Communication Networks

Prof. Laurent Vanbever

Online/COVID-19 Edition

Communication Networks

Spring 2020

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

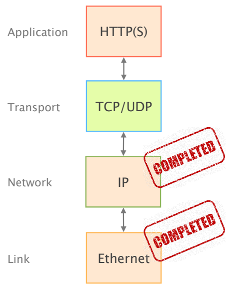
April 27 2020

Materials inspired from Scott Shenker & Jennifer Rexford

Last week on

Communication Networks

We continued our journey up the layers,
and 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

UDP: Datagram messaging service

UDP provides a connectionless, unreliable transport service

- No-frills extension of "best-effort" IP
- UDP provides only two services to the App layer
  - Multiplexing/Demultiplexing among processes
  - Discarding corrupted packets (optional)

TCP: Reliable, in-order delivery

TCP provides a connection-oriented, reliable, bytestream transport service

What UDP provides, plus:

- Retransmission of lost and corrupted packets
- Flow control (to not overflow receiver)
- Congestion control (to not overload network)
- "Connection" set-up & tear-down

Sockets

A socket is a software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system

- socketID = socket(…, socket.TYPE)
- socketID.sendto(message, …)
- socketID.recvfrom(…)

Two important types of sockets

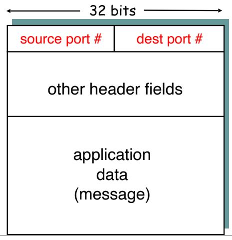
- UDP socket: TYPE is SOCK_DGRAM
- TCP socket: TYPE is SOCK_STREAM

Multiplexing and Demultiplexing

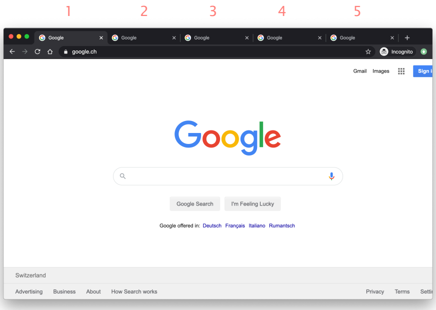
Host receives IP datagrams

- Each datagram has source and destination IP address,
- Each segment has source and destination port number

Host uses IP addresses *and* port numbers to direct the segment to appropriate socket



A TCP/UDP socket is identified by a 4-tuple:
(src IP, src port, dst IP, dest port)



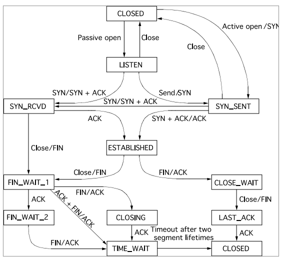
Let's say you open 5 tabs to google.ch

Your IP: 129.132.19.1 Google's IP: 172.217.168.3

Client OS		src IP	src port	dest IP	dest port
socket	1	129.132.19.1	54001	172.217.168.3	443
	2	129.132.19.1	55240	172.217.168.3	443
	3	129.132.19.1	48472	172.217.168.3	443
	4	129.132.19.1	35456	172.217.168.3	443
	5	129.132.19.1	42001	172.217.168.3	443

Server OS		src IP	src port	dest IP	dest port
socket	1	172.217.168.3	443	129.132.19.1	54001
	2	172.217.168.3	443	129.132.19.1	55240
	3	172.217.168.3	443	129.132.19.1	48472
	4	172.217.168.3	443	129.132.19.1	35456
	5	172.217.168.3	443	129.132.19.1	42001

The life of a TCP connection is a sequence of states, described with a Finite State Machine



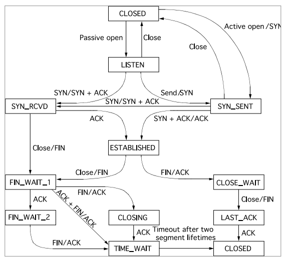
TCP connections start/end in the CLOSED state

Most of states relate to

- the connection establishment (three-way handshake)
- the connection termination (ensuring reliability)

Data is exchanged in the ESTABLISHED state

The TCP connection moves from one state to another in response of events (timeouts, "flagged" segments, …)



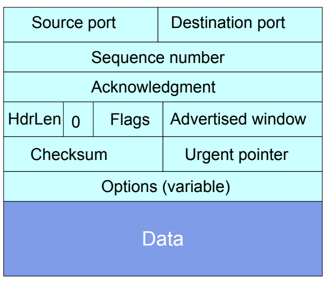
TCP connections start/end in the CLOSED state

Most of states relate to

- the connection establishment (three-way handshake)
- the connection termination (ensuring reliability)

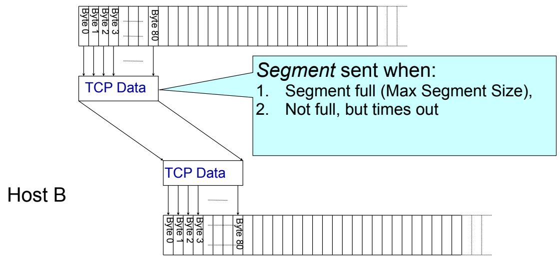
Data is exchanged in the ESTABLISHED state

TCP Header



Segments and Sequence Numbers

TCP “Stream of Bytes” Service… Provided Using TCP “Segments”



ACKing and Sequence Numbers

Sender sends packet

- Data starts with sequence number X
- Packet contains B bytes
  - X, X+1, X+2, ….X+B-1

Upon receipt of packet, receiver sends an ACK

- If all data prior to X already received:
  - ACK acknowledges X+B (because that is next expected byte)
- If highest contiguous byte received is smaller value Y
  - ACK acknowledges Y+1
  - Even if this has been ACKed before

TCP Connection Establishment and Initial Sequence Numbers

Initial Sequence Number (ISN)

Sequence number for the very first byte

- E.g., Why not just use ISN = 0?

Practical issue

- IP addresses and port #s uniquely identify a connection
- Eventually, though, these port #s do get used again
- … small chance an old packet is still in flight

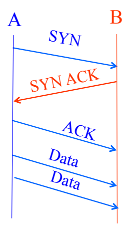
TCP therefore requires changing ISN

- initially set from 32-bit clock that ticks every 4 microseconds
- now drawn from a pseudo random number generator (security)

To establish a connection, hosts exchange ISNs

- **How does this help?**

Establishing a TCP Connection

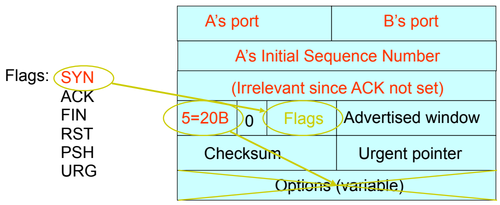


Each host tells its ISN to the other host.

Three-way handshake to establish connection

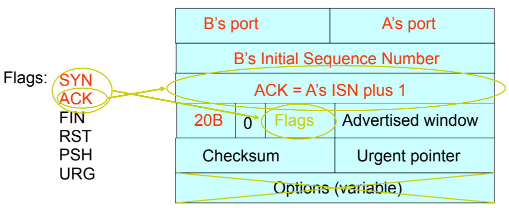
- Host A sends a **SYN** (open; “synchronize sequence numbers”)
- Host B returns a SYN acknowledgment (**SYN ACK**)
- Host A sends an **ACK** to acknowledge the SYN ACK

Step 1: A’s Initial SYN Packet



A tells B it wants to open a connection…

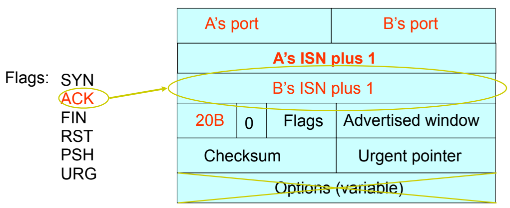
Step 2: B’s SYN-ACK Packet



B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A’s ACK of the SYN-ACK



A tells B it’s likewise okay to start sending

… upon receiving this packet, B can start sending data

This week on
Communication Networks

Congestion Control | DNS | Introduction to 2nd project

ethz.ch ⇨
129.132.19.216

reliable transport
starts *today*!

Congestion Control | DNS | Introduction to 2nd project

from slides
123 / 138 (03e_internet_udp_tcp.pdf)
16 / 88 (03f_internet_congestion_control.pdf)

Congestion Control | **DNS** | Introduction to 2nd project

ethz.ch ⇨
129.132.19.216

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		52.18.41.127	(load-balancing)
		34.248.59.175	
		52.209.236.203	+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		82.130.102.71
www.route-aggregation.net		82.130.102.71
comm-net.ethz.ch		82.130.102.71

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

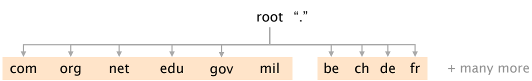
When you need… more scalability,
you add… a hierarchical structure

To scale,
DNS adopt three intertwined hierarchies

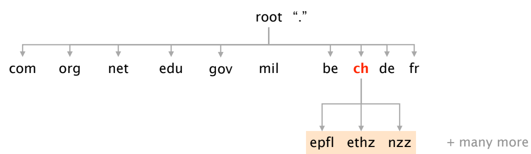
naming structure	hierarchy of addresses https://www.ee.ethz.ch/de/departement/
management	hierarchy of authority over names
infrastructure	hierarchy of DNS servers

naming structure — hierarchy of addresses
https://www.ee.ethz.ch/de/departement/

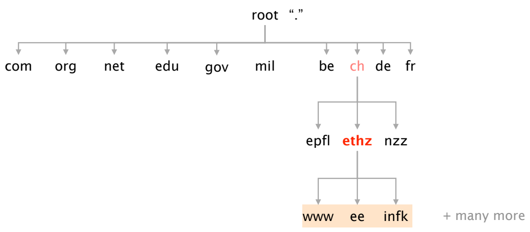
Top Level Domain (TLDs) sit at the top



Domains are subtrees



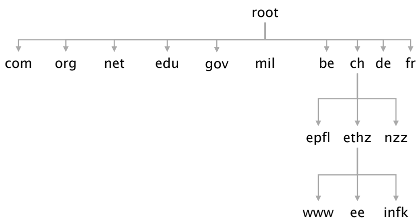
A name, *e.g.* ee.ethz.ch, represents
a leaf-to-root path in the hierarchy

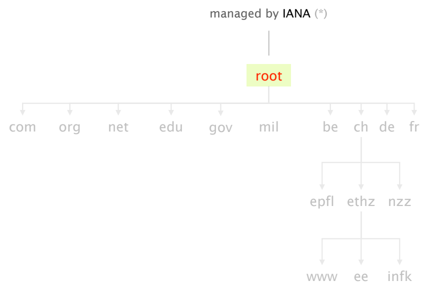


management

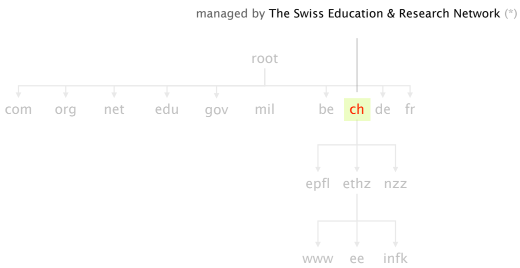
hierarchy of authority
over names

The DNS system is
hierarchically administered

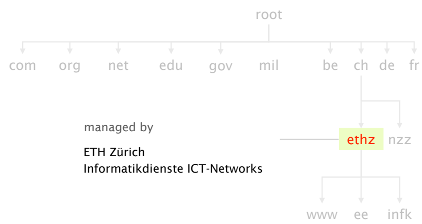




(*) see http://www.iana.org/domains/root/db



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

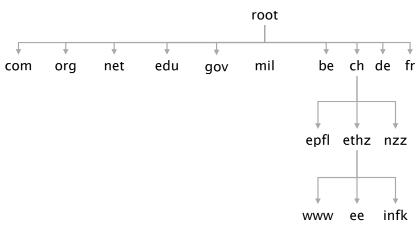


Hierarchical administration means
that name collision is trivially avoided

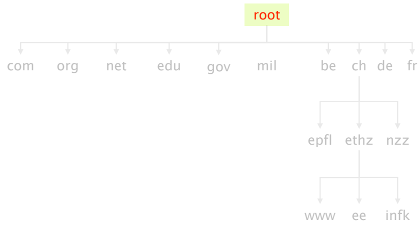
infrastructure

hierarchy of DNS servers

The DNS infrastructure is hierarchically organized

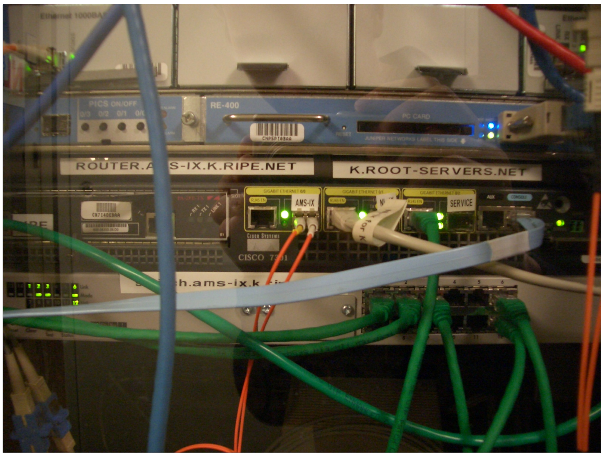


13 root servers (managed professionally) serve as root (*)



(*) 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



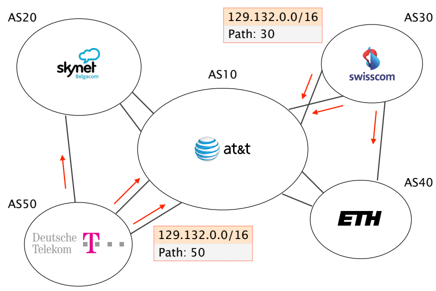
To scale root servers, operators rely on BGP anycast

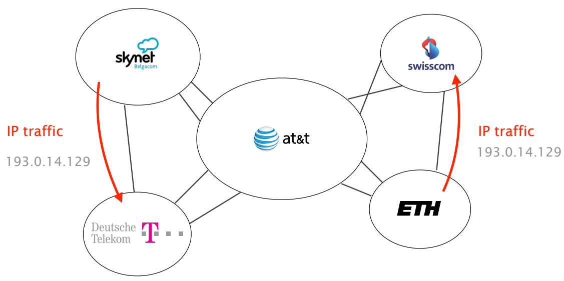
Intuition

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



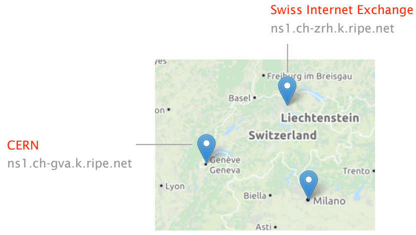


Do you see any problems in performing load-balancing this way?

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

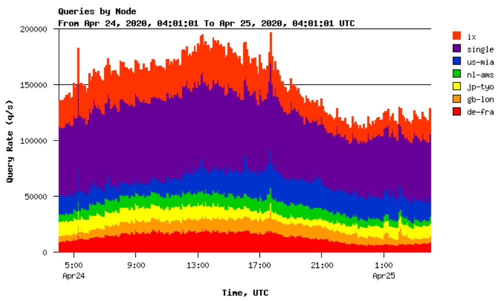


All locations announce 193.0.14.0/23 in BGP, with 193.0.14.129 being the IP of the server

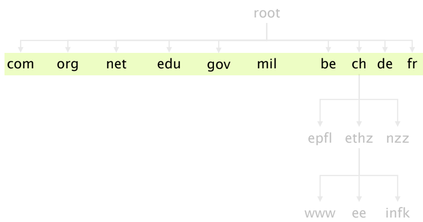
Two of these locations are in Switzerland: in Zürich and in Geneva

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

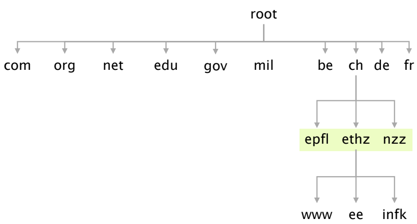
Each instance receives up to 70k queries per second summing up to more than 4 billions queries per day



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



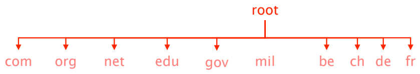
The bottom (and bulk) of the hierarchy is managed by Internet Service Provider or locally



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



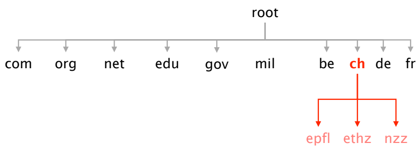

```
lvanbever:~$ dig @a.root-servers.net ch.

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.
```

also see https://www.iana.org/domains/root/db/ch.html

If you want to learn more on ".ch" take a look at SWITCH's annual report

https://www.nic.ch/export/shared/.content/files/SWITCH_Report_Registry_2019.pdf

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




```
lvanbever:~$ dig @a.nic.ch ethz.ch

ethz.ch.    3600  IN NS ns2.switch.ch.
ethz.ch.    3600  IN NS ns2.ethz.ch.
ethz.ch.    3600  IN NS ns1.ethz.ch.
```

Once arrived at the leaf of the hierarchy (*.ethz.ch), each DNS server knows the IP address of all children

```
lvanbever:~$ dig @ns1.ethz.ch comm-net.ethz.ch

comm-net.ethz.ch. 3600  IN CNAME virt07.ethz.ch.
virt07.ethz.ch.     3600  IN A  82.130.102.71
```

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

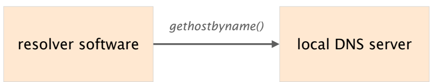
Provide *X* with the name and IP of your DNS servers
e.g., [ns1.next-startup.ch,129.132.19.253]

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

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

Using DNS relies on two components



trigger resolution process
send request to local DNS server

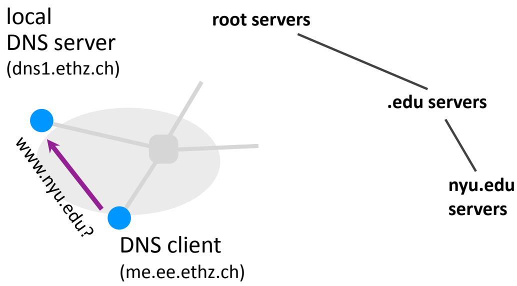
usually, near the endhosts
configured statically (resolv.conf)
or dynamically (DHCP)

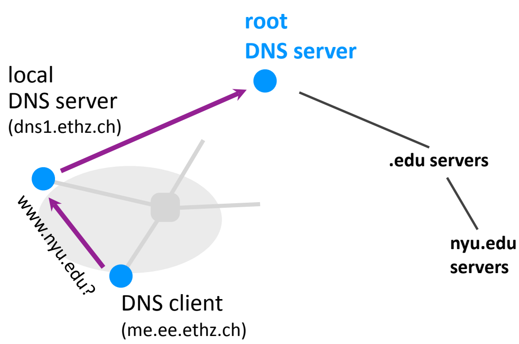
DNS query and reply uses UDP (port 53), reliability is implemented by repeating requests (*)

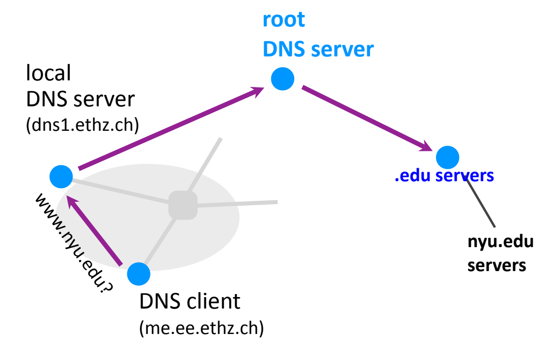
(*) see Book (Section 5)

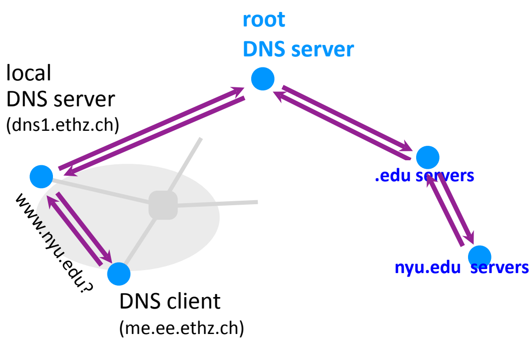
DNS resolution can either be recursive or iterative

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

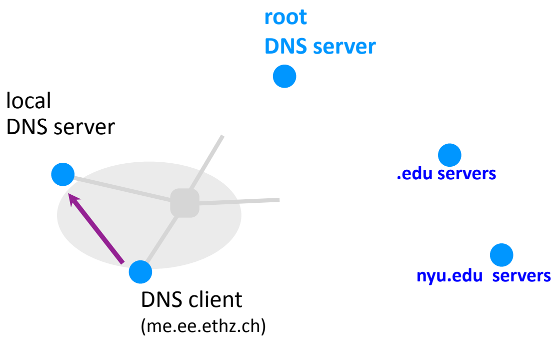


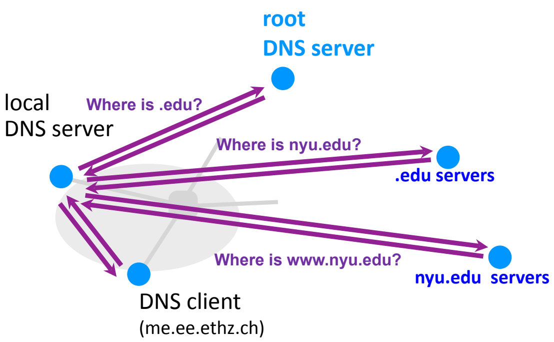






When performing a iterative query,
the server only returns the address of the "next server"





What about resolving speeds?
Waiting for servers all over the globe is not fast…

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

Congestion Control

DNS

Introduction to 2nd project

reliable transport starts *today*!

Check Tobias' slides on https://comm-net.ethz.ch

Communication Networks

Spring 2020

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

April 27 2020