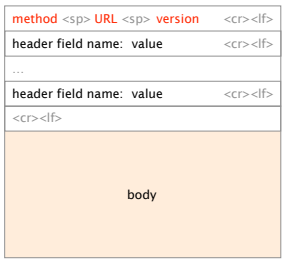
HTTP is text-based (ASCII)
human readable, easy to reason about

HTTP is stateless
it maintains *no info* about past client requests

HTTP clients make request to the server

HTTP request

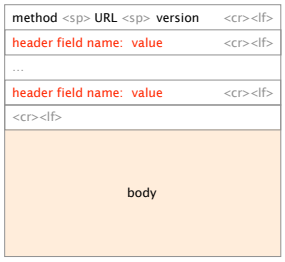
method <sp> URL <sp> version	<cr><lf>
header field name: value	<cr><lf>
...	
header field name: value	<cr><lf>
<cr><lf>	
body	



method	GET	return resource
	HEAD	return headers only
	POST	send data to server (forms)
URL		relative to server *(e.g., /index.html)*
version		1.0, 1.1, 2.0

HTTP clients make request to the server

HTTP request



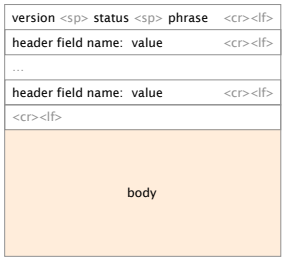
Request headers are of variable lengths, but still, human readable

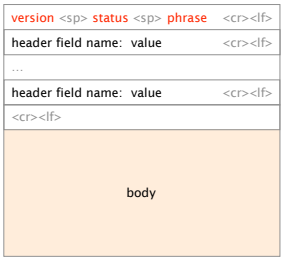
Uses

- Authorization info
- Acceptable document types/encoding
- From (user email)
- If-Modified-Since
- Referrer (cause of the request)
- User Agent (client software)

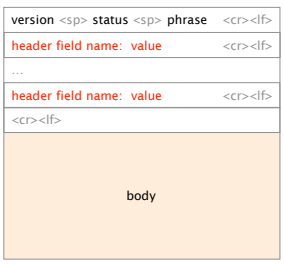
HTTP servers answers to clients' requests

HTTP response





	3 digit response code		reason phrase
Status	1XX	informational	
	2XX	success	200 OK
	3XX	redirection	301 Moved Permanently
			303 Moved Temporarily
			304 Not Modified
	4XX	client error	404 Not Found
	5XX	server error	505 Not Supported



Like request headers, response headers are of variable lengths and human-readable

Uses

- **Location** (for redirection)
- **Allow** (list of methods supported)
- **Content encoding** (*e.g.*, gzip)
- **Content-Length**
- **Content-Type**
- **Expires** (caching)
- **Last-Modified** (caching)

HTTP is a stateless protocol,
meaning each request is treated independently

advantages	disadvantages
server-side scalability	some applications need state! (shopping cart, user profiles, tracking)
failure handling is trivial	

How can you maintain state in a stateless protocol?

HTTP makes the client maintain the state.
This is what the so-called cookies are for!

client stores small state
on behalf of the server *X*

client sends state
in all future requests to *X*

can provide authentication

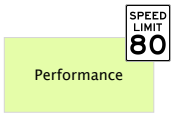
telnet google.ch 80

request
```
GET / HTTP/1.1
Host: www.google.ch
```

answer
```
HTTP/1.1 200 OK
Date: Sun, 01 May 2016 14:10:30 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Server: gws
```

browser will relay this value in following requests

```
Set-Cookie:
NID=79=g6IgURTq_BG4hSTFhEy1gTVFmSncQVsy
TJI260B3xyiXqy2wxD2YeHq1bBlwFyLoJhSc7jmcA
6TlFIBY7-
dW5IhjiRiQmY1JxT8hGCOtnLjfCL0mYcBBkpk8X4
NwAO28; expires=Mon, 31-Oct-2016 14:10:30
GMT; path=/; domain=.google.ch; HttpOnly
```



Performance goals vary depending
on who you ask

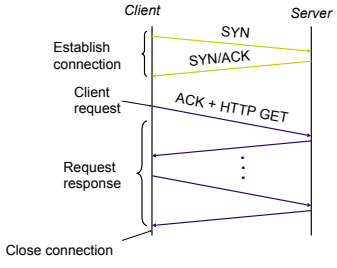
	User	Network operators	Content provider
			
wish	fast downloads high availability	no overload	happy users cost-effective infrastructure
solution	Improve HTTP to compensate for TCP weakspots	Caching and Replication	

User

wish fast downloads
high availability

solution Improve HTTP to compensate for TCP weakspots

Relying on TCP forces a HTTP client to
open a connection before exchanging anything



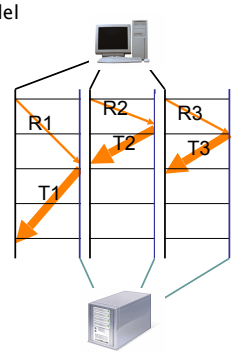
Most Web pages have multiple objects,
naive HTTP opens one TCP connection for each…

Fetching *n* objects requires ~2*n* RTTs

TCP establishment
HTTP request/response

One solution to that problem is to use multiple TCP connections in parallel

User	Happy!
Content provider	Happy!
Network operator	Not Happy!
	Why?



Another solution is to use persistent connections across multiple requests, default in HTTP/1.1

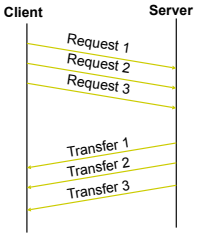
Avoid overhead of connection set-up and teardown
clients or servers can tear down the connection

Allow TCP to learn more accurate RTT estimate
and with it, more precise timeout value

Allow TCP congestion window to increase
and therefore to leverage higher bandwidth

Yet another solution is to pipeline requests & replies asynchronously, on one connection

- batch requests and responses to reduce the number of packets
- multiple requests can be packed into one TCP segment



Considering the time to retrieve *n* small objects, pipelining wins

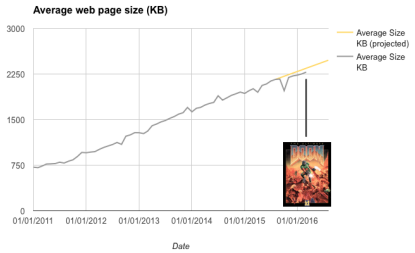
	# RTTS
one-at-a-time	~2*n*
M concurrent	~2*n*/M
persistent	~*n*+1
pipelined	2

Considering the time to retrieve *n* big objects, there is no clear winners as bandwidth matters more

RTTS

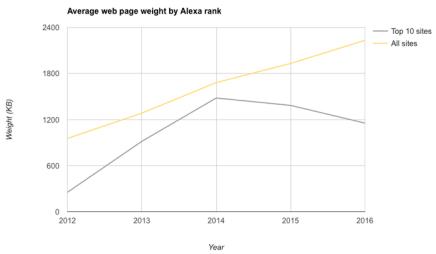
$$\sim n * \frac{\text{avg. file size}}{\text{bandwidth}}$$

Today, the average webpage size is 2.3 MB
as much as the original DOOM game…



(*) see https://mobiforge.com/research-analysis/the-web-is-doom

Top web sites have decreased in size though
because they care about TCP performance



(*) see https://mobiforge.com/research-analysis/the-web-is-doom

User | Network operators | Content provider

wish | no overload | happy users, cost-effective infrastructure

solution | Caching and Replication

Caching leverages the fact that highly popular content largely overlaps

Just think of how many times
you request the facebook logo
per day
vs
how often it *actually* changes

Caching it save time for your browser
and decrease network and server load

Yet, a significant portion of the HTTP objects are "uncachable"

Examples		
	dynamic data	stock prices, scores, ...
	scripts	results based on parameters
	cookies	results may be based on passed data
	SSL	cannot cache encrypted data
	advertising	wants to measure # of hits ($$$)

To limit staleness of cached objects, HTTP enables a client to validate cached objects

Server hints when an object expires (kind of TTL)
as well as the last modified date of an object

Client conditionally requests a ressources
using the "if-modified-since" header in the HTTP request

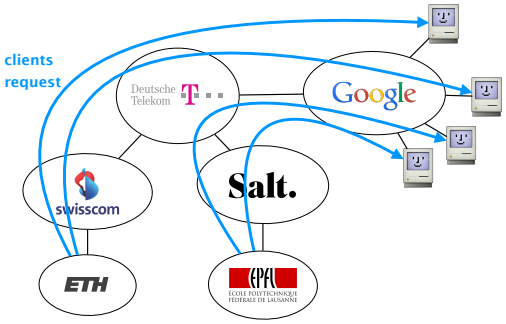
Server compares this against "last modified" time
of the resource and returns:

- Not Modified if the resource has not changed
- OK with the latest version

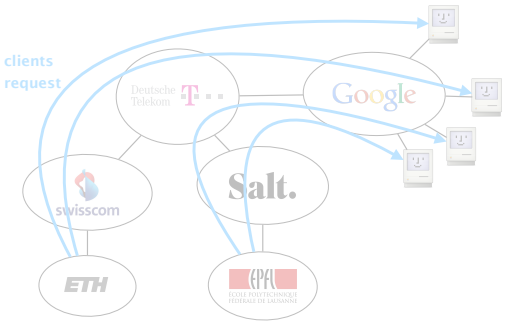
Caching can and is performed at different locations

client	browser cache
close to the client	forward proxy Content Distribution Network (CDN)
close to the destination	reverse proxy

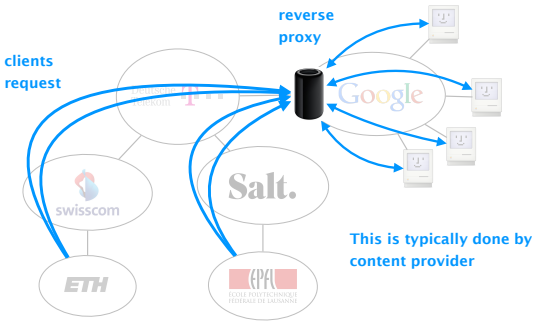
Many clients request the same information



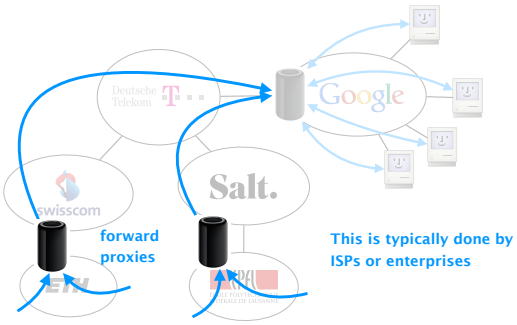
This increases servers and network's load, while clients experience unnecessary delays



Reverse proxies cache documents close to servers, decreasing their load



Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies



Network operators

Content provider

NETFLIX

wish

no overload

happy users

cost-effective infrastructure

solution

Caching and Replication

The idea behind replication is to duplicate popular content all around the globe

Spreads load on server

e.g., across multiple data-centers

Places content closer to clients

only way to beat the "speed-of-light"

Helps speeding up uncachable content

still have to pull it, but from closer

The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica

DNS-based

returns ≠ IP addresses

based on

- client geo-localization
- server load

BGP Anycast

advertise the same IP prefix from different locations

avoided in practice, any idea why?

Akamai is one of the largest CDNs in the world, boasting servers in more than 20,000 locations

http://wwwnui.akamai.com/gnet/globe/index.html

Akamai uses a combination of

- *pull* caching

  direct result of clients requests
- *push* replication

  when expecting high access rate

together with some dynamic processing

dynamic Web pages, transcoding,…

"Akamaizing" content is easily done by modifying content to reference the Akamai's domains

Akamai creates domain names for each client

a128.g.akamai.net for cnn.com

Client modifies its URL to refer to Akamai's domain

http://www.cnn.com/image-of-the-day.gif

becomes

http://a128.g.akamai.net/image-of-the-day.gif

Requests are now sent to the CDN infrastructure

Web

Video Streaming

HTTP-based

We want the highest video quality

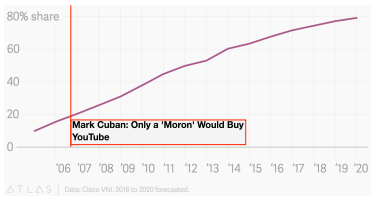
(c) copyright 2008, Blender Foundation / www.bigbuckbunny.org, CC-BY-3.0

1

Without seeing this …

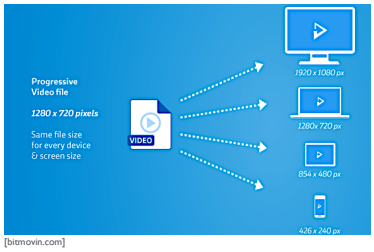
2

Why should you care? Just look at this: video's share of global internet traffic



3

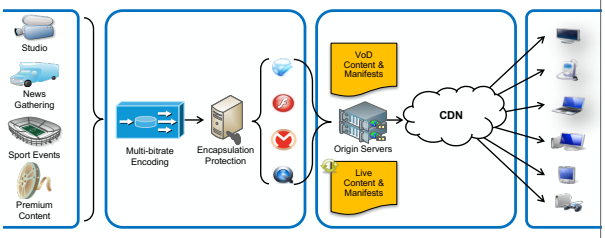
A naive approach: one-size-fits-all



[bitmovin.com]

4

In practice, things are complex



[Adapted from: Adaptive Streaming of Traditional and Omnidirectional Media, Begen & Timmerer, ACM SIGCOMM Tutorial, 2017]

5

The three steps behind most contemporary solutions

- Encode video in multiple bitrates
- Replicate using a content delivery network
- Video player picks bitrate adaptively
  - Estimate connection's available bandwidth
  - Pick a bitrate ≤ available bandwidth

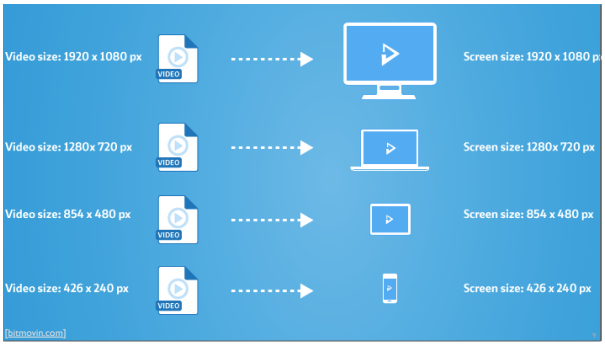
6

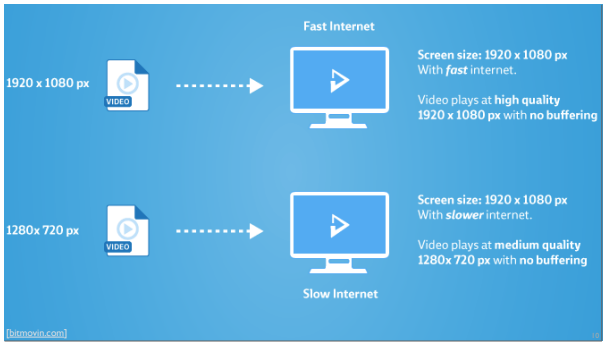
Encoding | Replication | Adaptation

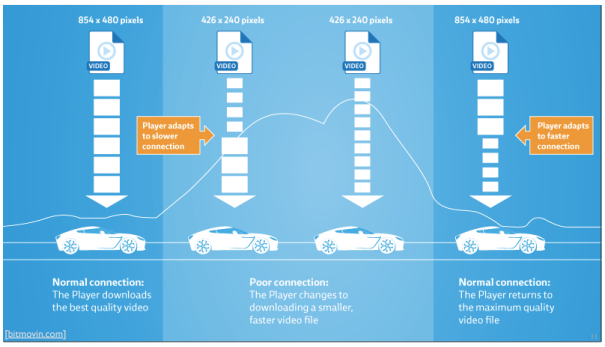
7

Encoding | Replication | Adaptation

8







Simple solution for encoding: use a “bitrate ladders”

Bitrate (kbps)	Resolution
235	320x240
375	384x288
560	512x384
750	512x384
1050	640x480
1750	720x480
2350	1280x720
3000	1280x720
4300	1920x1080
5800	1920x1080

[netflix.com]

Problem: this doesn't take into account the variability in the video content (slow moving vs. fast moving)

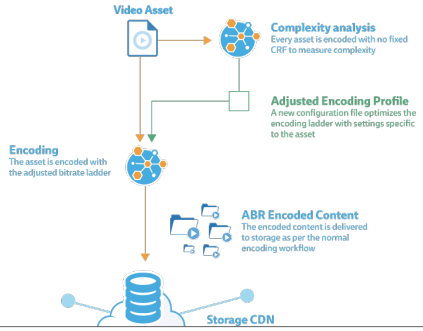
Bitrate (kbps)	Resolution
235	320x240
375	384x288
560	512x384
750	512x384
1050	640x480
1750	720x480
2350	1280x720
3000	1280x720
4300	1920x1080
5800	1920x1080

[netflix.com]

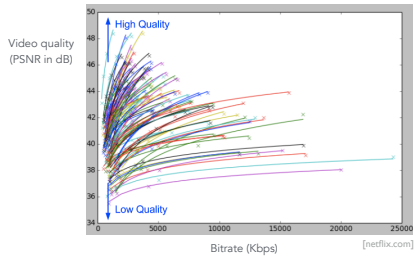

[netflix.com]


[bitmovin.com]

Encoding

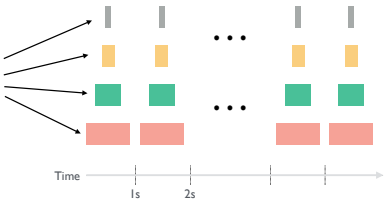


Encoding

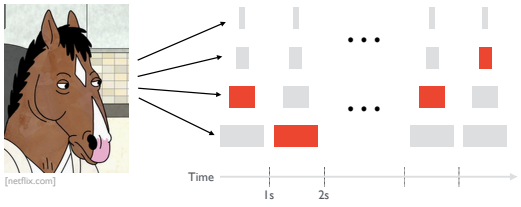


Your player download “chunks” of video at different bitrates


[netflix.com]



Depending on your network connectivity, your player fetches chunks of different qualities



17

Your player gets metadata about chunks via "Manifest"

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="urn:mpeg:DASH:schema:MPD:2011"
     xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011"
     profiles="urn:mpeg:dash:profile:isoff-main:2011"
     type="static"
     mediaPresentationDuration="PT0H9M56.46S"
     minBufferTime="PT15.0S">
  <BaseURL>http://witestlab.poly.edu/~ffund/video/2s_480p_only/</BaseURL>
  <Period start="PT0S">
      <AdaptationSet bitstreamSwitching="true">
    <Representation id="0" codecs="avc1" mimeType="video/mp4"
       width="480" height="360" startWithSAP="1" bandwidth="101492">
      <SegmentBase>
        <Initialization sourceURL="bunny_2s_100kbit/bunny_100kbit.mp4"/>
      </SegmentBase>
      <SegmentList duration="2">
        <SegmentURL media="bunny_2s_100kbit/bunny_2s1.m4s"/>
        <SegmentURL media="bunny_2s_100kbit/bunny_2s2.m4s"/>
        <SegmentURL media="bunny_2s_100kbit/bunny_2s3.m4s"/>
        <SegmentURL media="bunny_2s_100kbit/bunny_2s4.m4s"/>
        <SegmentURL media="bunny_2s_100kbit/bunny_2s5.m4s"/>
        <SegmentURL media="bunny_2s_100kbit/bunny_2s6.m4s"/>
```

[witestlab.poly.edu]

18

Encoding | Replication | Adaptation

19

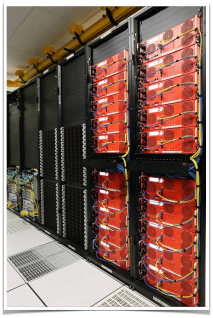
NETFLIX

Open Connect: Starting from a Greenfield (a mostly Layer 0 talk)

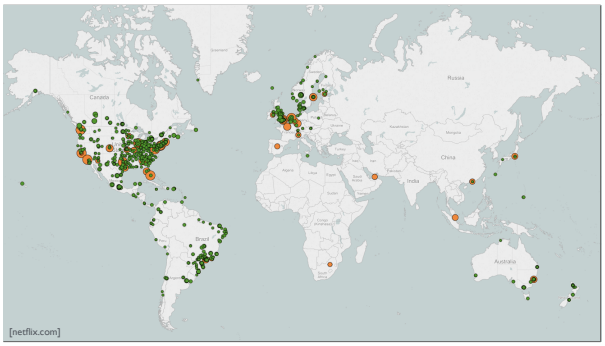
Dave Temkin
06/01/2015

Storage Appliance
- Still 4U high
- ~550 watts
- 288 TB of storage
- 2x 10G ports
- 20Gbit/s delivery

Flash Appliance
- 1U
- ~175 watts
- 24 TB of flash
- 2x 40G ports
- 40Gbit/s delivery



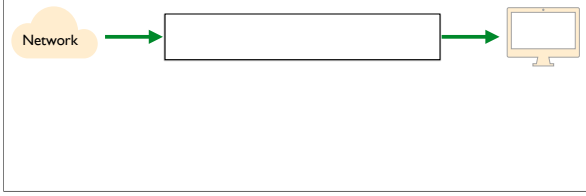
20

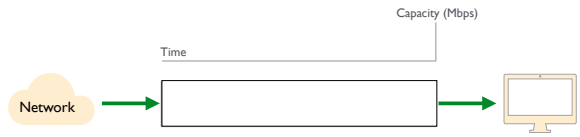


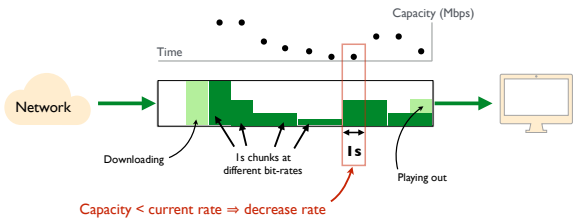
22

Encoding | Replication | Adaptation

23





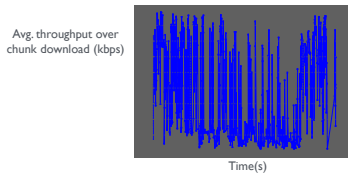


Common solution approach

- Encode video in multiple bitrates
- Replicate using a content delivery network
- Video player picks bitrate adaptively
  - Estimate connection's available bandwidth
  - Pick a bitrate ≤ available bandwidth

27

Estimating available capacity



[A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service, Huang et al., ACM SIGCOMM 2014]

28

Estimating available capacity

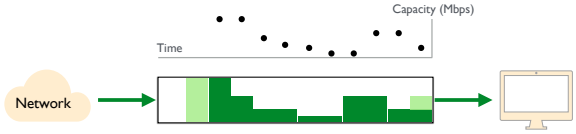
"A random sample of 300,000 Netflix sessions shows that roughly 10% of sessions experience a median throughput less than half of the 95th percentile throughput."

"20–30% of rebuffers are unnecessary"

[A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service, Huang et al., ACM SIGCOMM 2014]

29

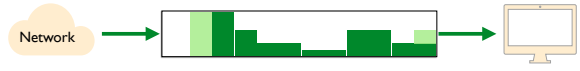
Capacity estimation



Decide based on the buffer alone?

30

Buffer-based adaptation



Nearly full buffer ⇒ large rate

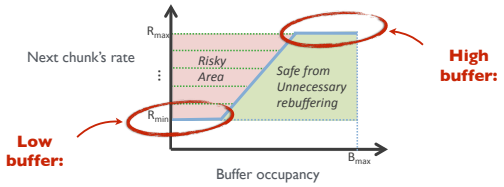
31

Buffer-based adaptation

Nearly empty buffer ⇒ small rate

32

Buffer-based adaptation



[A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service, Huang et al., ACM SIGCOMM 2014]

33

Problem: startup phase?

Pick a rate based on immediate past throughput

Summary

- Encode video in multiple bitrates
- Replicate using a content delivery network
- Video player picks bitrate adaptively
- Problem of active research interest, many competing algorithms and objectives

35

Communication Networks

Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
April 29 2019