Communication Networks

Prof. Laurent Vanbever

Communication Networks

Spring 2018

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
May 14 2018

Materials inspired from Scott Shenker & Jennifer Rexford

Unterrichtsbeurteilung
aka course evaluation

Please fill in the survey!
You should have received the link by email

Two weeks ago on
Communication Networks

TCP Congestion Control

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

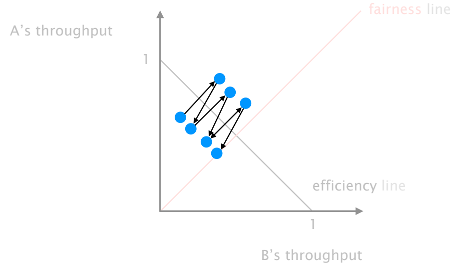
Intuition	Start slow but rapidly increase until a packet drop occurs	
Increase policy	cwnd = 1	initially
	cwnd += 1	upon receipt of an ACK

#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

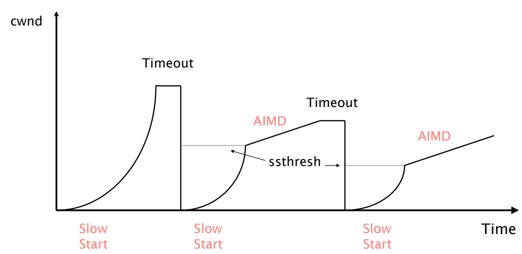
AIMD converge to fairness and efficiency, it then fluctuates around the optimum (in a stable way)



TCP congestion control in less than 10 lines of code

```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:
    if (cwnd < ssthresh):
        /* Slow Start*/
        cwnd = cwnd + 1
    else:
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2
    cwnd = 1
```

The congestion window of a TCP session typically undergoes multiple cycles of slow-start/AIMD



Going back all the way back to 0 upon timeout completely destroys throughput

solution — Avoid timeout expiration... which are usually >500ms

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

TCP automatically resends a segment after receiving 3 duplicates ACKs for it

this is known as a "fast retransmit"

After a fast retransmit, TCP switches back to AIMD, without going all way the back to 0

this is known as "fast recovery"

TCP congestion control (almost complete)

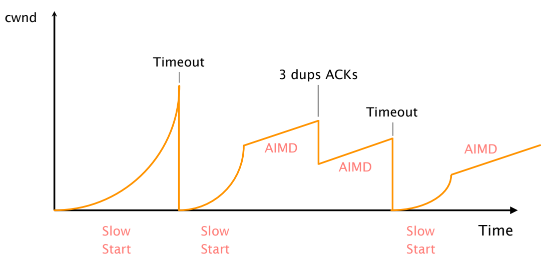
```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:
    if (cwnd < ssthresh):
        /* Slow Start*/
        cwnd = cwnd + 1
    else:
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd
    dup_ack = 0
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2
    cwnd = 1

Duplicate ACKs received:
    dup_ack ++;
    if (dup_ack >= 3):
        /* Fast Recovery */
        ssthresh = cwnd/2
        cwnd = ssthresh
```

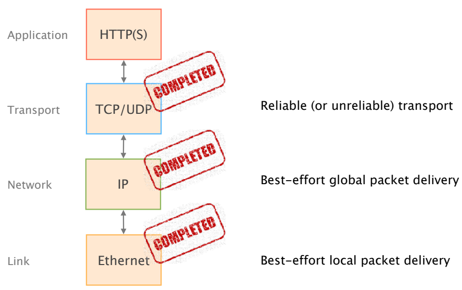
```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:
    if (cwnd < ssthresh):
        /* Slow Start*/
        cwnd = cwnd + 1
    else:
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd
    dup_ack = 0
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2
    cwnd = 1

Duplicate ACKs received:
    dup_ack ++;
    if (dup_ack >= 3):
        /* Fast Recovery */
        ssthresh = cwnd/2
        cwnd = ssthresh
```

Congestion control makes TCP throughput look like a "sawtooth"



We now have completed the transport layer (!)



This week on
Communication Networks

DNS

Web

google.ch ⟷ 172.217.16.131

http://www.google.ch

DNS

Web

google.ch ⟷ 172.217.16.131

Internet has one **global system** for

- **addressing** hosts — IP
  by design
- **naming** hosts — DNS
  by "accident", an afterthought

Internet has one **global system** for

- **naming** hosts — DNS
  by "accident", an afterthought

Using Internet services can be divided into four logical steps

step 1	A person has name of entity she wants to access	www.ethz.ch
step 2	She invokes an application to perform the task	Chrome
step 3	The application invokes DNS to resolve the name into an IP address	129.132.19.216
step 4	The application invokes transport protocol to establish an app-to-app connection	

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

In practice, names can be mapped to more than one IP

name	⟷ DNS	IP address	
www.ethz.ch		129.132.19.216	
www.google.ch		195.176.255.241	(load-balancing)
		195.176.255.245	

In practice, IPs can be mapped by more than one name

name	⟷ DNS	IP address
www.ethz.ch		129.132.19.216
www.vanbever.eu		188.165.240.60
www.routeur.be		188.165.240.60

How does one resolve a name into an IP?

initially	*all* host to address mappings were in a file called hosts.txt in /etc/hosts
problem	scalability in terms of query load & speed management consistency availability

When you need… more flexibility,
you add… a layer of indirection

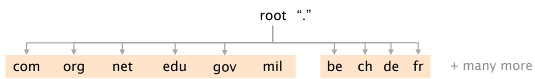
When you need… more scalability,
you add… a hierarchical structure

To scale,
DNS adopt three intertwined hierarchies

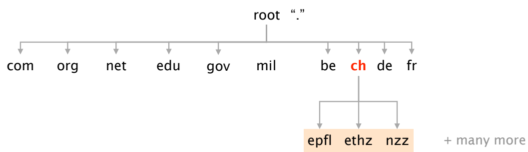
naming structure	hierarchy of addresses https://www.ee.ethz.ch/de/departement/
management	hierarchy of authority over names
infrastructure	hierarchy of DNS servers

naming structure addresses are hierarchical
https://www.ee.ethz.ch/de/departement/

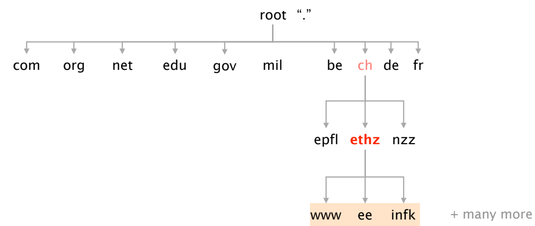
Top Level Domain (TLDs) sit at the top



Domains are subtrees

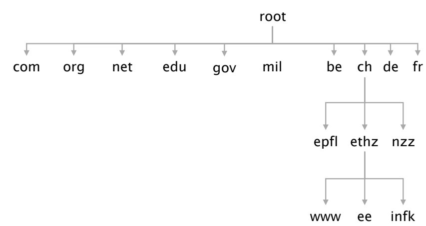


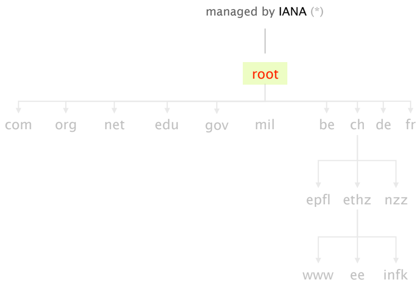
A name, *e.g.* ee.ethz.ch, represents
a leaf-to-root path in the hierarchy



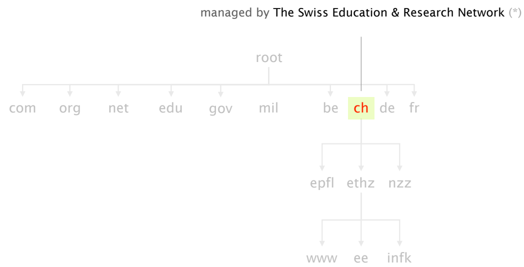
management hierarchy of authority
over names

The DNS system is
hierarchically administered

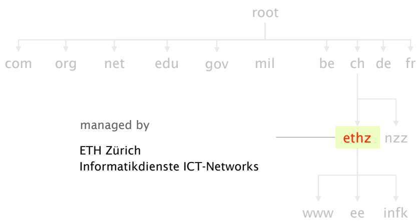




(*) see http://www.iana.org/domains/root/db



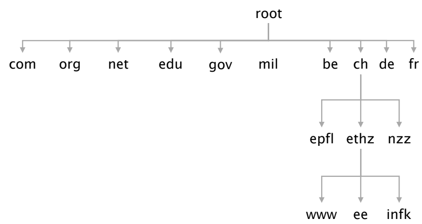
(*) see https://www.switch.ch/about/id/



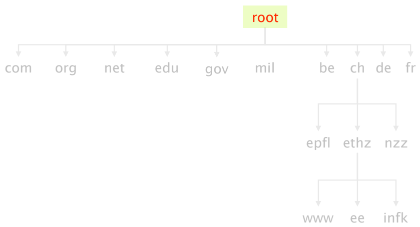
Hierarchical administration means that name collision is trivially avoided

infrastructure — hierarchy of DNS servers

The DNS infrastructure is hierarchically organized

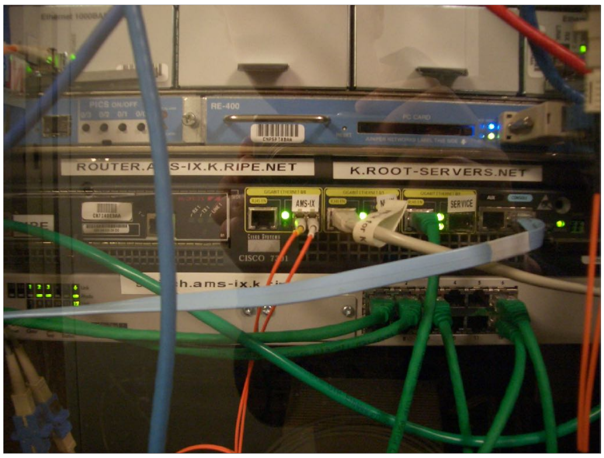


13 root servers (managed professionally) serve as root (*)



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

a. root-servers.net	VeriSign, Inc.
b. root-servers.net	University of Southern California
c. root-servers.net	Cogent Communications
d. root-servers.net	University of Maryland
e. root-servers.net	NASA
f. root-servers.net	Internet Systems Consortium
g. root-servers.net	US Department of Defense
h. root-servers.net	US Army
i. root-servers.net	Netnod
j. root-servers.net	VeriSign, Inc.
k. root-servers.net	RIPE NCC
l. root-servers.net	ICANN
m. root-servers.net	WIDE Project



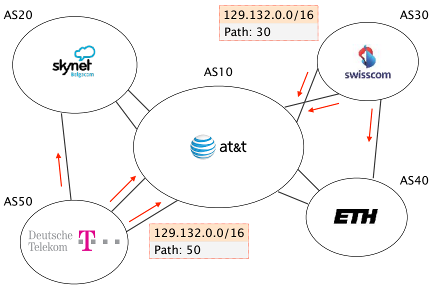
To scale root servers,
operators rely on BGP anycast

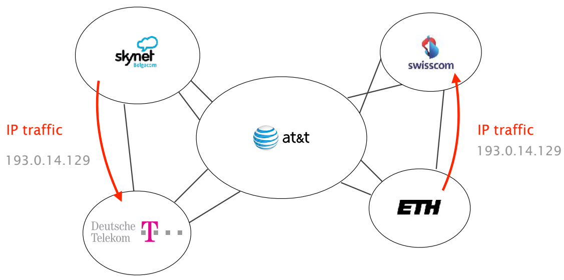
Intuition

Routing finds shortest-paths

If several locations announce the same prefix,
then routing will deliver the packets to
the "closest" location

This enables seamless replications of resources



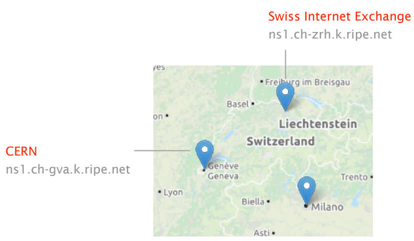


Do you see any problems in
performing load-balancing this way?

Instances of the k-root server (*) are hosted
in more than 40 locations worldwide

(*) see k.root-servers.org

Two of these locations are in Switzerland:
in Zürich and in Geneva

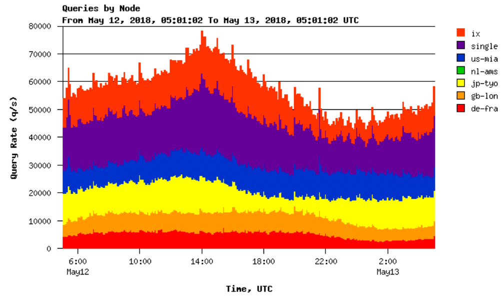


All locations announce 193.0.14.0/23 in BGP,
with 193.0.14.129 being the IP of the server

Two of these locations are in Switzerland:
in Zürich and in Geneva

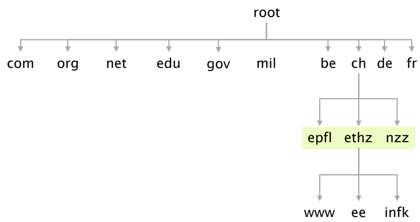
Do you mind guessing which one we use, here… in Zürich?

Each instance receives up to 70k queries per second
summing up to more than 4 billions queries per day



TLDs server are also managed professionally
by private or non-profit organization

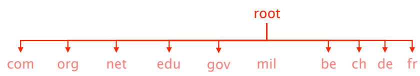
The bottom (and bulk) of the hierarchy is
managed by Internet Service Provider or locally



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

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

Each root server knows
the address of all TLD servers

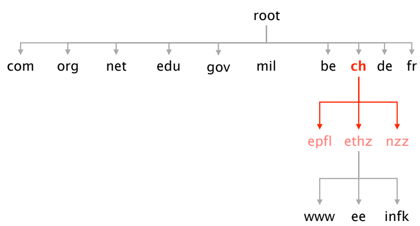



```
lvanbever:~$ dig @a.root-servers.net ch.

ch.                    172800  IN      NS      a.nic.ch.
ch.                    172800  IN      NS      b.nic.ch.
ch.                    172800  IN      NS      c.nic.ch.
ch.                    172800  IN      NS      d.nic.ch.
ch.                    172800  IN      NS      e.nic.ch.
ch.                    172800  IN      NS      f.nic.ch.
ch.                    172800  IN      NS      h.nic.ch.
```

From there on,
each server knows the address of all children

Any .ch DNS server knowns
the addresses of all sub-domains



To scale,
DNS adopt three intertwined hierarchies

naming structure	addresses are hierarchical https://www.ee.ethz.ch/de/departement/
management	hierarchy of authority over names
infrastructure	hierarchy of DNS servers

To ensure availability, each domain must have at least a primary and secondary DNS server

Ensure name service availability
as long as one of the servers is up

DNS queries can be load-balanced
across the replicas

On timeout, client use alternate servers
exponential backoff when trying the same server

Overall, the DNS system is highly scalable, available, and extensible

scalable	#names, #updates, #lookups, #users, but also in terms of administration
available	domains replicate independently of each other
extensible	any level (including the TLDs) can be modified independently

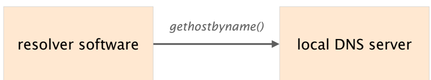
You've founded next-startup.ch and want to host it yourself, how do you insert it into the DNS?

You register next-startup.ch at a registrar *X*
e.g. Swisscom or GoDaddy

Provide *X* with the name and IP of your DNS servers
e.g., [ns1.next-startup.ch,129.132.19.253]

You set-up a DNS server @129.132.19.253
define A records for www, MX records for next-startup.ch...

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)

DNS query and reply uses UDP (port 53), reliability is implemented by repeating requests (*)

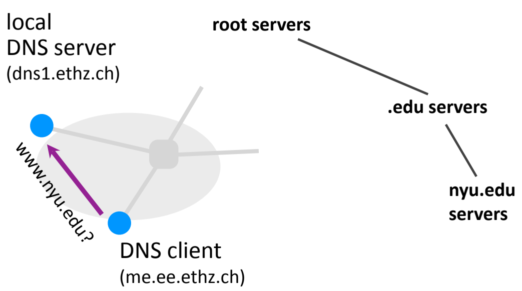
(*) see Book (Section 5)

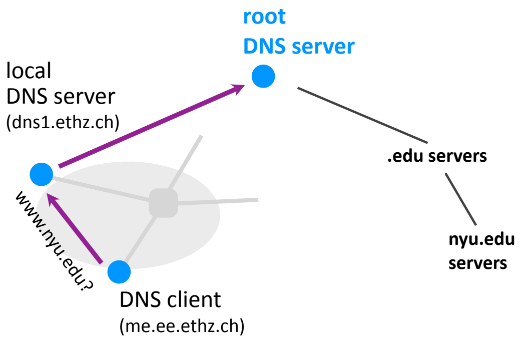
A DNS server stores Resource Records composed of a (name, value, type, TTL)

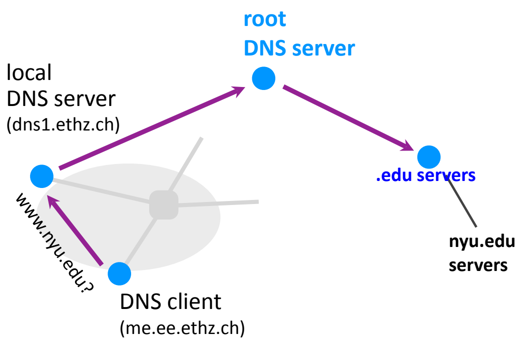
Records	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

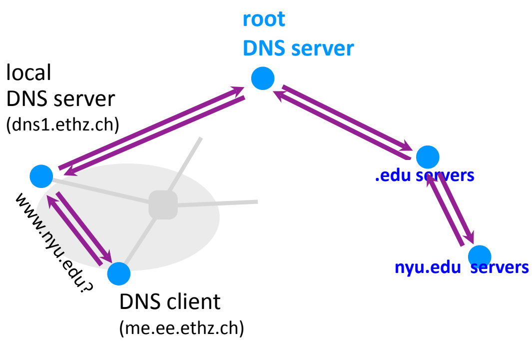
DNS resolution can either be
recursive or iterative

When performing a recursive query,
the client offload the task of resolving to the server

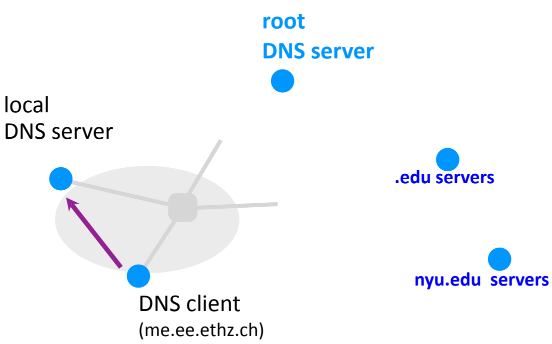


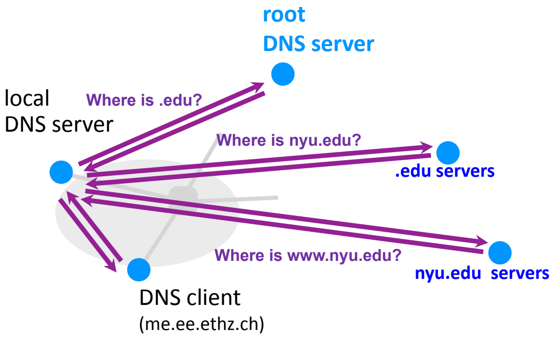






When performing a iterative query, the server
only returns the address of the next server to query





To reduce resolution times,
DNS relies on caching

DNS servers cache responses to former queries
and your client *and* the applications (!)

Authoritative servers associate a lifetime to each record
Time-To-Live (TTL)

DNS records can only be cached for TTL seconds
after which they must be cleared

As top-level servers rarely change & popular website visited often, caching is very effective (*)

Top 10% of names account for 70% of lookups

9% of lookups are unique
Limit cache hit rate to 91%

Practical cache hit rates ~75%

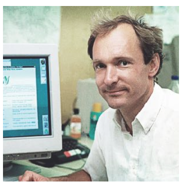
(*) see https://pdos.csail.mit.edu/papers/dns:ton.pdf

DNS

Web

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 Servers Proxies	Objects files, pictures, videos, … *organized* in Web sites a collection of objects	URL: name content HTTP: transport content

---

We'll focus on
its implementation

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

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

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 <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

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

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

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 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=g6lgURTq_BG4hSTFhEy1gTVFmSncQVsy
TJI260B3xyiXqy2wxD2YeHq1bBlwFyLoJhSc7jmcA
6TlFIBY7-
dW5lhjiRiQmY1JxT8hGCOtnLjfCL0mYcBBkpk8X4
NwAO28; expires=Mon, 31-Oct-2016 14:10:30
GMT; path=/; domain=.google.ch; HttpOnly
```

http://
Protocol

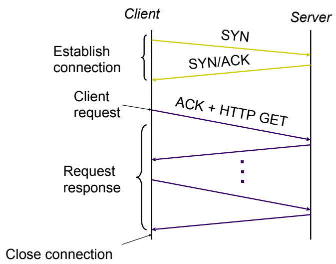
SPEED LIMIT 80
Performance

Performance goals vary depending on who you ask

	User	Network operators	Content provider
		← →	NETFLIX
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



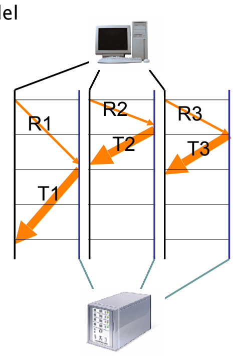
Most Web pages have multiple objects, naive HTTP opens one TCP connection for each…

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

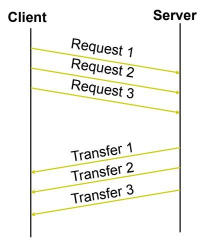
Avoid overhead of connection set-up and teardown
clients or servers can tear down the connection

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

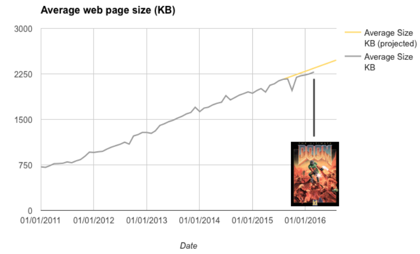
Considering the time to retrieve *n* big objects, there is no clear winners as bandwidth matters more

RTTS

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

Today, the average webpage size is 2.3 MB

as much as the original DOOM game…

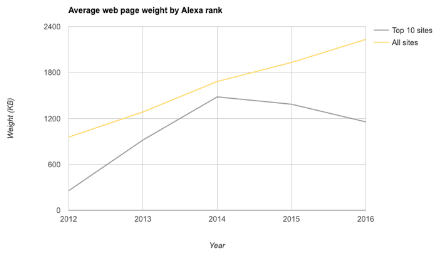


(*) see https://mobiforge.com/research-analysis/the-web-is-doom

---

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

NETFLIX

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

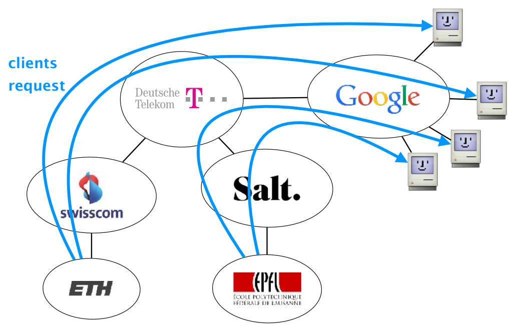
Server compares this against "last modified" time
of the resource and returns:

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

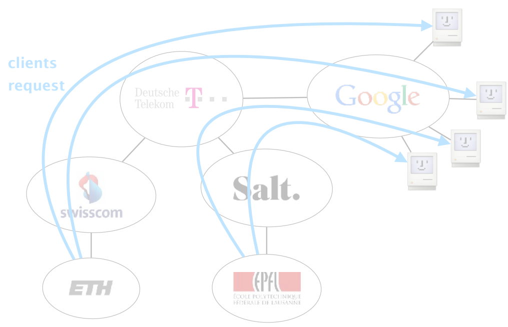
Caching can and is performed at different locations

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

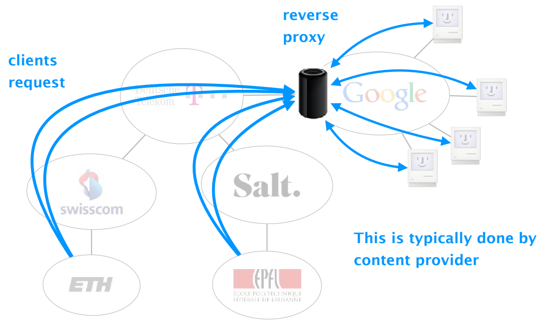
Many clients request the same information



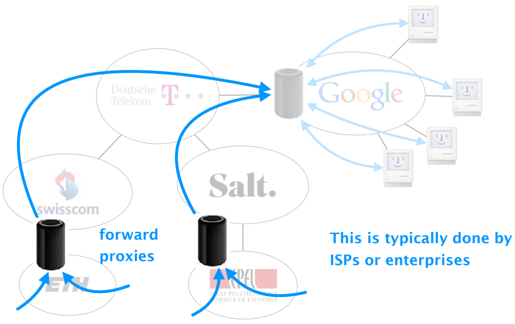
This increases servers and network's load, while clients experience unnecessary delays



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

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

Communication Networks

Spring 2018

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

May 14 2018