Communication Networks

Prof. Laurent Vanbever

Communication Networks
Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

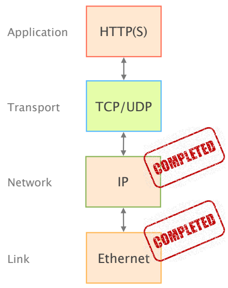
ETH Zürich (D-ITET)

April 15 2019

Materials inspired from Scott Shenker & Jennifer Rexford

Last week on
Communication Networks

We started to look at the transport layer



What Problems Should Be Solved Here?

Data delivering, to the *correct* application
- IP just points towards next protocol
- *Transport needs to demultiplex incoming data (ports)*

Files or bytestreams abstractions for the applications
- Network deals with packets
- *Transport layer needs to translate between them*

Reliable transfer (if needed)

Not overloading the receiver

Not overloading the network

What Is Needed to Address These?

Demultiplexing: identifier for application process
- Going from host-to-host (IP) to process-to-process

Translating between bytestreams and packets:
- Do segmentation and reassembly

Reliability: ACKs and all that stuff

Corruption: Checksum

Not overloading receiver: "Flow Control"
- Limit data in receiver's buffer

Not overloading network: "Congestion Control"

UDP: User Datagram Protocol

Lightweight communication between processes
- Avoid overhead and delays of ordered, reliable delivery
- Send messages to and receive them from a socket

UDP described in RFC 768 – (1980!)
- IP plus port numbers to support (de)multiplexing
- Optional error checking on the packet contents
  - (checksum field = 0 means "don't verify checksum")

SRC port	DST port
checksum	length
DATA	

Transmission Control Protocol (TCP)

Reliable, in-order delivery
- Ensures byte stream (eventually) arrives intact
  - In the presence of corruption and loss

Connection oriented
- Explicit set-up and tear-down of TCP session

Full duplex stream-of-bytes service
- Sends and receives a stream of bytes, not messages

Flow control
- Ensures that sender doesn't overwhelm receiver

Congestion control
- Dynamic adaptation to network path's capacity

TCP Header

Source port			Destination port
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			
Data			

This week on
Communication Networks

Congestion Control

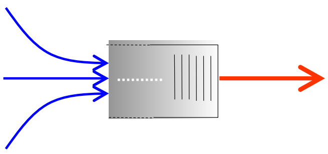
DNS

google.ch ⟷ 172.217.16.131

Congestion Control

DNS

Because of traffic burstiness and lack of BW reservation,
congestion is inevitable



If many packets arrive within
a short period of time
the node cannot keep up anymore

Congestion is harmful

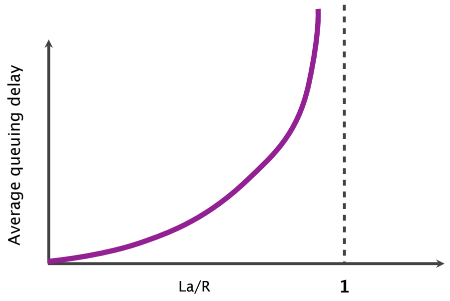
average packet arrival rate	*a*	[packet/sec]
transmission rate of outgoing link	*R*	[bit/sec]
fixed packets length	*L*	[bit
average bits arrival rate	*La*	[bit/sec]
traffic intensity	*La/R*	

When the traffic intensity is >1, the queue will increase without bound, and so does the queuing delay

Golden rule

Design your queuing system,
so that it operates far from that point

When the traffic intensity is <=1,
queueing delay depends on the burst size



Congestion is not a new problem

The Internet almost died of congestion in 1986
throughput collapsed from 32 Kbps to… 40 bps

Van Jacobson saved us with Congestion Control
his solution went right into BSD

Recent resurgence of research interest after brief lag
new methods (ML), context (Data centers), requirements

The Internet almost died of congestion in 1986
throughput collapsed from 32 Kbps to… 40 bps

original behavior	On connection, nodes send full window of packets
	Upon timer expiration, retransmit packet immediately
meaning	sending rate only limited by flow control
net effect	window-sized burst of packets

Increase in network load results in
a decrease of useful work done

Sudden load increased the round-trip time (RTT)
faster than the hosts' measurements of it

As RTT exceeds the maximum retransmission interval,
hosts begin to retransmit packets

Hosts are sending each packet several times,
eventually some copies arrive at the destination.

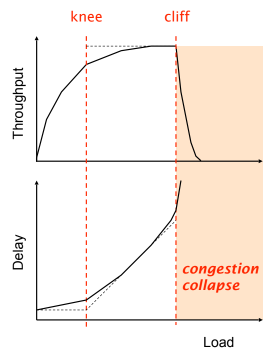
This phenomenon is known as congestion collapse

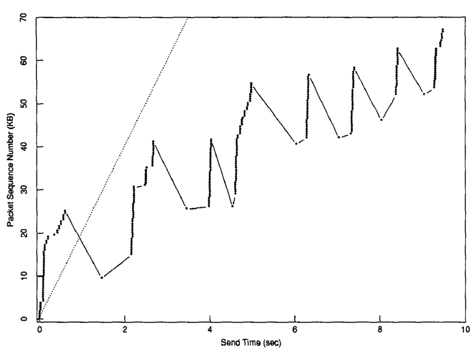
Knee point after which

throughput	increases	slowly
delay	increases	quickly

Cliff point after which

throughput	decreases	quickly
delay	tends to	infinity





Van Jacobson saved us with Congestion Control
his solution went right into BSD

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

TCP solves both using two distinct windows

Flow control	prevents one fast sender from overloading a slow receiver solved using a **receiving window**
Congestion control	prevents a set of senders from overloading the network solved using a **"congestion" window**

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

There are essentially three ways to detect congestion

Approach #1	Network could tell the source but signal itself could be lost
Approach #2	Measure packet delay but signal is noisy delay often varies considerably
Approach #3	Measure packet loss fail-safe signal that TCP already has to detect

Packet dropping is the best solution

delay- and signaling-based methods are hard & risky

Approach #3	Measure packet loss fail-safe signal that TCP already has to detect

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…

Remember that 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

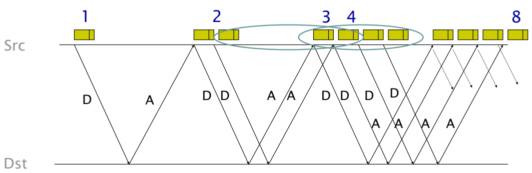
#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps…

The goal here is 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

This increase phase, known as slow start, corresponds to an… exponential increase of CWND!



slow start is called like this only because of starting point

The problem with slow start is that it can result in a full window of packet losses

Example — Assume that CWND is just enough to "fill the pipe"
After one RTT, CWND has doubled
All the excess packets are now dropped

Solution — We need a more gentle adjustment algorithm once we have a rough estimate of the bandwidth

#2 bandwidth adaptation

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

The goal here is to track the available bandwidth, and oscillate around its current value

Two possible variations

- Multiplicative Increase or Decrease
  cwnd = a * cwnd
- Additive Increase or Decrease
  cwnd = b + cwnd

… leading to four alternative design

---

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

To select one scheme, we need to consider the 3rd problem: fairness

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

#3 fairness

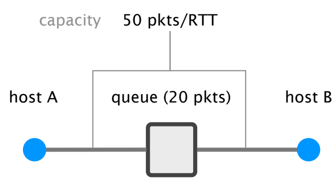
How to share bandwidth "fairly" among flows, without overloading the network

---

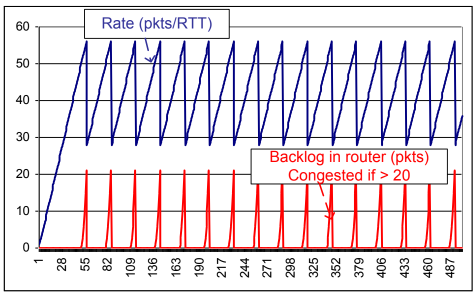
TCP notion of fairness: 2 identical flows should end up with the same bandwidth

---

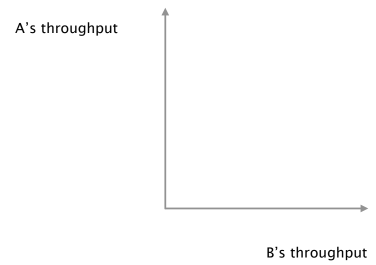
Consider first a single flow between A and B and AIMD



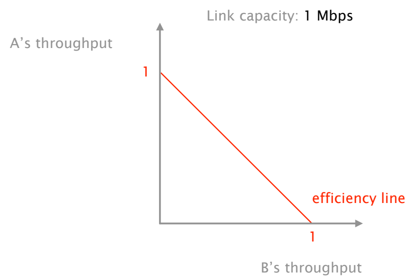
without	congestion	CWND increases by one packet every ACK
upon	congestion	CWND decreases by a factor 2



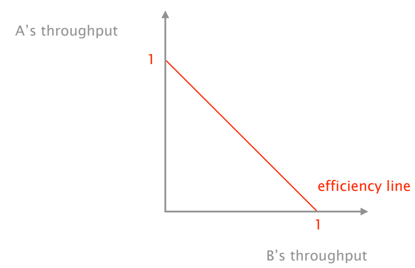
We can analyze the system behavior
using a system trajectory plot

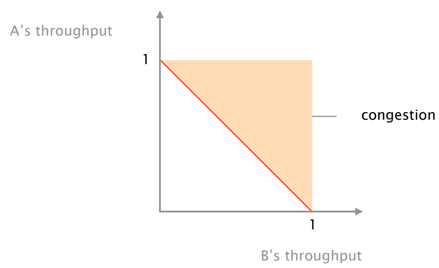


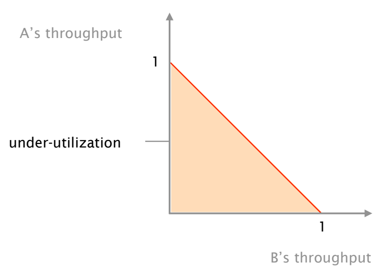
The system is efficient if the capacity is fully used,
defining an efficiency line where $a + b = 1$



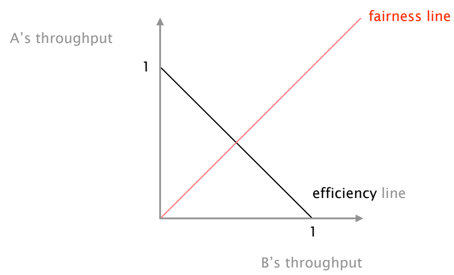
The goal of congestion control is to bring the system
as close as possible to this line, and stay there

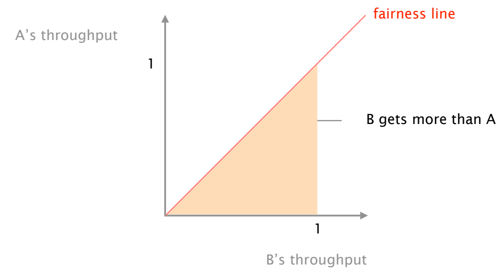


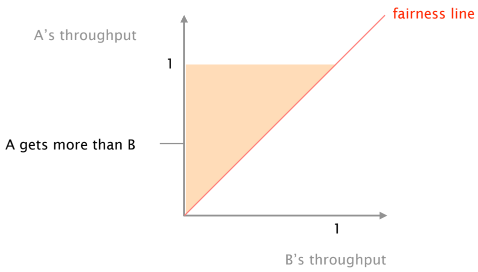


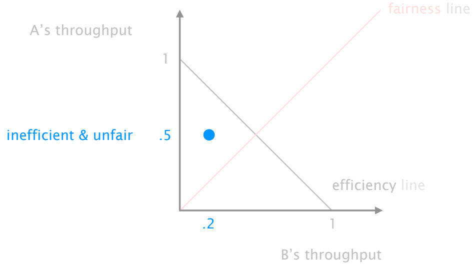


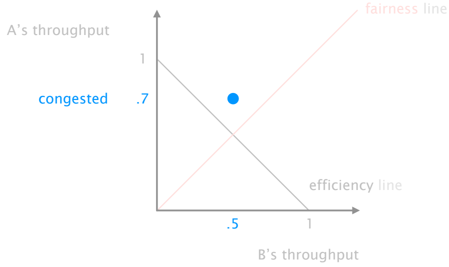
The system is fair whenever A and B have equal
throughput, defining a fairness line where $a = b$

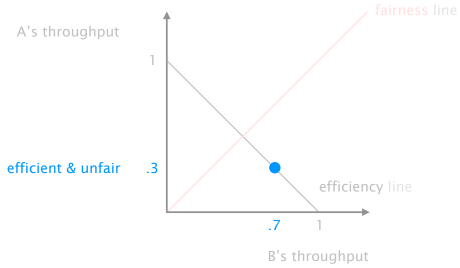


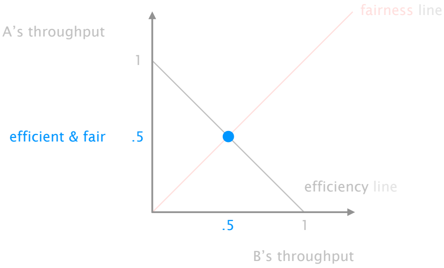






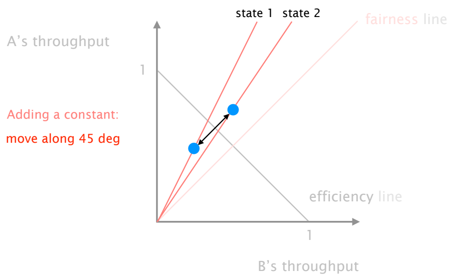




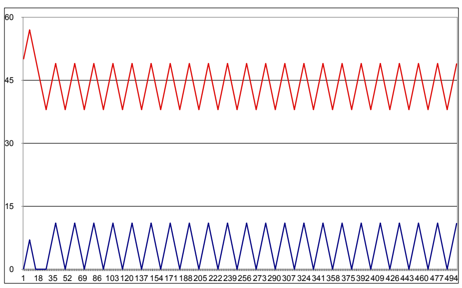


	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

AIAD does not converge to fairness, nor efficiency:
the system fluctuates between two fairness states

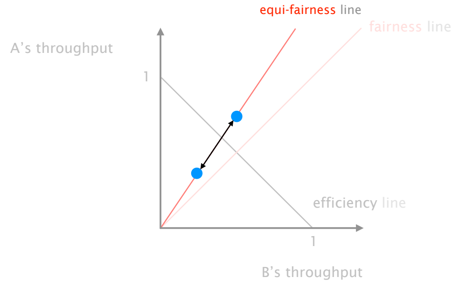


AIAD does not converge to fairness, nor efficiency:
the system fluctuates between two fairness states



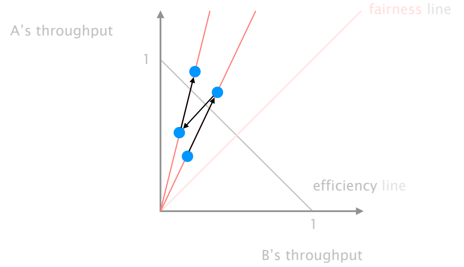
	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

MIMD does not converge to fairness, nor efficiency:
the system fluctuates along a equi-fairness line

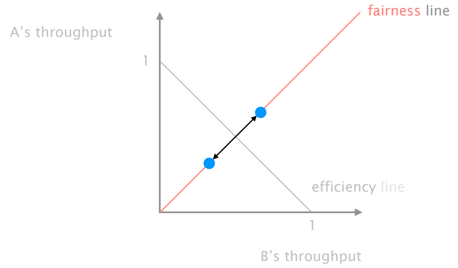


	increase behavior	decrease behavior
AIAD	gentle	gentle
AIMD	gentle	aggressive
MIAD	aggressive	gentle
MIMD	aggressive	aggressive

MIAD converges to a totally unfair allocation,
favoring the flow with a greater rate at the beginning

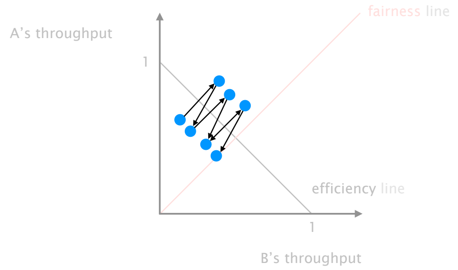


If flows start along the fairness line, MIAD fluctuates along it, yet deviating from it at the slightest change



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

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



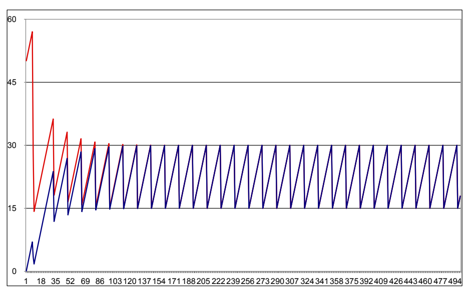
AIMD converge to fairness and efficiency,
it then fluctuates around the optimum (in a stable way)

Intuition

During increase,
both flows gain bandwidth at the same rate

During decrease,
the faster flow releases more

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



In practice, TCP implements AIMD

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

In practice, TCP implements AIMD

Implementation

After each ACK,

Increment cwnd by 1/cwnd

linear increase of max. 1 per RTT

Question

When does a sender leave slow-start and start AIMD?

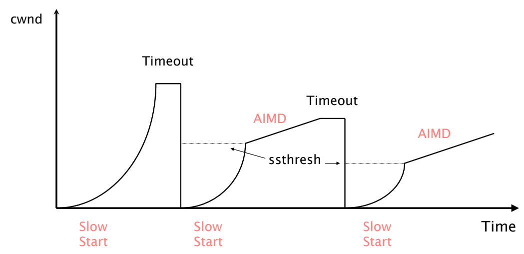
Introduce a slow start treshold, adapt it in function of congestion:

on timeout, sstresh = CNWD/2

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)

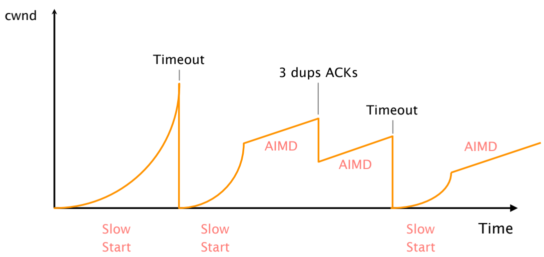
Initially:
cwnd = 1
ssthresh = infinite
New ACK received:
if (cwnd < ssthresh):
/* Slow Start*/
cwnd = cwnd + 1
else:
/* Congestion Avoidance */
cwnd = cwnd + 1/cwnd
dup_ack = 0
Timeout:
/* Multiplicative decrease */
ssthresh = cwnd/2
cwnd = 1

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

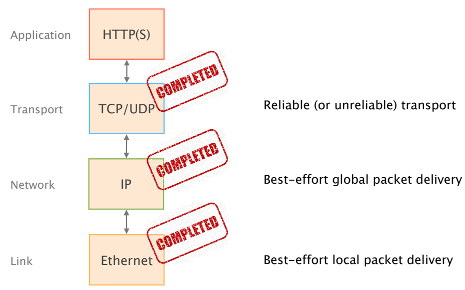
Initially:
cwnd = 1
ssthresh = infinite
New ACK received:
if (cwnd < ssthresh):
/* Slow Start*/
cwnd = cwnd + 1
else:
/* Congestion Avoidance */
cwnd = cwnd + 1/cwnd
dup_ack = 0
Timeout:
/* Multiplicative decrease */
ssthresh = cwnd/2
cwnd = 1

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 (!)



Congestion Control

DNS

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.netflix.com		52.31.246.79	(load-balancing)
		52.49.6.246	
		52.50.212.245	+5 more!

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.route-aggregation.net		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

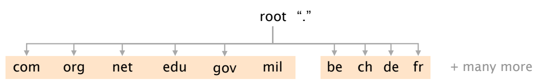
To scale, DNS adopt three intertwined hierarchies

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

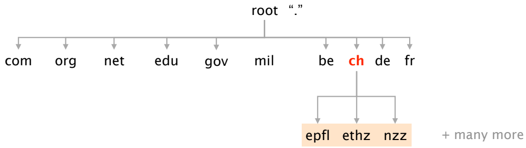
naming structure — addresses are hierarchical

https://www.ee.ethz.ch/de/departement/

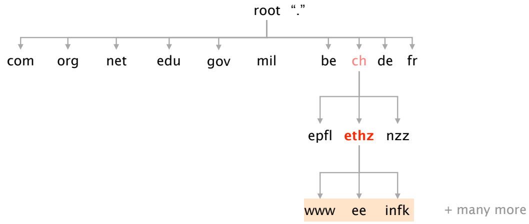
Top Level Domain (TLDs) sit at the top



Domains are subtrees



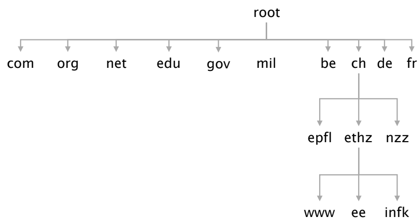
A name, *e.g.* ee.ethz.ch, represents
a leaf-to-root path in the hierarchy

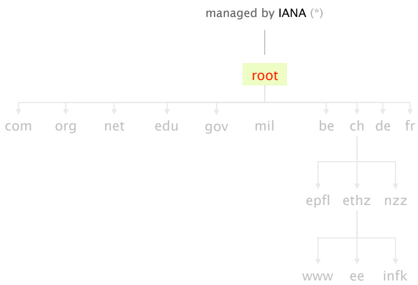


management

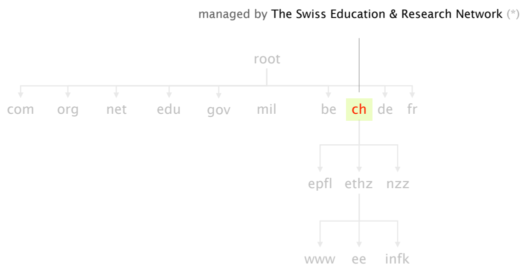
hierarchy of authority
over names

The DNS system is
hierarchically administered

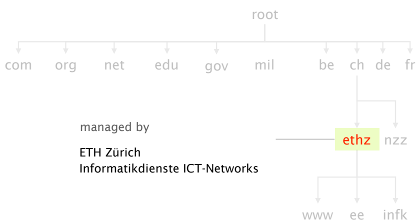




(*) see http://www.iana.org/domains/root/db



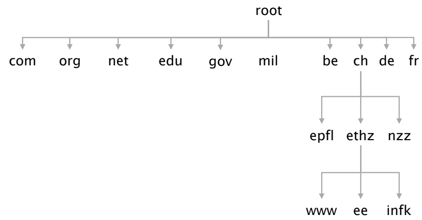
(*) see https://www.switch.ch/about/id/



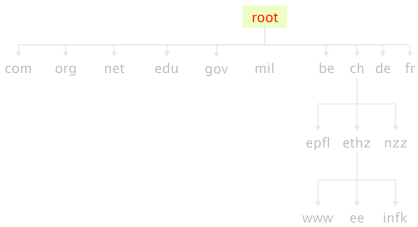
Hierarchical administration means
that name collision is trivially avoided

infrastructure   hierarchy of DNS servers

The DNS infrastructure is hierarchically organized

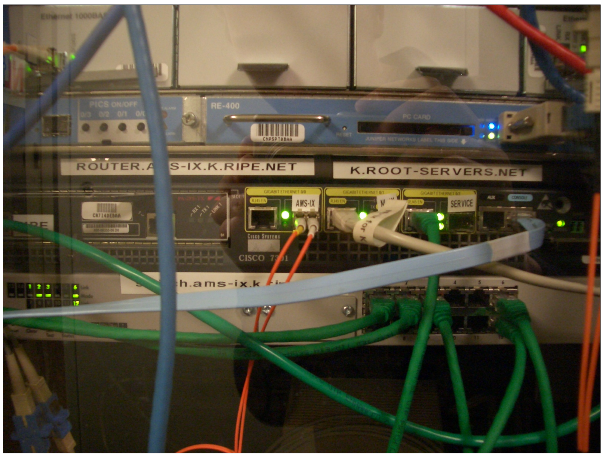


13 root servers (managed professionally) serve as root (*)



(*) 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



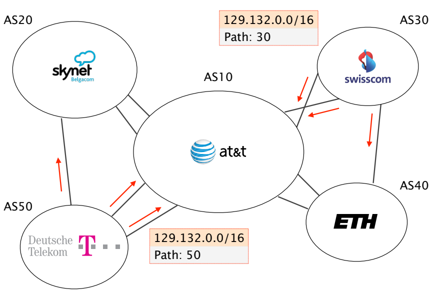
To scale root servers, operators rely on BGP anycast

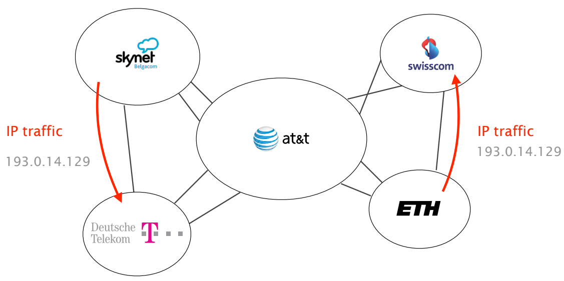
Intuition

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



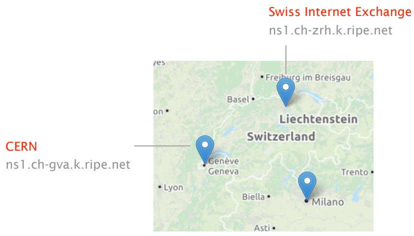


Do you see any problems in performing load-balancing this way?

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

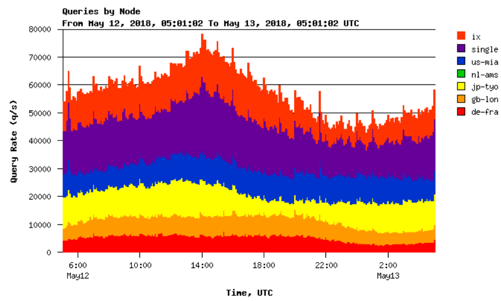


All locations announce 193.0.14.0/23 in BGP, with 193.0.14.129 being the IP of the server

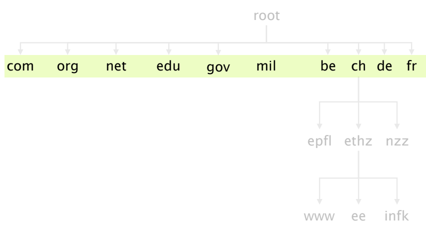
Two of these locations are in Switzerland: in Zürich and in Geneva

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

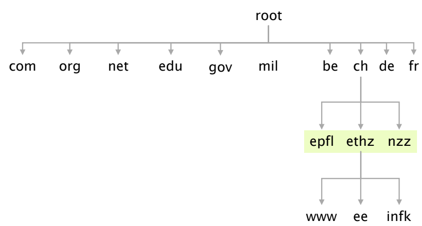
Each instance receives up to 70k queries per second summing up to more than 4 billions queries per day



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



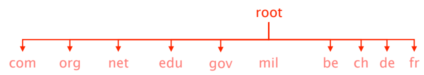
The bottom (and bulk) of the hierarchy is managed by Internet Service Provider or locally



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

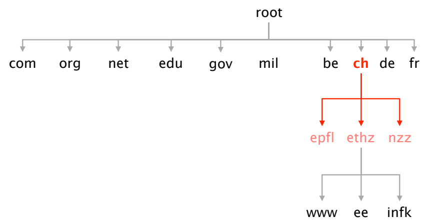



```
lvanbever:~$ dig @a.root-servers.net ch.

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

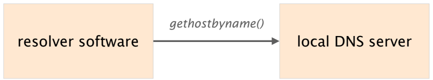
You've founded next-startup.ch and want to host it
yourself, how do you insert it into the DNS?

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 (*)

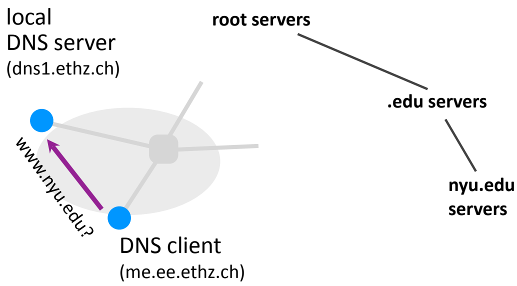
(*) see Book (Section 5)

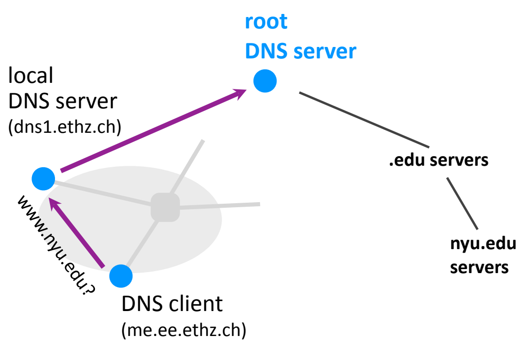
A DNS server stores Resource Records
composed of a (name, value, type, TTL)

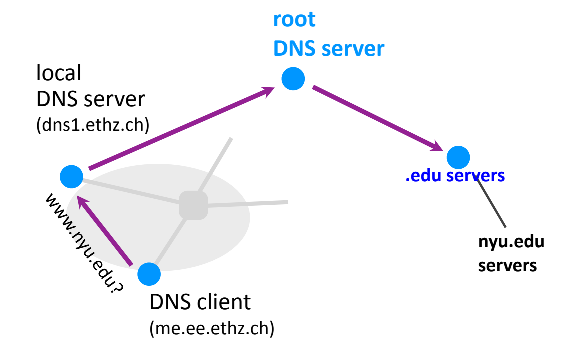
Records	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

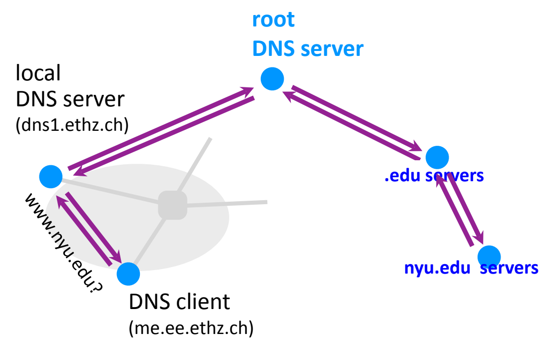
DNS resolution can either be
recursive or iterative

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

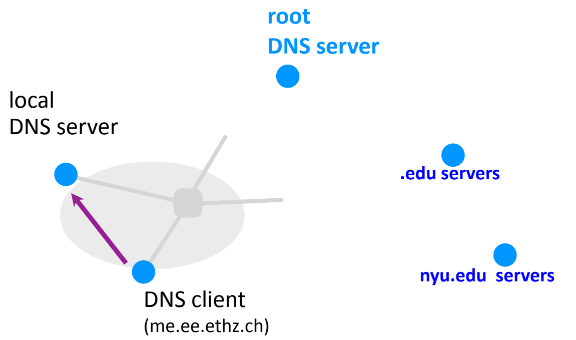


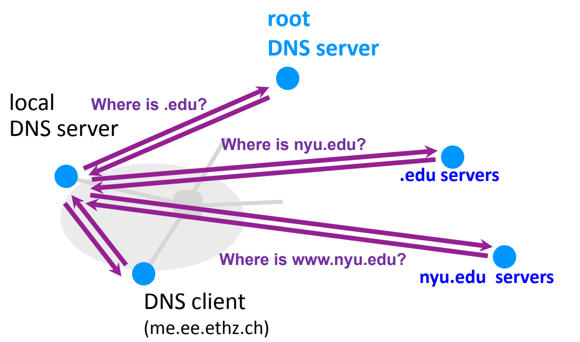






When performing a iterative query, the server only returns the address of the next server to query





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

Communication Networks
Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
April 15 2019