NTNU – Trondheim
Norwegian University of
Science and Technology

Application Layer – DNS and P2P

TTM4200 – Computer Networks

individual Readiness Assurance Test

- Find the right answer **alone.**
- Choose the answer that fits **best.**
- **One** handwritten A4 page as helping material.

team Readiness Assurance Test

- Find the right answer **in teams (breakout rooms).**
- Choose the answer that fits **best.**
- **One** handwritten A4 page as helping material.

- **Discuss!**

Time for a break

we restart at 11:15

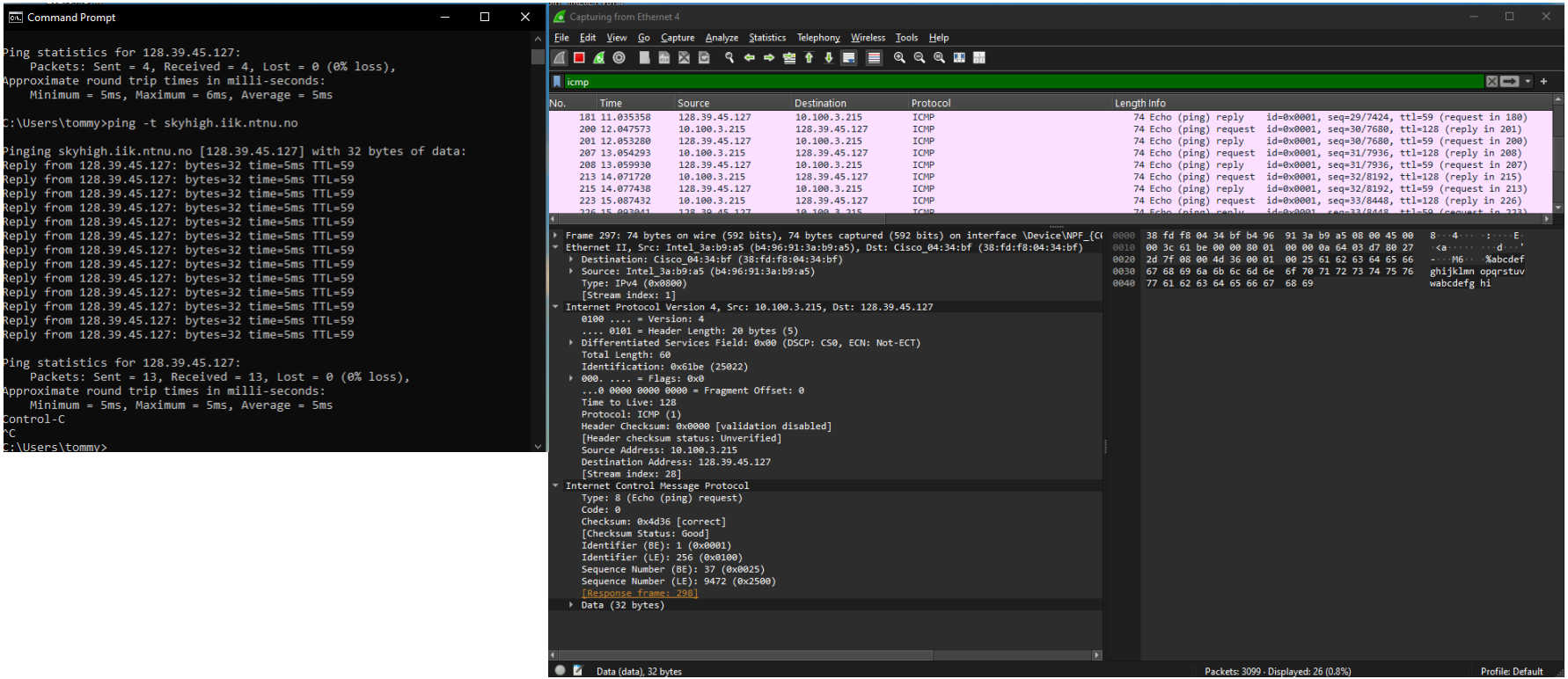
Windows users might consider to install WSL (Windows Subsystem for Linux):

- Open PowerShell or Windows Command Prompt in **administrator** mode
- enter the wsl --install command, then restart your machine

Weekly Feedback

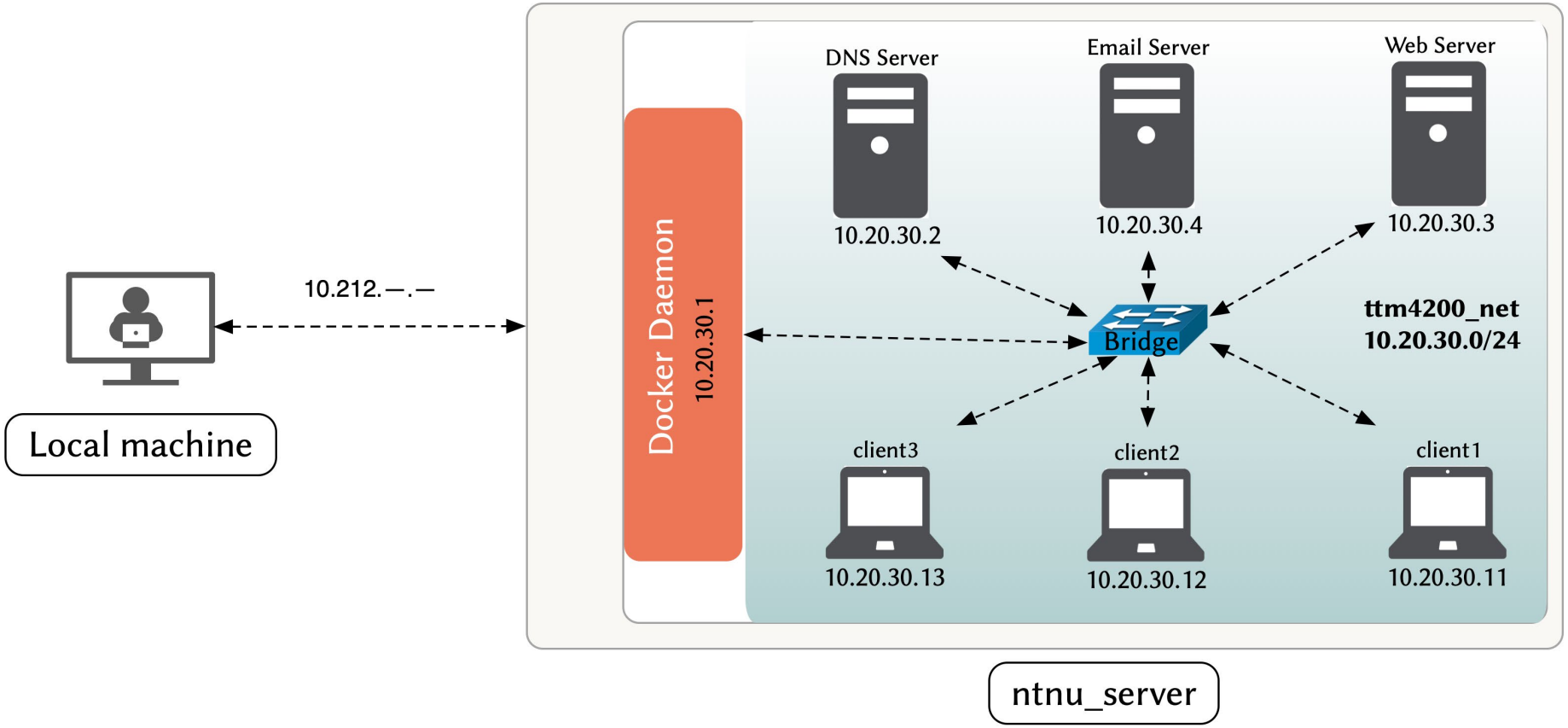
- Connection to Labs
- Learn more about
  - VMs
  - Web cache (Forward and reverse)
  - RTT: Round Trip Time
  - Queuing
  - Link to Cybersecurity: https://krebsonsecurity.com/2025/01/mastercard-dns-error-went-unnoticed-for-years/

RTT


Command Prompt
Ping statistics for 128.39.45.127:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 5ms, Maximum = 6ms, Average = 5ms
C:\Users\tommy>ping -t skyhigh.iik.ntnu.no
Pinging skyhigh.iik.ntnu.no [128.39.45.127] with 32 bytes of data:
Reply from 128.39.45.127: bytes=32 time=5ms TTL=59
Ping statistics for 128.39.45.127:
Packets: Sent = 13, Received = 13, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 5ms, Maximum = 5ms, Average = 5ms
Control-C
^C
C:\Users\tommy>
Capturing from Ethernet 4
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
No. Time Source Destination Protocol Length Info
181 11.035358 128.39.45.127 10.100.3.215 ICMP 74 Echo (ping) reply id=0x0001, seq=29/7424, ttl=59 (request in 180)
200 12.047573 10.100.3.215 128.39.45.127 ICMP 74 Echo (ping) request id=0x0001, seq=30/7680, ttl=128 (reply in 201)
201 12.053280 128.39.45.127 10.100.3.215 ICMP 74 Echo (ping) reply id=0x0001, seq=30/7680, ttl=59 (request in 200)
207 13.054293 10.100.3.215 128.39.45.127 ICMP 74 Echo (ping) request id=0x0001, seq=31/7936, ttl=128 (reply in 208)
208 13.059930 128.39.45.127 10.100.3.215 ICMP 74 Echo (ping) reply id=0x0001, seq=31/7936, ttl=59 (request in 207)
213 14.071720 10.100.3.215 128.39.45.127 ICMP 74 Echo (ping) request id=0x0001, seq=32/8192, ttl=128 (reply in 215)
215 14.077438 128.39.45.127 10.100.3.215 ICMP 74 Echo (ping) reply id=0x0001, seq=32/8192, ttl=59 (request in 213)
223 15.087432 10.100.3.215 128.39.45.127 ICMP 74 Echo (ping) request id=0x0001, seq=33/8448, ttl=128 (reply in 226)
Frame 297: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{C
Ethernet II, Src: Intel_3a:b9:a5 (b4:96:91:3a:b9:a5), Dst: Cisco_04:34:bf (38:fd:f8:04:34:bf)
Destination: Cisco_04:34:bf (38:fd:f8:04:34:bf)
Source: Intel_3a:b9:a5 (b4:96:91:3a:b9:a5)
Type: IPv4 (0x0800)
[Stream index: 1]
Internet Protocol Version 4, Src: 10.100.3.215, Dst: 128.39.45.127
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 60
Identification: 0x61be (25022)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: ICMP (1)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.100.3.215
Destination Address: 128.39.45.127
[Stream index: 28]
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x4d36 [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 37 (0x0025)
Sequence Number (LE): 9472 (0x2500)
[Response frame: 298]
Data (32 bytes)
0000 38 fd f8 04 34 bf b4 96 91 3a b9 a5 08 00 45 00
0010 00 3c 61 be 00 00 80 01 00 00 0a 64 03 d7 80 27
0020 2d 7f 08 00 4d 36 00 01 00 25 61 62 63 64 65 66
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
0040 77 61 62 63 64 65 66 67 68 69
Data (data), 32 bytes
Packets: 3099 · Displayed: 26 (0.8%)
Profile: Default

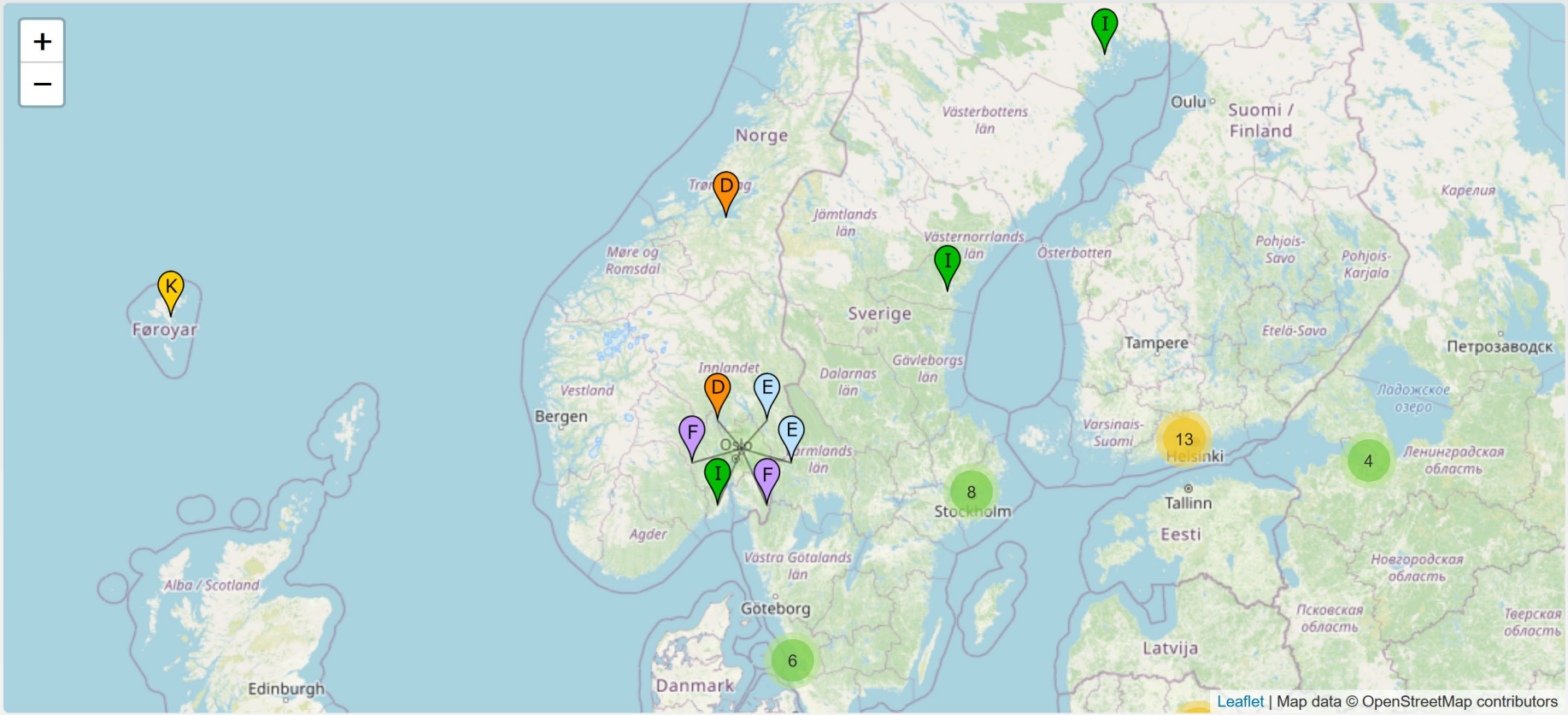
Introduction to Lab 2

- Work
- Ask questions
- Read the materials
- Search for documentation
- Try different things
- Make mistakes
- Work


DNS Server
Email Server
Web Server
10.20.30.2
10.20.30.4
10.20.30.3
Docker Daemon
10.20.30.1
Bridge
ttm4200_net
10.20.30.0/24
10.212.—.—
Local machine
client3
client2
client1
10.20.30.13
10.20.30.12
10.20.30.11
ntnu_server

As of 01/30/2023 9:35 a.m., the root server system consists of 1617 instances operated by the 12 independent root server operators.

As of 2025-01-25T18:31:00Z, the root server system consists of 1919 instances operated by the 12 independent root server operators.



As of 2025-01-25T18:31:00Z, the root server system consists of 1919 instances operated by the 12 independent root server operators.

Types of DNS Resolver

- Provided to customers/users by network provider (ISP)
- Provided to everyone by organizations or large content providers
  - ➔ Open resolvers

Image: mic.com/articles/85987/turkish-protesters-are-spray-painting-8-8-8-8-and-8-8-4-4-on-walls-here-s-what-it-means#.Ezqr7nadu

DNS records

DNS: distributed database storing resource records (RR)

RR format: (`name, value, type, ttl`)

type=A

- `name` is hostname
- `value` is IP address

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

type=MX

- `value` is name of SMTP mail server associated with `name`

dig

```
dig ntnu.no
dig -x 129.241.200.18
dig @ns1.ntnu.no ntnu.no any

Optional:
dig cloudflare.com any @ns3.cloudflare.com
(https://blog.cloudflare.com/rfc8482-saying-goodbye-to-any/)
```

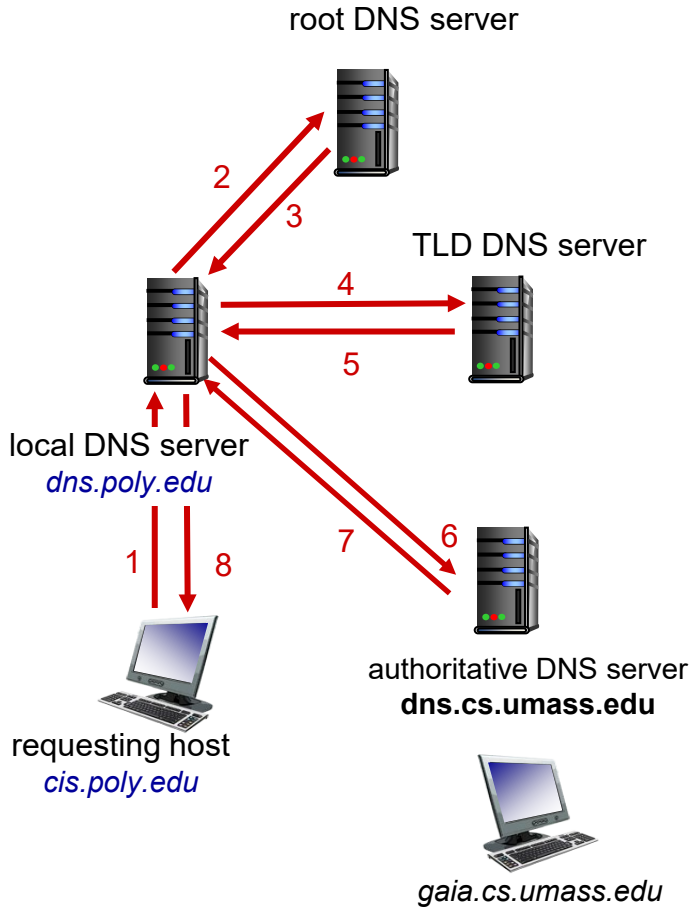
Problem IX

Assume that the RTT between a client and the local DNS server is RTT_l, while the RTT between the local DNS server and other DNS servers is RTT_r. Assume that no DNS server performs caching.

a) What is the total response time for the scenario illustrated in the figure?

b) What is the total response time for the scenario if a recursive query is used instead?

c) Assume now that the DNS record for the requested name is cached at the local DNS server. What is the total response time for the two scenarios?



Join at menti.com | use code 5458 9245

NTNU
Kunnskap for en bedre verden

Instructions

Go to

www.menti.com

Enter the code

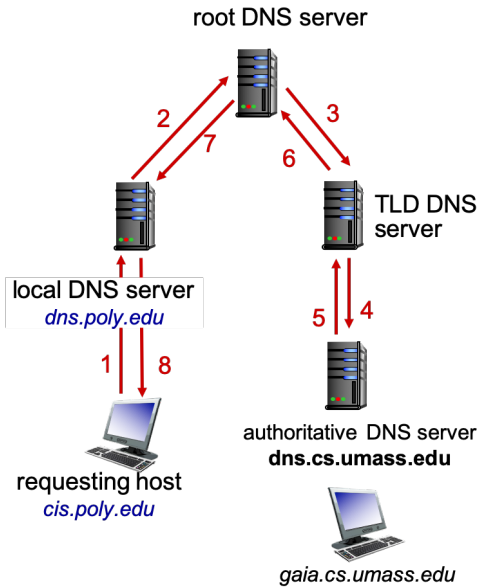
5458 9245

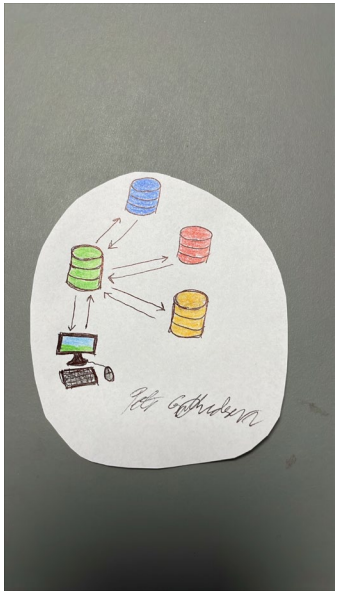
Or use QR code

Enough for today

we continue Thursday at 12:15

Recursive vs. iterative





To iterate is human,
to recurse divine.

(L. Peter Deutsch; also famous for https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing)

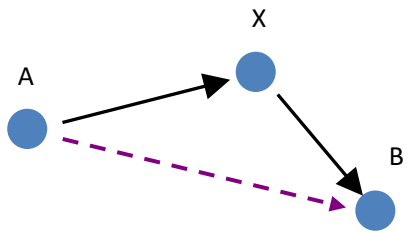
RATs

- Tough one – but excellent discussions and learning during the tRATs!

Indirection

Rather than reference an entity directly, reference it (“indirectly”) via another entity, which in turn can or will access the original entity

Every problem in computer science can be solved by adding another level of indirection"
-- Butler Lampson



Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

NETFLIX

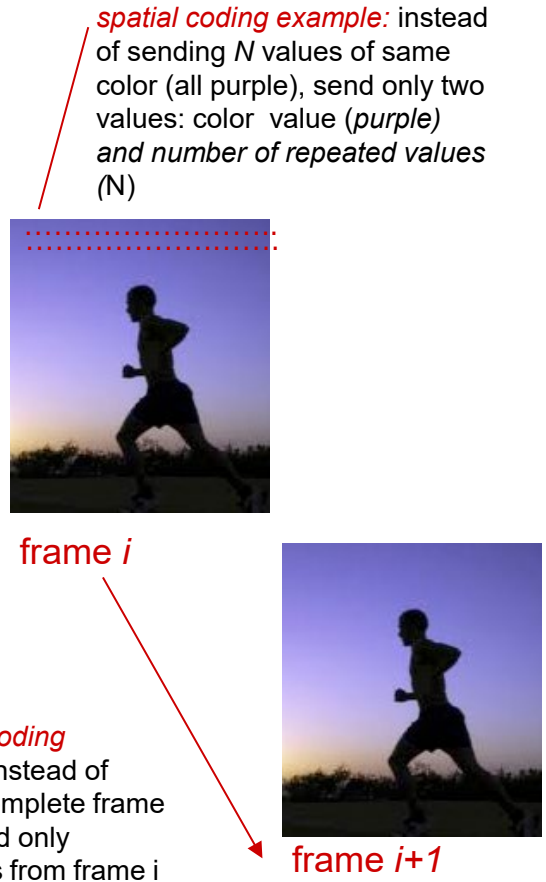
迅雷看看 www.kankan.com 网络高清影院

Akamai

YouTube

Multimedia: video

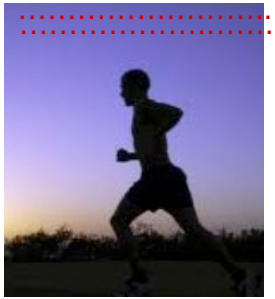
- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)



Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)
  - H.264 and H.265 (4K<->800Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



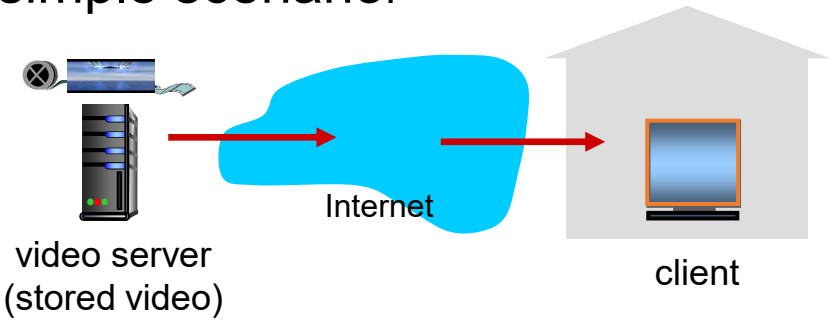
frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

Streaming stored video

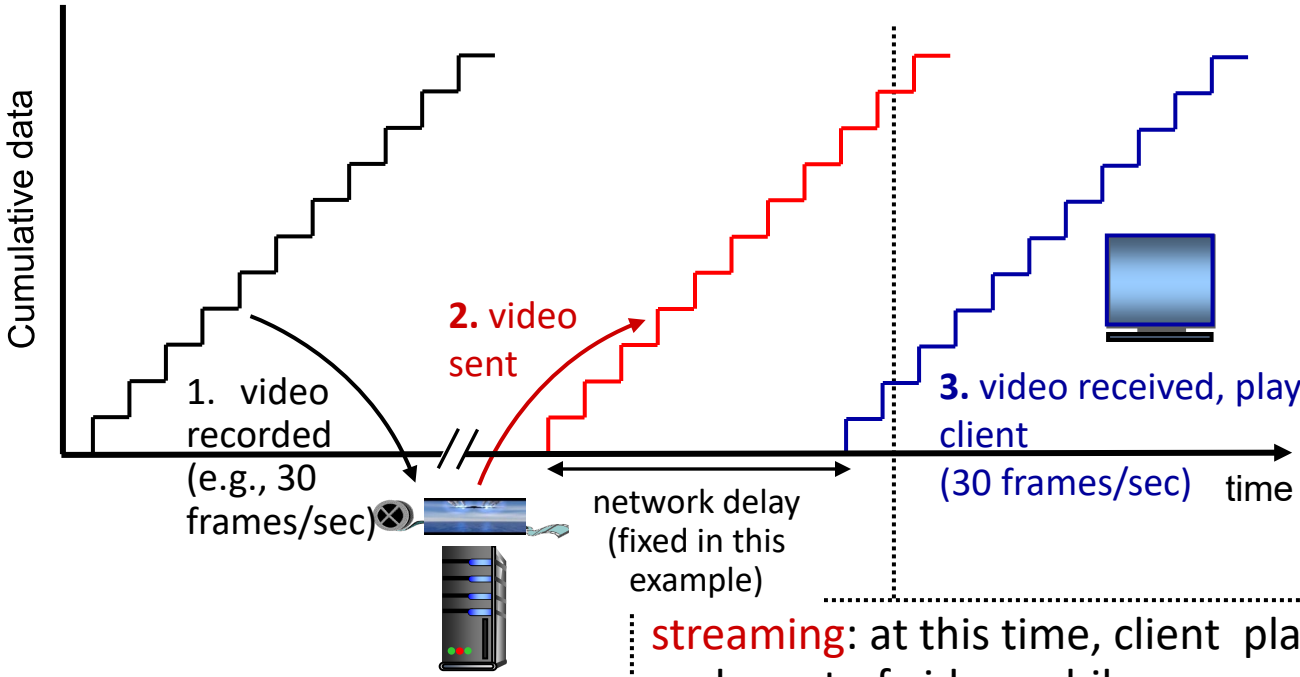
simple scenario:



Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

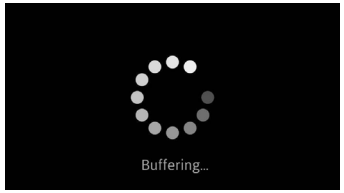
Streaming stored video



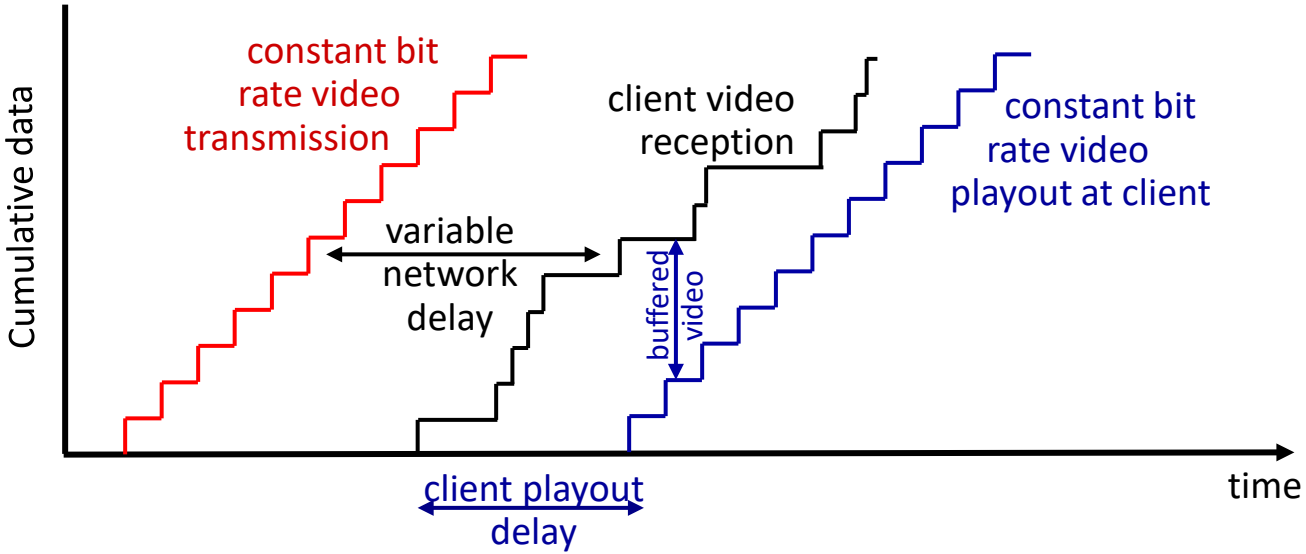
streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted



Streaming stored video: playout buffering



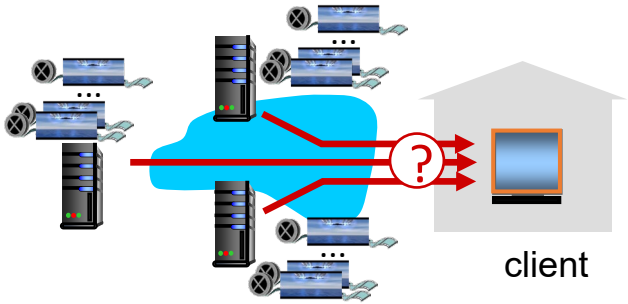
- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks

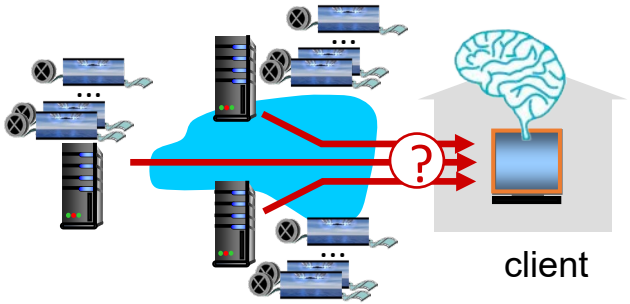


client:

- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

Streaming multimedia: DASH

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)



➔ https://howvideo.works/

Streaming video = encoding + DASH + playout buffering

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long (and possibly congested) path to distant clients

….quite simply: this solution *doesn't scale*

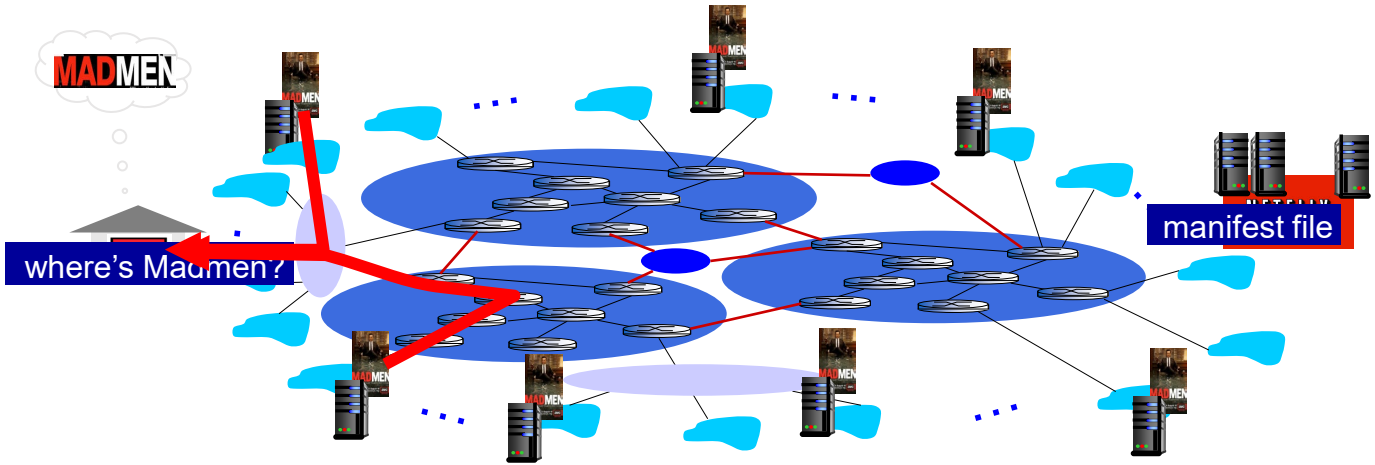
Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



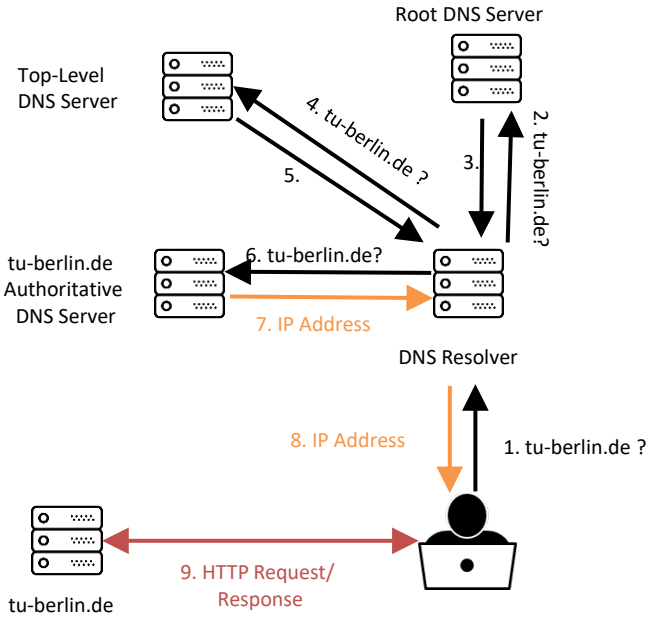
Content distribution networks (CDNs)

OTT challenges: coping with a congested Internet from the "edge"

- what content to place in which CDN node?
- from which CDN node to retrieve content? At which rate?

How do CDNs work? (Simplified)

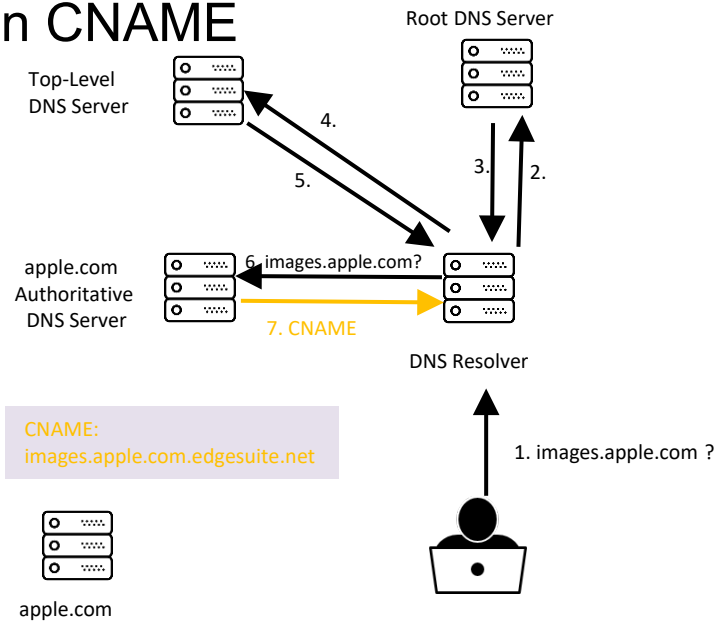
- Normal web request:
  - First resolve name recursively
  - Then HTTP request



Slide adapted from "Drafting Behind Akamai", Sigcomm 2006

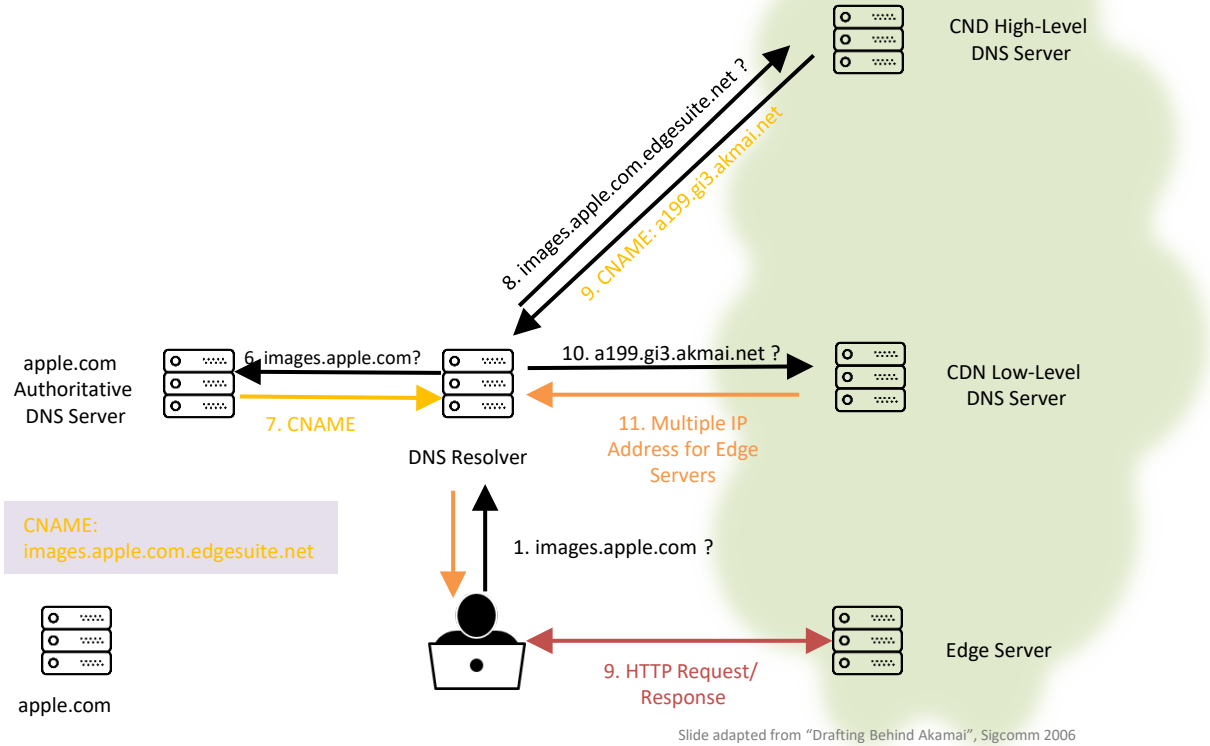
How do CDNs work? (Simplified)

- DNS request for CDN content
  - Results in CNAME



Slide adapted from "Drafting Behind Akamai", Sigcomm 2006

How do CDNs work? (Simplified)



Slide adapted from "Drafting Behind Akamai", Sigcomm 2006

- https://jvns.ca/blog/2021/12/15/mess-with-dns/
- https://messwithdns.net/

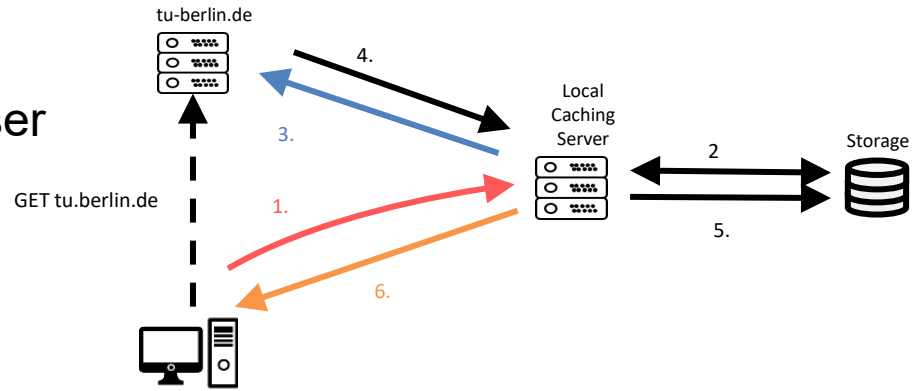
Comparison: Web Cache

1. Intercept and redirect HTTP request to caching server
2. Check if requested page is stored. If yes: Goto 6

3. & 4. If no: caching server request web page from server

5. Caching server stores received web page

6. Deliver web page to the user



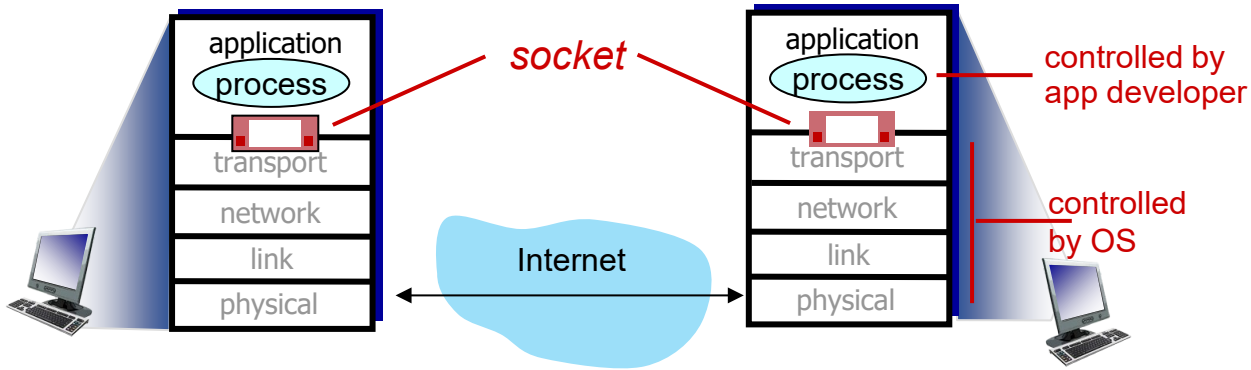
Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

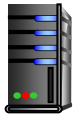
Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Client/server socket interaction: UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Copy-and-paste-friendly version of the code available at https://s.ntnu.no/iikt2105-sockets

Example App: UDP Client

Python UDPClient

include Python's socket library →
```
from socket import *
```
```
serverName = 'hostname'
serverPort = 12000
```
create UDP socket →
```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```
get user keyboard input →
```
message = input('Input lowercase sentence:')
```
attach server name, port to message; send into socket →
```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```
read reply data (bytes) from socket →
```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```
print out received string and close socket →
```
print(modifiedMessage.decode())
clientSocket.close()
```

Copy-and-paste-friendly version of the code available at https://s.ntnu.no/iikt2105-sockets

Example App: UDP Server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print('The server is ready to receive')
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

send upper case string back to this client →

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - *source* port numbers used to distinguish clients (more in Chap 3)

Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server processes

Client/server socket interaction: TCP

server (running on hostid)

client 

create socket,
port=**x**, for incoming
request:
serverSocket = socket()

wait for incoming
connection request
connectionSocket =
serverSocket.accept()

TCP
connection setup

create socket,
connect to **hostid**, port=**x**
clientSocket = socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

read reply from
clientSocket

close
clientSocket

Copy-and-paste-friendly version of the code available at https://s.ntnu.no/iikt2105-sockets

Example App: TCP Client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000 →

No need to attach server name, port →

Copy-and-paste-friendly version of the code available at https://s.ntnu.no/iikt2105-sockets

Example App: TCP Server

Python TCPServer

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('The server is ready to receive')
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    connectionSocket.close()
```

create TCP welcoming socket →

server begins listening for incoming TCP requests →

loop forever →

server waits on accept() for incoming requests, new socket created on return →

read bytes from socket (but not address as in UDP) →

close connection to this client (but *not* welcoming socket) →

Readings for next week

Book pages

- 173-179 (Sec. 2.6.1-3)
- 182-195 (2.7)

Key concepts

Video & CDN
- Streaming
- Encoding
- DASH for supporting several different encodings
- Video buffer

Cluster selection

Sockets
- TCP vs UDP
- serverSocket vs connectionSocket
- bind

Short survey

See you next week!

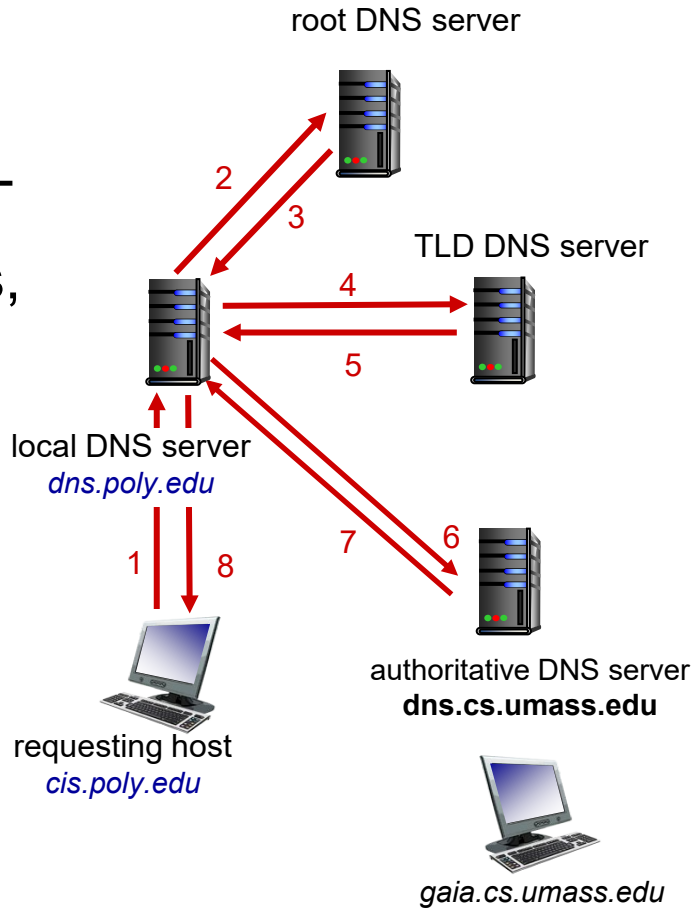
Problem X

Referring to Problem IX, suppose the HTML file references 8 very small objects on the same server. Neglecting transmission times, how much time elapses with:

a) Non-persistent HTTP with no parallel TCP connections?

b) Non-persistent HTTP with the browser configured for 5 parallel TCP connections?

c) Persistent HTTP?



Problem 22 (*pg. 203)*

Consider distributing a file of $F = 10\ \text{Gbit}$ to N peers. The server has an upload rate of $u_s = 1\ \text{Gbit/s}$ and each peer has a download rate of $d_i = 200\ \text{Mbit/s}$ and an upload rate of u. For $N = 10, 100 \text{ and } 1000$ and $u = 2\ \text{Mbit/s}, 10\ \text{Mbit/s}$ $\text{and } 100\ \text{Mbit/s}$, prepare a table giving the minimum distribution time for each of the combinations of N and u for both client-server distribution and P2P distribution.

Problem 26 (*pg. 204)*

Suppose Bob joins a BitTorrent torrent but he does not want to upload any data to any other peers (so-called free-rider).

a) Alice who has been using BitTorrent tells Bob that he cannot receive a complete copy of the file that is shared by the swarm. Is Alice correct or not? Why?

b) Charlie claims that Alice is wrong and that he has even been using a collection of multiple computers (with distinct IP addresses) in the computer lab in his department to make his downloads faster, using some additional coordination scripting. What could his script have done?

Solving Problem 26 (*pg. 204)*

- Alice is incorrect. As long as there are enough peers staying in the swarm for a long enough time. Bob can always receive data through optimistic unchoking by other peers.
- Charlie can run a client on each host, let each client “free-ride,” and combine the collected chunks from the different hosts into a single file. He can even write a small scheduling script to make the different hosts ask for different chunks of the file. This is actually a kind of Sybil attack in P2P networks.

Problem 19 (*pg. 203)*

In this problem, we use the useful *dig* tool available on Unix and Linux hosts to explore the hierarchy of DNS servers. Recall that in Figure 2.19, a DNS server in the DNS hierarchy delegates a DNS query to a DNS server lower in the hierarchy, by sending back to the DNS client the name of that lower-level DNS server. First read the man page for *dig*, and then answer the following questions.

a) Starting with a root DNS server (from one of the root servers [a-m]. root-servers.net), initiate a sequence of queries for the IP address for your department's Web server by using *dig*. Show the list of the names of DNS servers in the delegation chain in answering your query.

b) Repeat part (a) for several popular Web sites, such as google.com, yahoo .com, or amazon.com.