

# Communication Networks

Prof. Laurent Vanbever

Online/COVID-19 Edition

## Communication Networks

Spring 2020



Laurent Vanbever  
[nsg.ee.ethz.ch](mailto:nsg.ee.ethz.ch)

ETH Zürich (D-ITET)  
May 4 2020

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

Last week on  
Communication Networks

Congestion  
Control

DNS



[google.ch](https://www.google.ch) ↔ 172.217.16.131  
(the beginning)

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 <b>RWND</b>	How many bytes can be sent without overflowing the receiver buffer? based on the receiver input
Congestion Window <b>CWND</b>	How many bytes can be sent without overflowing the routers? based on network conditions
Sender Window	$\text{minimum}(\text{CWND}, \text{RWND})$

## The 2 key mechanisms of Congestion Control



## The 2 key mechanisms of Congestion Control



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

<p>           duplicated ACKs                        timeout         </p>	<p> <b>mild</b> congestion signal            packets are still making it              <b>severe</b> congestion signal            multiple consequent losses         </p>
---	--

## The 2 key mechanisms of Congestion Control



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

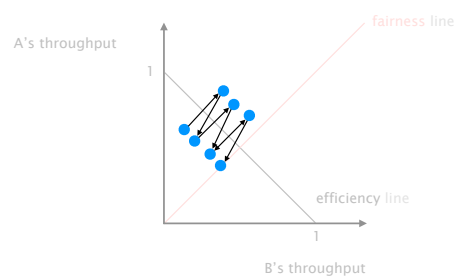
Increase policy      **cwnd = 1** initially  
                                 **cwnd += 1** upon receipt of an ACK

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

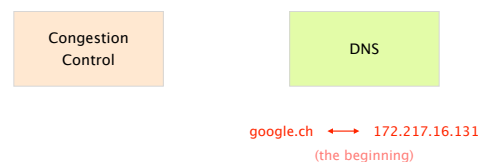
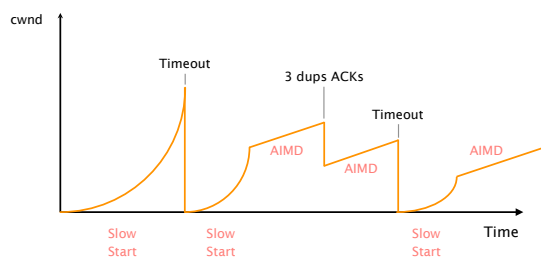
	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

#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"



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

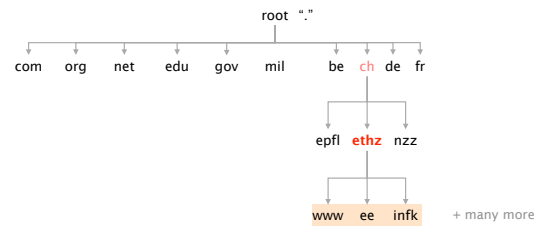
To scale, DNS adopt **three** intertwined hierarchies

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

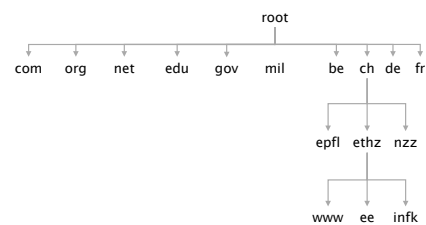
A name, *e.g.* ee.ethz.ch, represents  
a leaf-to-root path in the hierarchy



management

hierarchy of authority  
over names

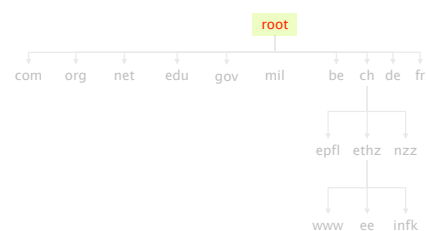
The DNS system is  
hierarchically administered



infrastructure

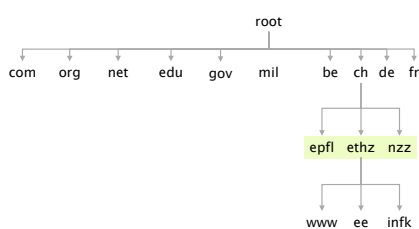
hierarchy of DNS servers

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>

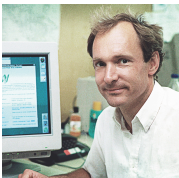


Each server knows the address of all children

This week on  
Communication Networks



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



Tim Berners-Lee Photo: CERN

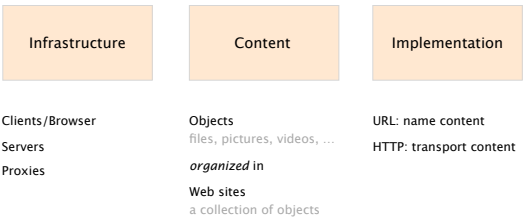
His goal:  
provide distributed access to data

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

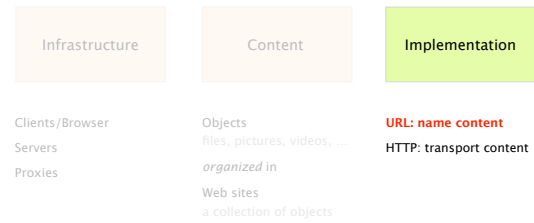
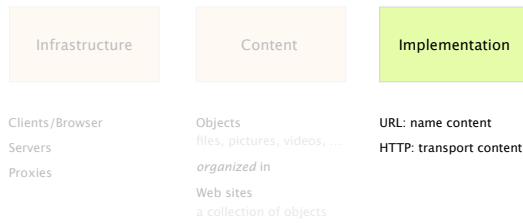
The Web was and still is so successful as  
it enables everyone to self-publish content

- 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



We'll focus on  
its implementation



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

protocol://hostname[:port]/directory\_path/resource

protocol://hostname[:port]/directory\_path/resource

HTTP(S)  
FTP  
SMTP...

protocol://hostname[:port]/directory\_path/resource

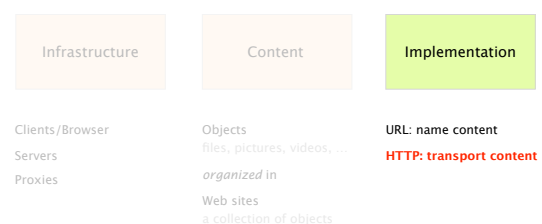
DNS Name  
IP address

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



HTTP is a rather simple  
synchronous request/reply protocol

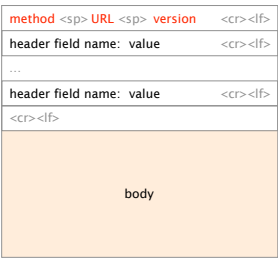
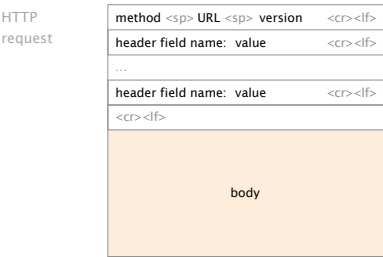
HTTP is layered over a bidirectional byte stream  
typically TCP, but QUIC is ramping up

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

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

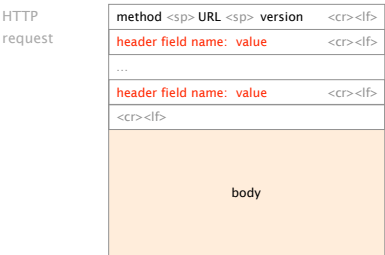


HTTP clients make request to the server



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



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

Uses	Authorization info
	Acceptable document types/encoding
	From (user email)
	Host (identify the server to which the request is sent)
	If-Modified-Since
	Referrer (cause of the request)
	User Agent (client software)

Uses

- Authorization info
- Acceptable document types/encoding
- From (user email)
- Host (identify the server to which the request is sent)
- If-Modified-Since
- Referrer (cause of the request)
- User Agent (client software)

Recall that multiple DNS names can map to the same IP address

name	DNS	IP address
www.ethz.ch		129.132.19.216
vanbever.eu		82.130.102.71
route-aggregation.net		82.130.102.71
comm-net.ethz.ch		82.130.102.71

The "Host" header indicates the server (82.130.102.71) the desired domain name (this is known as virtual hosting)

name	DNS	IP address
www.ethz.ch		129.132.19.216
vanbever.eu		82.130.102.71
route-aggregation.net		82.130.102.71
comm-net.ethz.ch		82.130.102.71

Virtual hosting enables *one* IP address to host *multiple* websites

82.130.102.71  
(resolved through DNS)

```
connect openssl s_client -crif -quiet -connect comm-net.ethz.ch:443

request GET / HTTP/1.1
Host: comm-net.ethz.ch

answer HTTP/1.1 200 OK
Date: Fri, 01 May 2020 08:36:56 GMT
Server: Apache/2.4.18 (Ubuntu)

<head>
...
<title>Communication Networks 2020</title>
....
```

82.130.102.71  
(resolved through DNS)

```
connect openssl s_client -crif -quiet -connect comm-net.ethz.ch:443

request GET / HTTP/1.1
Host: vanbever.eu

answer HTTP/1.1 200 OK
Date: Fri, 01 May 2020 08:44:26 GMT
Server: Apache/2.4.18 (Ubuntu)

<head>
...
<title>Laurent Vanbever</title>
....
```

HTTP servers answers to clients' requests

HTTP response

version <sp> status <sp> phrase <cr><lf>
header field name: value <cr><lf>
...
header field name: value <cr><lf>
<cr><lf>
body

version <sp> status <sp> phrase <cr><lf>
header field name: value <cr><lf>
...
header field name: value <cr><lf>
<cr><lf>
body

	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

version	<sp>	status	<sp>	phrase	<cr><lf>
header field name:	value				<cr><lf>
...					
header field name:	value				<cr><lf>
<cr><lf>					
body					

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 <b>need state!</b> (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

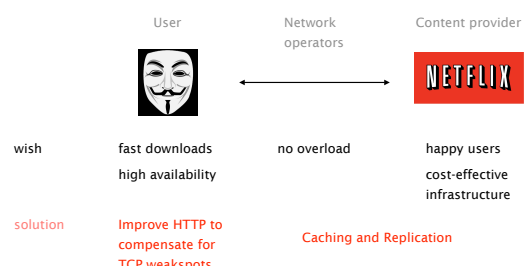
```
telnet google.ch 80

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  Set-Cookie:
will relay  NID=79=g6lgURTq_8G4hSTFhEy1gTVFmSncQVsy
this value  TjI260B3xyiXqy2wxD2YeHq1b8lwFyLajhSc7jmcA
in following 6TIFIBY7-
requests    dW5lhjIRIQmY1JxT8hGCOtnLjfCL0mYcBBkp8X4
           NwAO28; expires=Mon, 31-Oct-2016 14:10:30
           GMT; path=/; domain=.google.ch; HttpOnly
```

Performance goals vary depending on who you ask



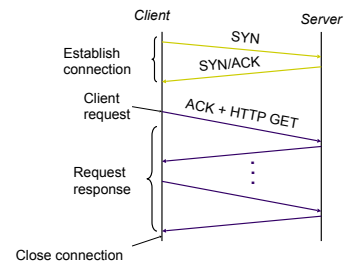
User



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

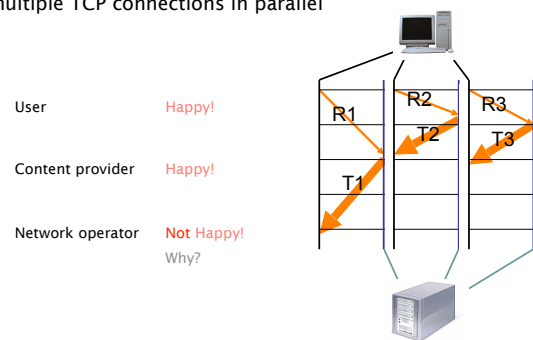


Most Web pages have multiple objects,  
naive HTTP opens one TCP connection for each...

Fetching  $n$  objects requires  $\sim 2n$  RTTs

TCP establishment  
HTTP request/response

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



Another solution is to use persistent connections  
across multiple requests (the default in HTTP/1.1)

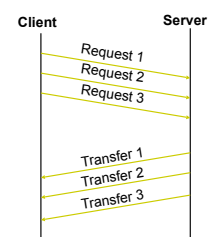
Avoid overhead of connection set-up and teardown  
clients or servers can tear down the connection

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



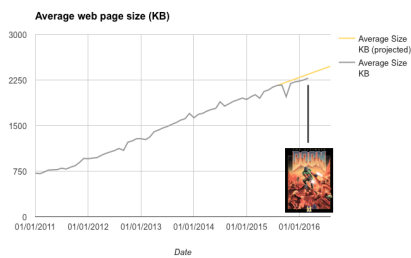
Considering the time to retrieve  $n$  small objects,  
pipelining wins

	# RTTs
one-at-a-time	$\sim 2n$
M concurrent	$\sim 2n/M$
persistent	$\sim n+1$
pipelined	2

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

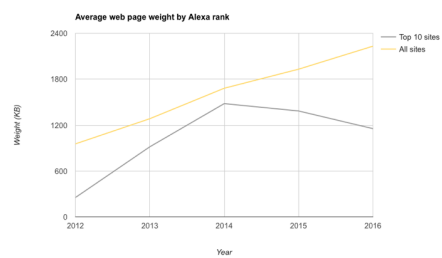
# RTTs
$\sim n \cdot \frac{\text{avg. file size}}{\text{bandwidth}}$

The average webpage size nowadays is 2.3 MB  
as much as the original DOOM game...

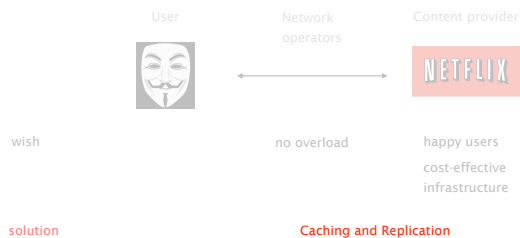


(\*) see <https://mobiforge.com/research-analysis/the-web-is-doom>

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




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



solution

Caching and Replication

Caching leverages the fact that  
highly popular content **largely** overlaps

Just think of how many times  
you request the   
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 resources  
using the "if-modified-since" header in the HTTP request

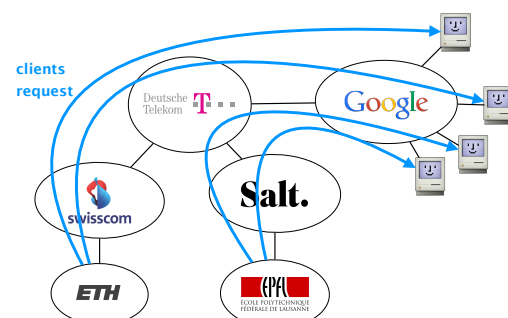
Server compares this against "last modified" time  
of the resource and returns:

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

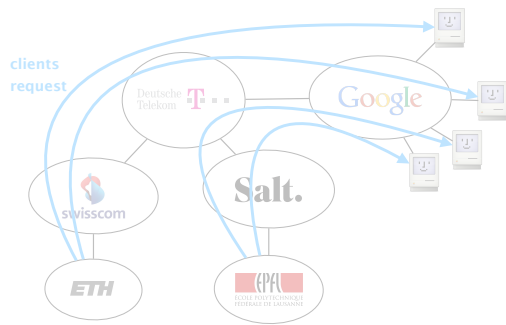
Caching can and is performed at different locations

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

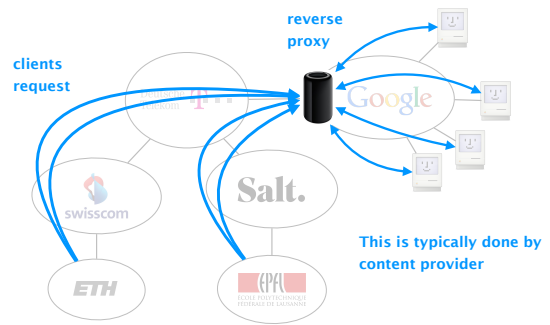
Many clients request the same information



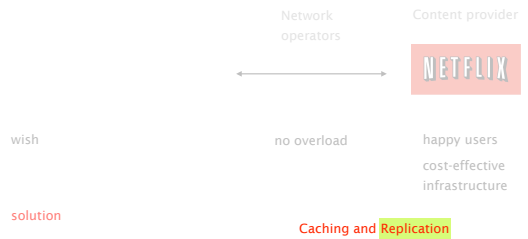
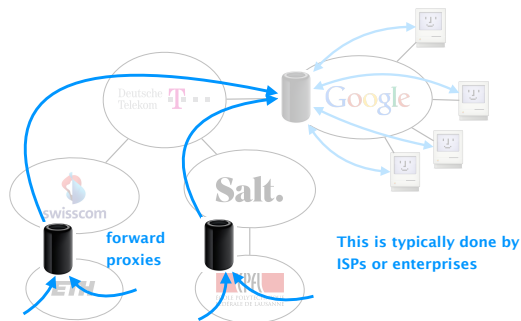
This increases servers and network's load, while clients experience unnecessary delays



Reverse proxies cache documents close to servers, decreasing their load



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



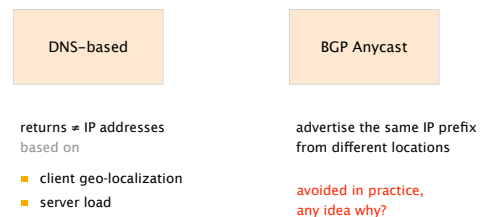
The idea behind replication is to duplicate popular content all around the globe

Spreads load on server  
e.g., across multiple data-centers

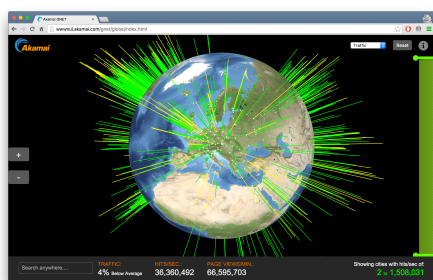
Places content closer to clients  
only way to beat the "speed-of-light"

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



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



<http://wwwui.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

## Communication Networks

Spring 2020



Laurent Vanbever  
[nsg.ee.ethz.ch](mailto:nsg.ee.ethz.ch)

ETH Zürich (D-ITET)  
May 4 2020