Lecture 4:

The Application Layer (part 2)

Katerina Argyraki, EPFL

Today, we will continue discussing the application layer.
We will start by recapping some basic concepts from previous lectures.

Interface

- A point where two systems, subjects, organizations, … meet and interact.

We discussed the concept of an “interface”, which is the point where two entities — systems, subjects, organisations — meet and interact.

Application Programming Interface

- Interface between application and Internet
- A set of functions that an application must use to communicate over the Internet

For instance, we discussed the “Application Programming Interface (API)” between a distributed application and the Internet.

This interface consists of a set of functions that are the only way for an application to communicate over the Internet.

Network interface

- Interface between an end-system and the network
- A piece of hardware or software that sends and receives packets
- Example: your network card is a (hardware) network interface

Then we discussed the concept of a “network interface,”
which its the interface between an end-system and the network.

This interface consists of a piece of hardware or a piece of software involved in sending and receiving packets.

DNS name

- Identifies a network interface
  = identifies an end-system

- Also called a "hostname"
  - an end-system is also called a "host"

From there, we moved on to the concept of a DNS name, which identifies a particular network interface.

This is a good moment to mention that a DNS name is also called a “hostname,”
because end-systems are also called “hosts.”

URL

- Identifies a web object
  - example: www.epfl.ch/index.html
- Format: DNS name + file name
  - www.epfl.ch identifies a network interface
  - index.html identifies a file

Then we discussed the concept of a URL, which identifies a web object.
We said that a URL consists of 2 parts:
- a DNS name (which identifies a network interface, hence also an end-system)
- and a file name (which identifies a particular file stored in that end-system).

Process name/address

- Identifies a process
  = app-layer piece of code
  - example: 128.178.50.12, 80
- Format: IP address + port number
  - 128.178.50.12 identifies a network interface
  - 80 identifies a process

We also discussed processes:

We said that a process is a piece of code that runs in the application layer,
which is reachable at particular “process name” or “process address,”
which consists of 2 parts:
- an IP address (which identifies a network interface, hence an end-system)
- and a port number (which identifies a process running in the application layer of that end-system).

Some port numbers are universal.
For example, port 80 is universally used to refer to web-server processes.
So, the process address I am showing you on this slide names
a web-server process running in the application layer of an end-system that has a network interface with the specified IP address.

Web request revisited

- You enter a URL into your web client
  - http://www.epfl.ch/index.html
- Web client extracts DNS name
  - www.epfl.ch
- Translates DNS name to IP address
  - 104.20.228.42
- Forms web-server process name
  - 104.20.228.42, 80 (or 8080)
  - 80, 8080: port numbers for http servers

Now let's revisit what happens when you enter a URL into your web browser:
- The browser first extracts the DNS name from the URL,
- then translates the DNS name into an IP address,
- adds port number 80 to it (because that is the universal port number for web-server processes),
- and forms the process name at which it can reach the web server that stores the requested URL.

So:
- You provide your browser with a URL, which is easy for humans to remember,
- but your browser translates that into a process name, which is what it needs in order to communicate with the web server.

Web request revisited

- You enter a URL into your web client
  - https://www.epfl.ch/index.html
- Web client extracts DNS name
  - www.epfl.ch
- Translates DNS name to IP address
  - 104.20.228.42
- Forms web-server process name
  - 104.20.228.42, 443 (or 8443)
  - 443, 8443: port numbers for https servers

Web request revisited

- You enter a URL into your web client
  - www.epfl.ch/index.html
- Web client extracts DNS name
  - www.epfl.ch
- Translates DNS name to IP address
  - 104.20.228.42
- Forms web-server process name
  - 104.20.228.42, 80

Now we will focus on one particular step of this sequence of events:
the translation from a DNS name to an IP address.

How does the web browser know which is the IP address that corresponds to DNS name www.epfl.ch?

Example 2: DNS

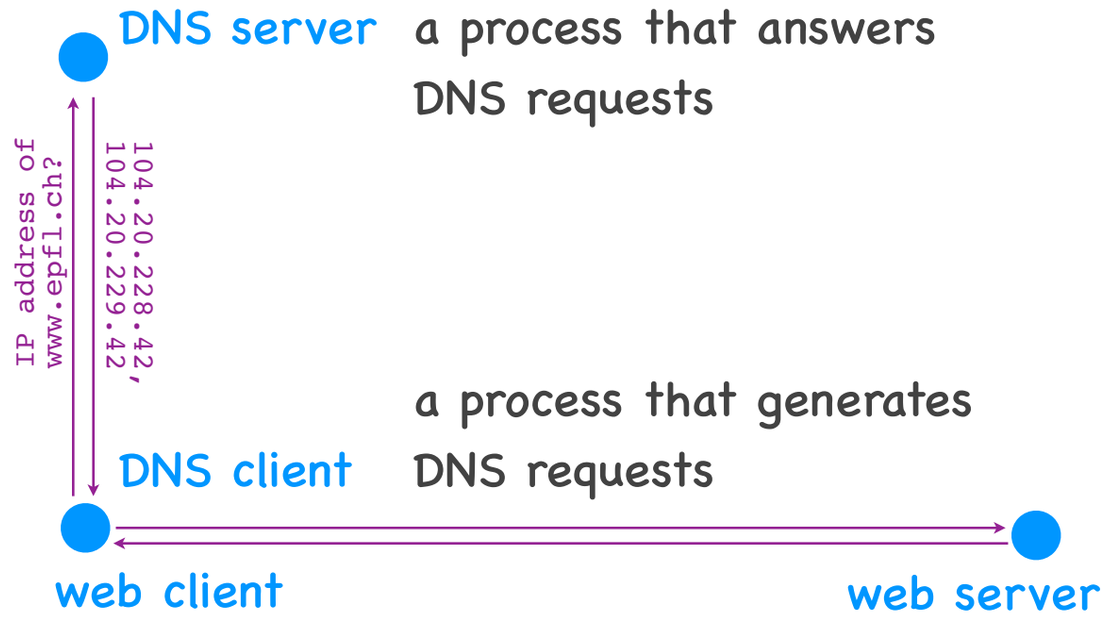
The translation from DNS names to IP addresses is done by the Domain Name System (DNS), which is the next application that we will look at.

DNS is a very special application, which is very important to the operation of the Internet, because other applications use it to translate DNS names to IP addresses.

Design an application =

- Design the architecture
  - which process does what?
- Design the communication protocol
  - what sequences of messages can be exchanged?
- Choose the transport-layer technology
  - what kind of delivery is needed?

First question: what architecture does DNS use: client/server or P2P?
It uses…



The client-server architecture.

Suppose we have a web client, also called a web browser, that wants to contact the epfl web server.
The end-system that runs the web-browser process also runs a DNS client process.
So, the web browser talks to the DNS client, and says “I need the IP address of www.epfl.ch."
The DNS client sends this request to another process, running on a different end-system, that is called a DNS server.
The DNS server responds to the DNS client with the requested IP address,
the DNS client forwards the response to the web browser,
and, at that point, the web browser, can form the process name of the epfl web server that it wants to contact.

So:

- A DNS client is a process that makes requests to a DNS server on behalf of a web browser or some other process, and also forwards the responses to them.
- A DNS server is a process that is always running on an end-system and translates DNS names to IP addresses (among other things).

www.epfl.ch	104.20.228.42, 104.20.229.42
www.search.ch	195.141.85.90
facebook.com	157.240.201.35
google.com	172.217.168.14
www.stanford.edu	34.196.104.129, 3.90.95.150

If we looked inside a DNS server, we would essentially find a database, mapping lots and lots of DNS names to IP addresses (plus other information that we are not discussing today).

Could we have a single DNS server in the entire Internet?

—>Could we have a single DNS server serving the entire Internet?

No, in the case of DNS, a single-server design “does not scale”:

- It would receive too much traffic (because all the DNS clients in the world would send requests to it).
- It could not be close to every single DNS client in the world, so many DNS clients would experience long delays when communicating with it.
- It would be a single point of failure. If it failed, nobody would be able to browse the web (modulo caching, but we will discuss that later).
- It would be hard for a single management team to maintain it (because they would need to keep track of all the DNS names in the world and the corresponding IP addresses).

Scalability (informally)

- Ability to grow
- As the system grows,
  it maintains its properties
  at a reasonable cost

Informally, scalability is a system's ability to grow,
meaning to maintain its properties, as it grows, at a reasonable cost.

However powerful we make a single DNS server,
we cannot have just one serving the entire Internet.

Instead, we have a …

Hierarchy of DNS servers

root servers

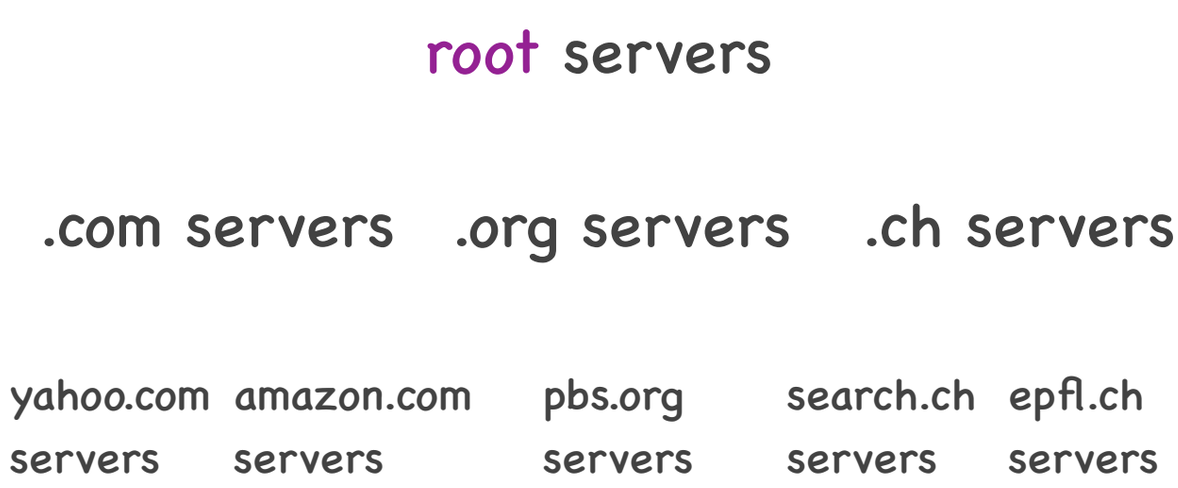
TLD (top-level domain) servers

authoritative servers

hierarchy of DNS servers:

- At the top of the hierarchy, we have the root servers,
- below that the top-level-domain (TLD) servers,
- and below that the authoritative servers.

Hierarchy of DNS servers



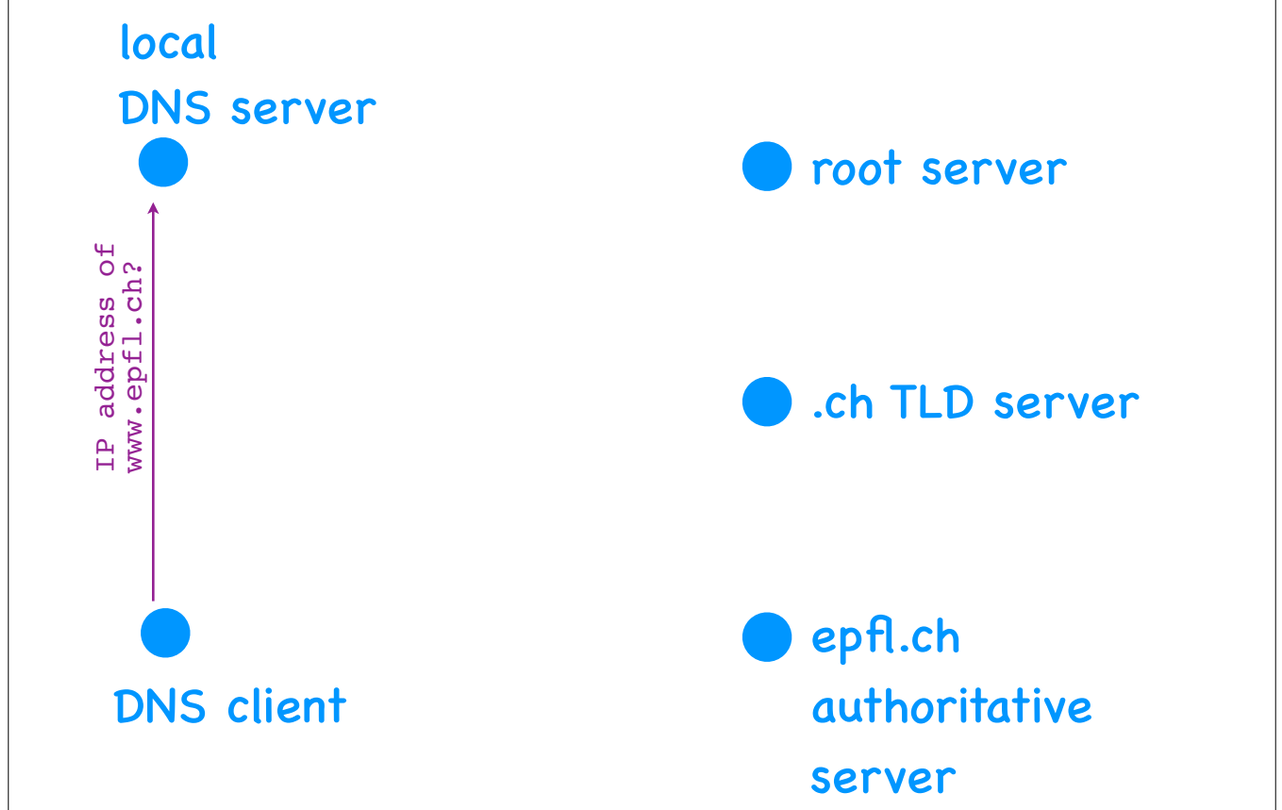
A few more details:

A “top level domain” or “TLD” is a name that constitutes the last component of a DNS name. E.g., “.com”, “.org”, “.eu”, “.ch.”
For each TLD, there exists a set of TLD servers.

A “lower-level domain” is a name that constitutes the last two components of a DNS name. E.g., “yahoo.com", “amazon.com", “epfl.ch."
For each lower-level domain, there exists a set of authoritative servers.

Each DNS server in this hierarchy knows only how to reach its children:

- A root server knows how to reach TLD servers for all TLDs.
- A .ch TLD server knows how to reach authoritative servers for all the .ch domains.
- An epfl.ch authoritative server knows all the mappings for all the DNS names that belong to EPFL.



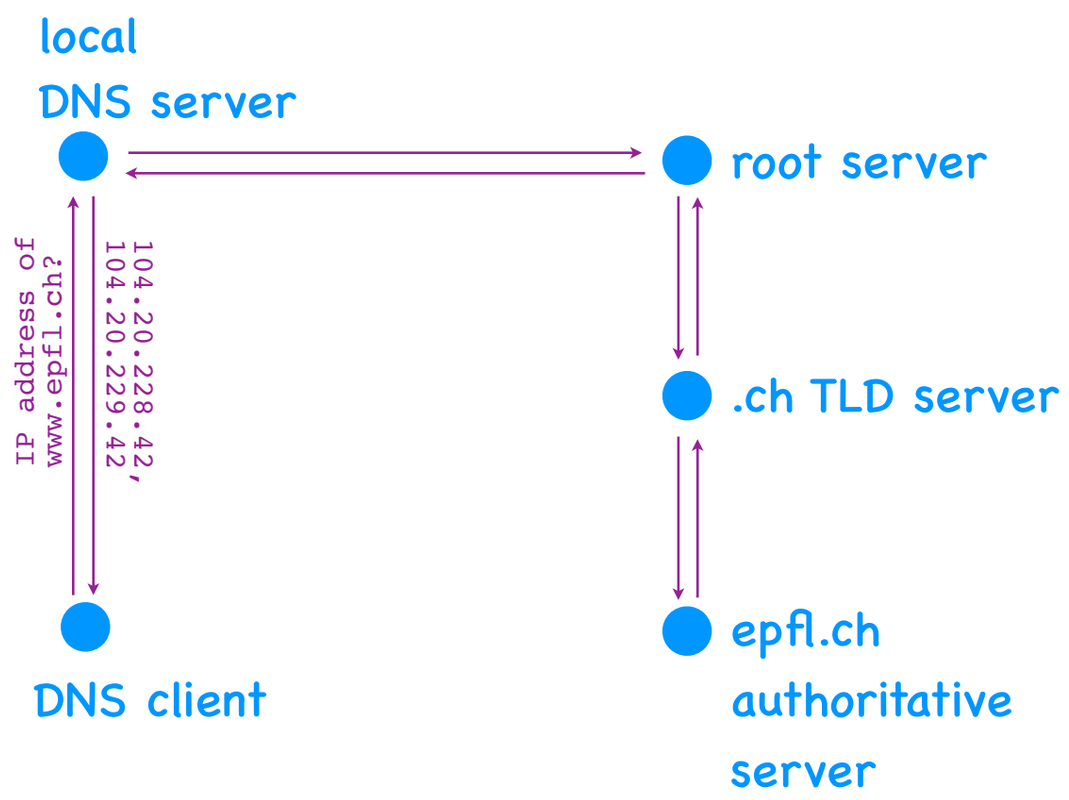
Let’s see how all these DNS servers work together to provide their service.

Every DNS client in the world knows the IP address of at least one “local” DNS server
(a DNS server that is physically close to the DNS client).

Every DNS server in the world knows the IP address of at least one root DNS server.

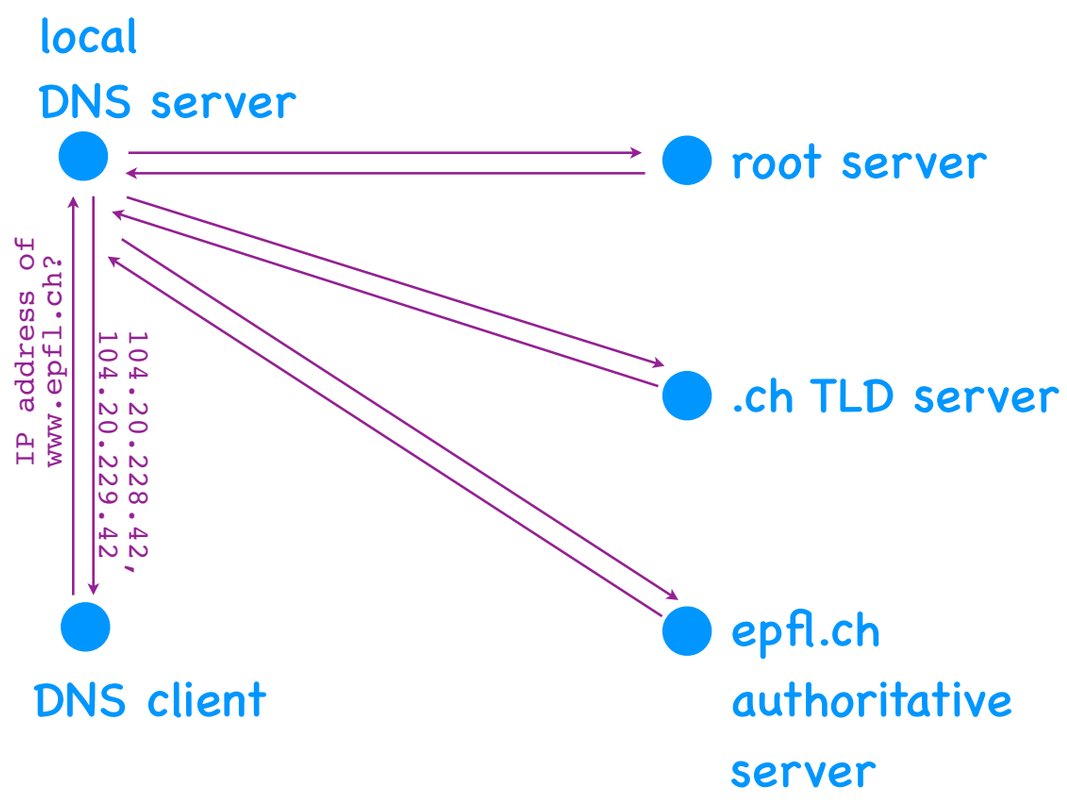
When a DNS client wants to resolve a DNS name, it asks a local DNS server.
The local DNS server may not know the IP address of www.epfl.ch, but it certainly knows the IP address of a root server.
The root server may not know the IP address of www.epfl.ch, but it certainly knows the IP address of a .ch TLD server.
The .ch TLD server also may not know the IP address of www.epfl.ch, but it certainly knows the IP address of an epfl.ch authoritative server.
The epfl.ch authoritative server certainly knows the IP address for www.epfl.ch, because that is its job.

There are two ways for the local DNS server to get the answer:



Through recursive requests:

- The local DNS server asks the root server,
- which asks the TLD server,
- which asks the authoritative server,
- which responds to the TLD server,
- which responds to the root server,
- which responds to the local DNS server.



Or though iterative requests:

- The local DNS server asks the root server, which responds with the IP address of the TLD server.
- The local DNS server asks the TLD server, which responds with the IP address of the authoritative server.
- The local DNS server asks the authoritative server, which responds with the requested IP address.

DNS processes

- DNS client
  - helps apps map DNS names to IP addresses
- Local DNS server
  - answers requests from nearby DNS clients
- Hierarchy of DNS servers
  - answers requests from local DNS servers

So: DNS uses a client/server architecture:
- DNS client processes help applications resolve DNS names to IP addresses. These act as clients to
- Local DNS server processes. These act as clients to
- An entire hierarchy of DNS servers.

Hierarchy of DNS servers

- Three levels: root servers, TLD servers, authoritative servers
- Each node knows how to reach its children
  - root servers know TLD servers for each TLD
  - TLD servers know authoritative servers for each lower-level domain within their TLD

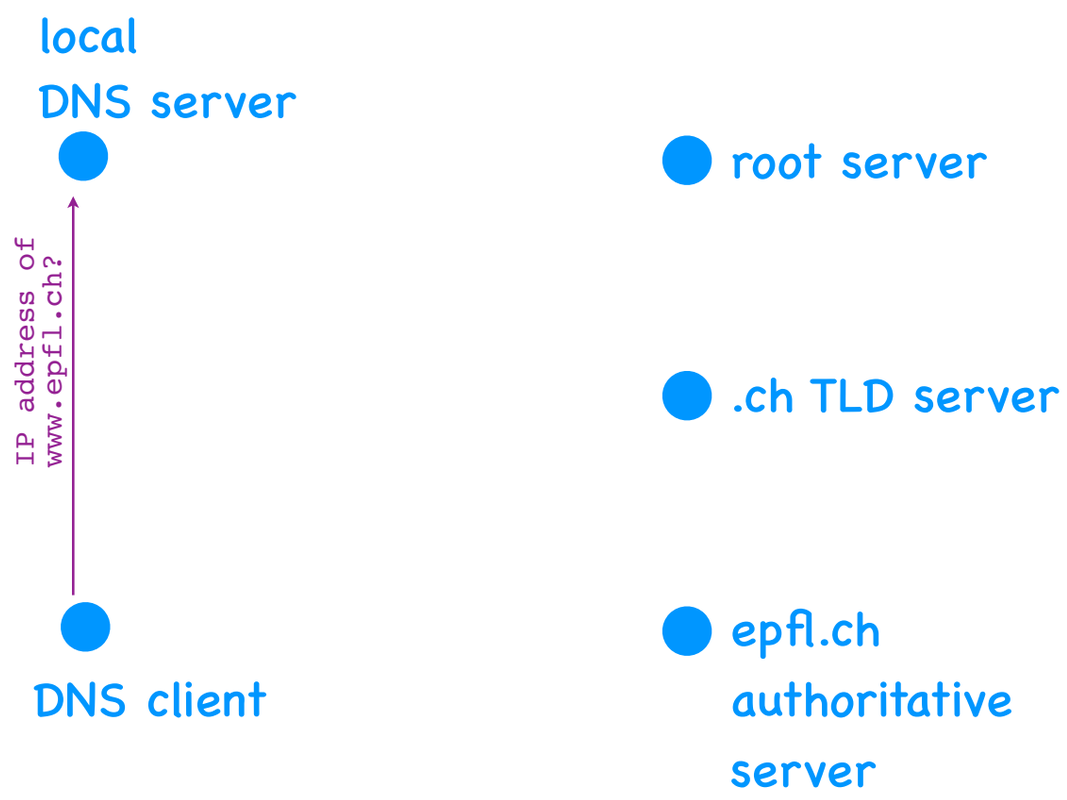
The hierarchy has 3 levels: root, TLD, and authoritative servers.

And each server in the hierarchy knows how to reach its children: …

For example, a .ch TLD server knows authoritative servers for all the lower-level domains that end with .ch, for example, epfl.ch, ethz.ch, meteosuisse.ch, and so on.

Hierarchy

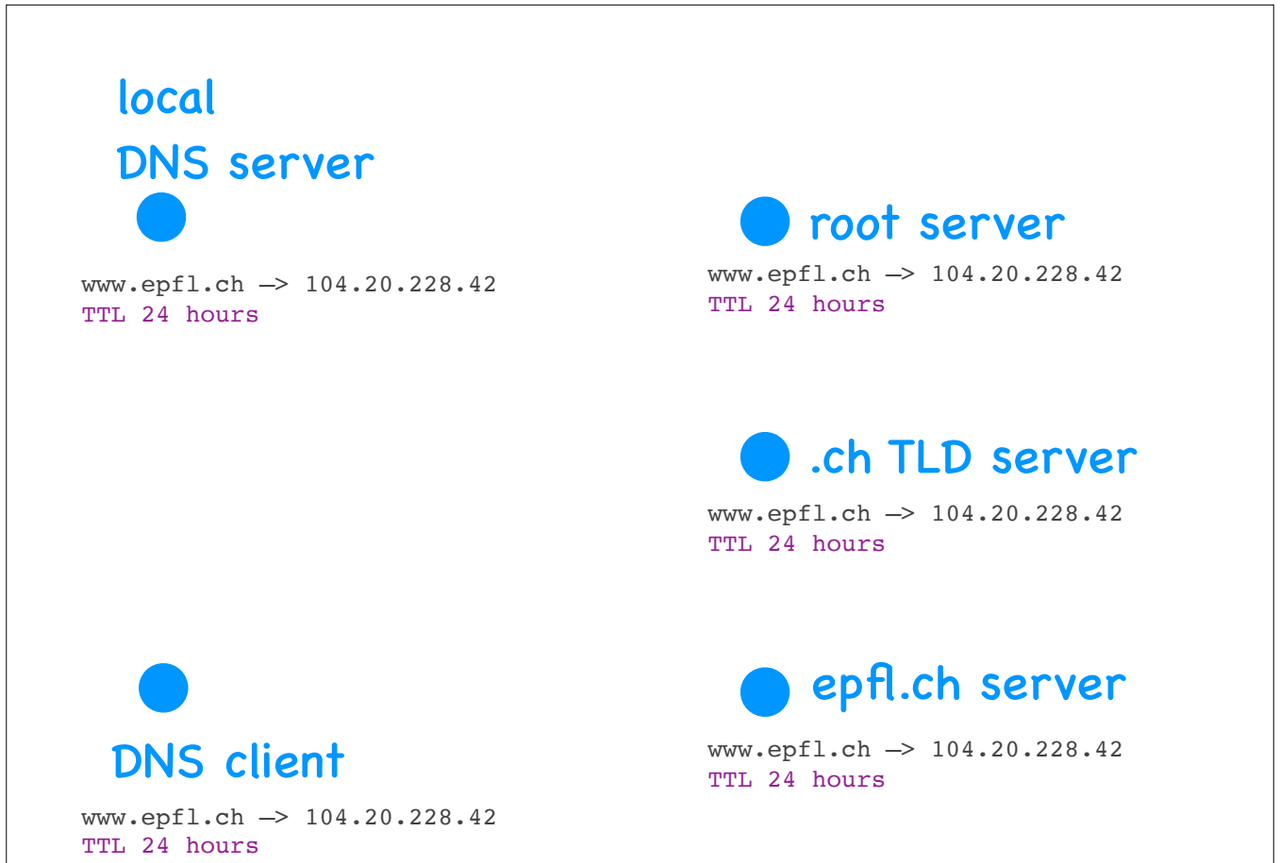
- Universal technique
  for scaling large systems



The way I have described things, when a DNS client makes a request, it has to wait until 4 different DNS servers are contacted, and this may take a long time.
Is there anything that can be done to improve the delay experienced by DNS clients around the world?

The answer is caching.
All DNS clients and DNS servers cache the DNS responses they receive, so that they do not need to repeat the same DNS requests.

How to prevent stale data?



The answer is: with time to live (TTL).

Whenever a mapping is established at an authoritative server, there is a time to live (TTL) associated with that mapping.

When the authoritative server sends that mapping to a TLD server, in response to a DNS request, it also sends the TTL.

Similarly, when a root server or a local DNS server, or a DNS client receive this mapping, they also receive the TTL.

So, DNS clients and servers that cache a mapping know for how long the mapping is expected to be correct.

DNS caching

- All DNS clients and servers cache name-to-IP address mappings
- Reduces load at all levels
- Reduces delay experienced by apps
- Relies on time to live (TTL) for freshness

Caching

- Universal technique
  for improving performance

- Challenge: stale data
  - option #1: dynamic check for staleness
  - may introduce significant delay

We said in the last lecture that caching is a universal technique in computer science.
Whenever we have a set of entities interested in the same data,
the moment one of these entities pays the cost to bring the data closer,
it is worth keeping the data closer -- so that the other entities do not have to pay the same cost.

The challenge is dealing with stale data.
The first option we saw for addressing this challenge is to dynamically check for staleness, which is what web server proxies do.
But this, we said, may introduce significant delay.

Caching

- Universal technique
  for improving performance

- Challenge: stale data
  - option #1: dynamic check for staleness
  - option #2: tolerate some inconsistency

The second option we are seeing now for addressing this challenge is rely on time to live, which assumes a bounded update rate and introduces some amount of inconsistency.

Why do we use option #1 for web caching
but option #2 for DNS caching?

Which of the following DNS servers is guaranteed to know the IP address of www.epfl.ch?

(a) A local DNS server.

(b) A root DNS server.

(c) An epfl.ch authoritative server.

Every Internet end-system must know the IP address of at least one

(a) DNS server.

(b) root DNS server.

(c) authoritative DNS server for each lower-level domain it wants to communicate with.

Every DNS server must know the IP address of at least one

(a) local DNS server.

(b) root DNS server.

(c) authoritative DNS server for each lower-level domain in the world.

Design an application =

- Design the architecture
  - which process does what?
- Design the communication protocol
  - what sequences of messages can be exchanged?
- Choose the transport-layer technology
  - what kind of delivery is needed?

Second question: what is the communication protocol used by DNS client and server processes?

DNS protocol elements

- Resource Record (RR)
  - piece of information,
    e.g., DNS name to IP address mapping
  - multiple types: A, CNAME, MX, SOA, ...
- Query: request for an RR
- Answer: response to a question

It is called the DNS protocol, and it contains the following basic elements:

A “resource record” (RR) is a piece of information, for example, a mapping from a DNS name to a set of IP addresses.

A “query” is a request for an RR.
An “answer” is the corresponding response.

DNS protocol elements

- Message
  - contains sets of queries and answers
  - (plus other elements…)
- A DNS client and server or two DNS servers can exchange any sequence of messages

A DNS message contains sets of questions and answers.

A DNS client and server, or two DNS servers can simply exchange any sequence of DNS messages.

Design an application =

- Design the architecture
  - which process does what?
- Design the communication protocol
  - what sequences of messages can be exchanged?
- Choose the transport-layer technology
  - what kind of delivery is needed?

Third and last question: what transport-layer technology does DNS use? TCP or UDP?

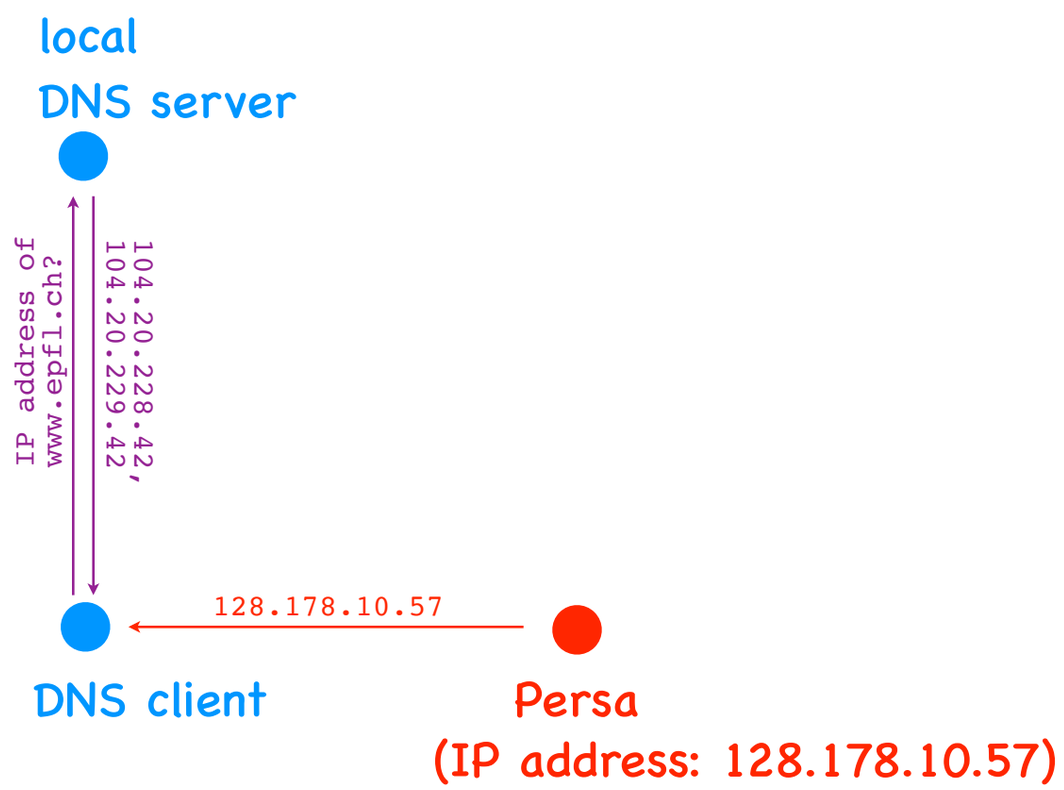
Would you use TCP or UDP
for DNS's transport layer? Why?

DNS transport layer

- UDP (for short exchanges)
  - does not make sense to pay the cost of TCP connection setup
- TCP (typically between DNS servers)
  - can amortise the cost of TCP connection setup

How can one attack the DNS system?

Before we close our exploration of DNS, there is one more interesting question to discuss: how can one attack the DNS system?



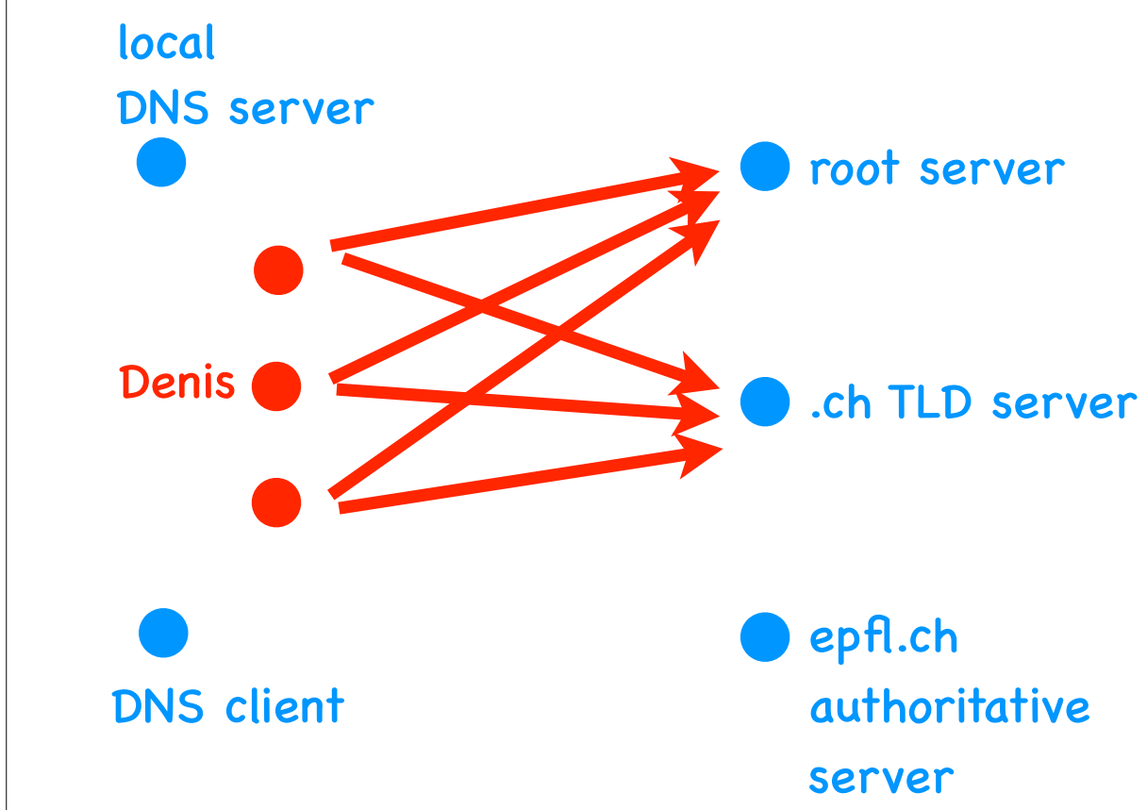
Suppose we have a DNS client, which asks its local DNS server for the IP address of www.epfl.ch.

And suppose there is an attacker called Persa, who has this IP address.

What can Persa do?
Her name should give you a hint.

Persa can respond to the DNS client that the IP address of www.epfl.ch is her own IP address.

Of course, the local DNS server will also respond with the correct IP address.
But, if Persa manages to send her response faster, then the DNS client will not be waiting for a response any more, and will just ignore it.



Now consider the entire DNS hierarchy,

and an attacker called Denis, who controls multiple end-systems.

What can Denis do to interfere with the operation of DNS?
His name should give you a hint.

Denis can denial-of-service the root DNS servers and/or the TLD servers.
For example, he can send lots and lots of DNS requests to the root and TLD servers,
such that they spend all their resources responding to his requests,
and they cannot respond to legitimate requests,
or their response time becomes very slow.

Why are the root and TLD servers good attack targets?
Because there are relatively few of them,
and if one brings them down,
they are affecting the operation of the entire Internet.

Hierarchy

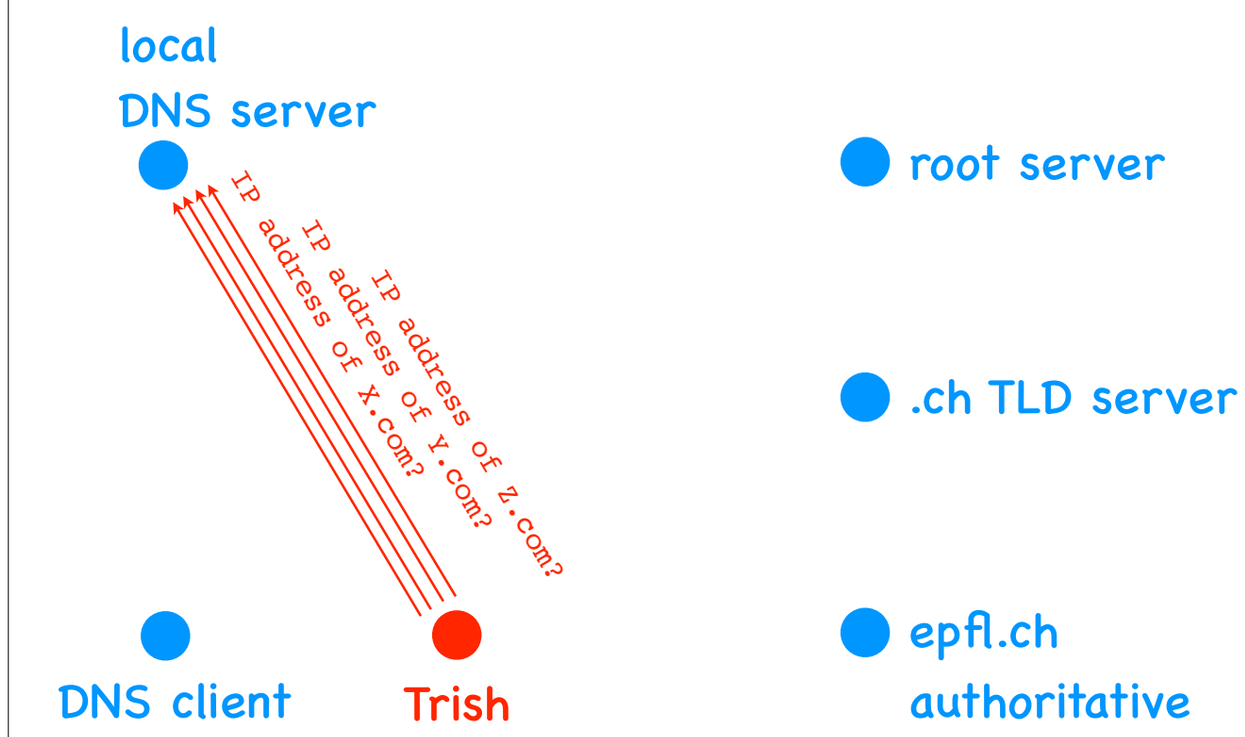
- Universal technique
  for scaling large systems

- Nodes that are high up in the
  hierarchy make good attack targets

Parenthesis here:

We said that hierarchy is a universal technique for scaling large systems.

However, when one designs a hierarchical system,
they should be careful, because nodes that are high up in the hierarchy make good attack targets.



Consider again the DNS system,

and an attacker called Trish.

Trish can send lots of DNS requests to a DNS server,
but not with the purpose of preventing it from responding to legitimate requests.

Rather, she asks the DNS server to translate obscure DNS names that nobody really cares for.

What will be the result of this?

The DNS server will do what it needs to do to translate these DNS names and will then cache the results. If there are many such results, the DNS server may end up filling its cache with these unpopular DNS names that nobody other than Trish will ask about. Which means that it may end up evicting from its cache the popular DNS names that its clients care for.

So, this is an attack that interferes not with the functionality of DNS, but its performance.

Caching

- Universal technique
  for improving performance

- Trashing the cash is a potential
  vulnerability

Parenthesis:

We said that caching is a universal technique for improving performance.

But, whenever a system relies on caching for normal operation,
trashing the cache becomes a potential vulnerability.

Attacks against DNS

- Impersonate a DNS server
  and provide an incorrect mapping
- DoS the root servers
  and/or TLD servers
- Trash the cache of a DNS server
  to slow down its responses

667

Original story 1:26 pm EDT: Facebook—and apparently all the major services Facebook owns—are down today. We first noticed the problem at about 11:30 am Eastern time, when some Facebook links stopped working. Investigating a bit further showed major DNS failures at Facebook:

Jim Salter
@jrssnet

So, @facebook's DNS is broken this morning...

TL;DR: Google anycast DNS returns SERVFAIL for Facebook queries; querying a.ns.facebook.com directly times out.

```
root@jrs-router:/etc/bind# dig @a.ns.facebook.com www.facebook.com

; <<>> DiG 9.11.3-1ubuntu1.15-Ubuntu <<>> @a.ns.facebook.com www.facebook.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
root@jrs-router:/etc/bind# dig @8.8.8.8 m.facebook.com

; <<>> DiG 9.11.3-1ubuntu1.15-Ubuntu <<>> @8.8.8.8 m.facebook.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 49071
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;m.facebook.com.                        IN      A

;; Query time: 15 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Oct 04 11:46:05 EDT 2021
;; MSG SIZE  rcvd: 43
```

667

Original story 1:26 pm EDT: Facebook—and apparently all the major services Facebook owns—are down today. We first noticed the problem at about 11:30 am Eastern time, when some Facebook links stopped working. Investigating a bit further showed major DNS failures at Facebook:

Jim Salter
@jrssnet

So, @facebook's DNS is broken this morning...

TL;DR: Google anycast DNS returns SERVFAIL for Facebook queries; querying a.ns.facebook.com directly times out.

```
root@jrs-router:/etc/bind# dig @a.ns.facebook.com www.facebook.com

; <<>> DiG 9.11.3-1ubuntu1.15-Ubuntu <<>> @a.ns.facebook.com www.facebook.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
root@jrs-router:/etc/bind# dig @8.8.8.8 m.facebook.com

; <<>> DiG 9.11.3-1ubuntu1.15-Ubuntu <<>> @8.8.8.8 m.facebook.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 49071
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;m.facebook.com.                        IN      A

;; Query time: 15 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Mon Oct 04 11:46:05 EDT 2021
;; MSG SIZE  rcvd: 43
```

Example 3: BitTorrent (almost)

The next application we will explore is a simplified version of BitTorrent.

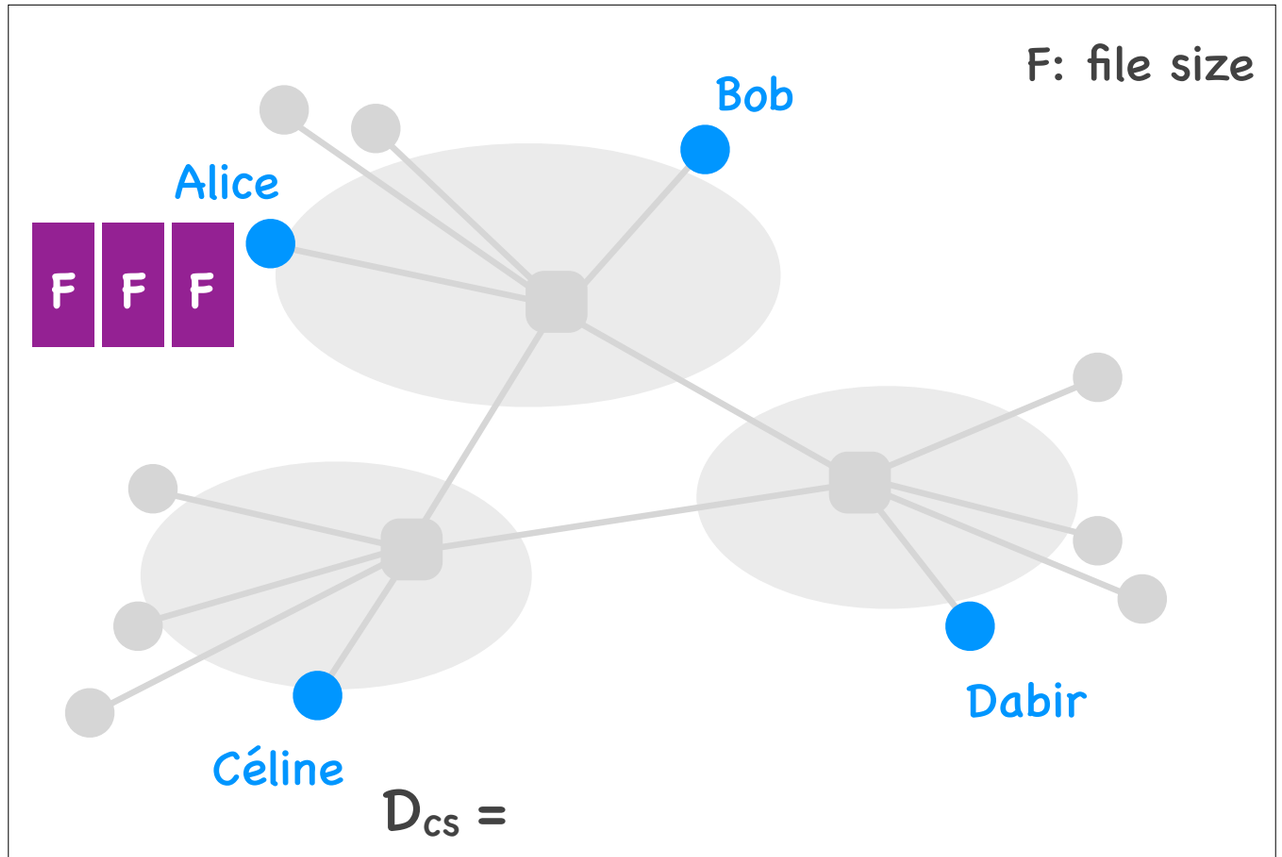
Design an application =

- Design the architecture
  - which process does what?
- Design the communication protocol
  - what sequences of messages can be exchanged?
- Choose the transport-layer technology
  - what kind of delivery is needed?

We will not look at BitTorrent's communication protocol or transport layer, we will focus on its architecture, which is P2P.

What does it mean that peer-to-peer "scales better" than client-server?

In the last lecture discussion, we said that P2P scales better than client-server.
The time has come to explore more concretely what it means for one architecture to “scale better” than another.



We will consider the following scenario:

Alice has a large file of size F, and she wants to share it with her three friends, Bob, Céline, and Dabir.

We will examine two approaches of distributing the file:
- a client-server approach, where Alice acts as a single server, while her friends act as clients;
- a P2P approach, where Alice and her friends act as peers.

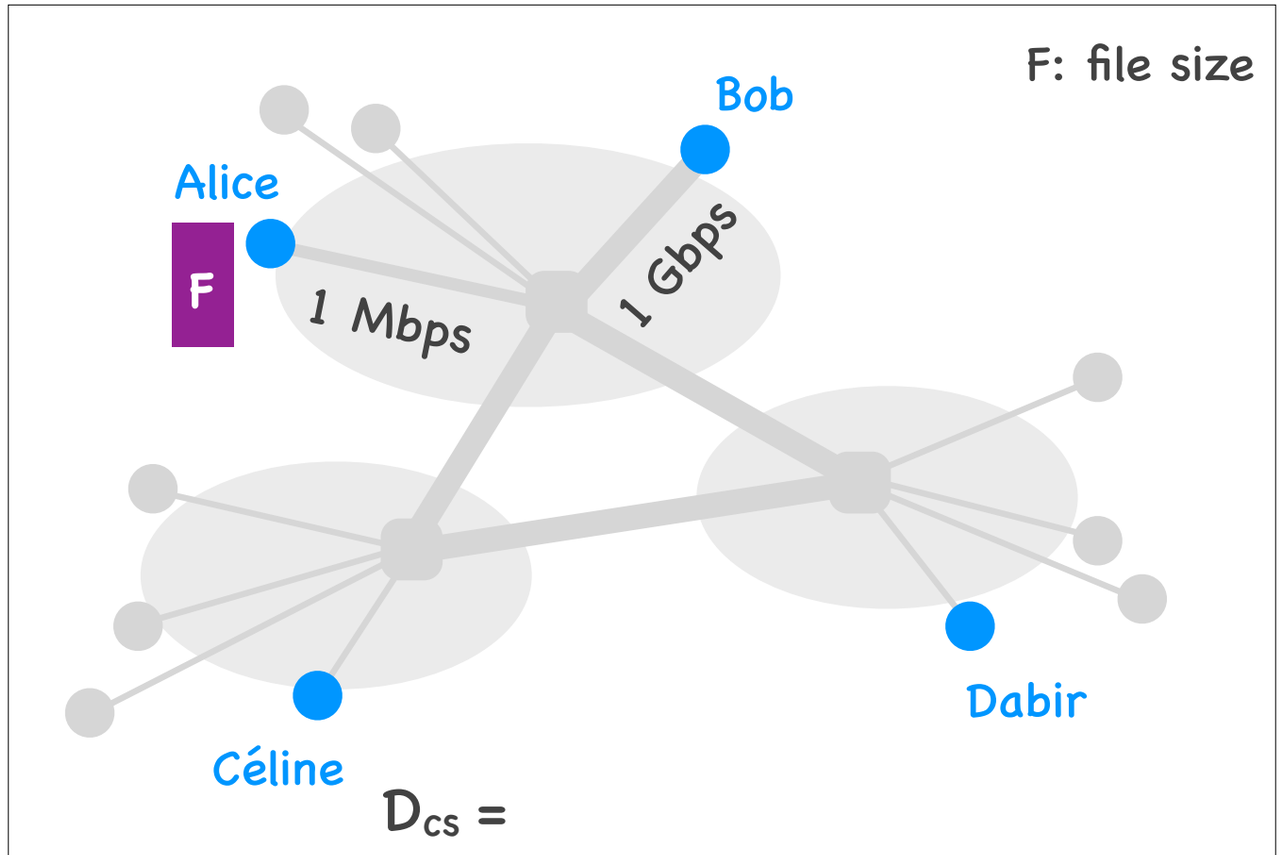
For each approach, we will compute a bound for the file distribution time, i.e., the amount it takes from the moment Alice transmits the first bit of the file until the last of her friends receives the last bit of the file.

We will assume there is no other traffic on the network, and there are no processing delays.
We will also assume that the transmission rates of the links in the middle of the network, i.e., between packet switches, are significantly larger than the transmission rates of the links at the edge of the network, i.e., the links that connect the end-systems to the network.

We will start from the client-server approach.

Alice creates three copies of the file and sends one copy to each friend.

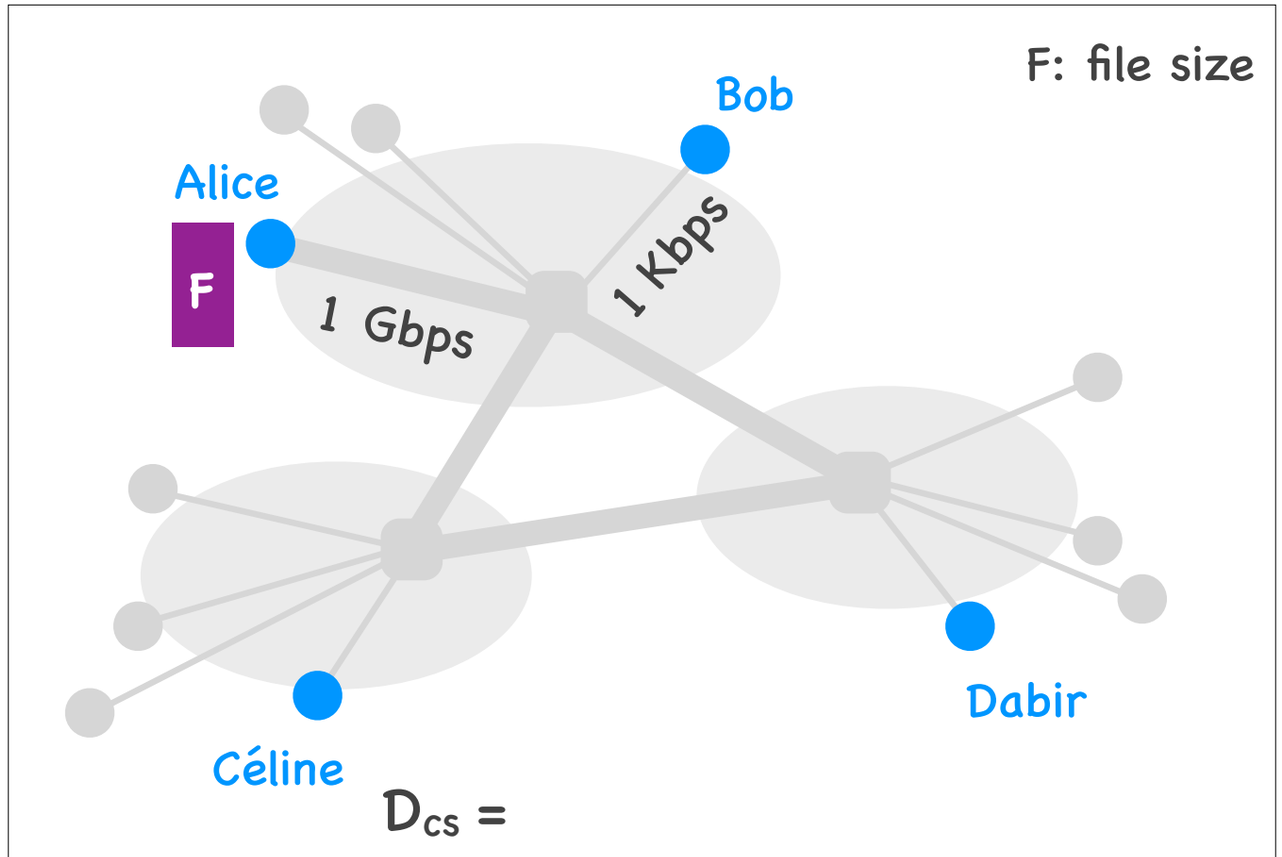
Let’s compute the file distribution time.



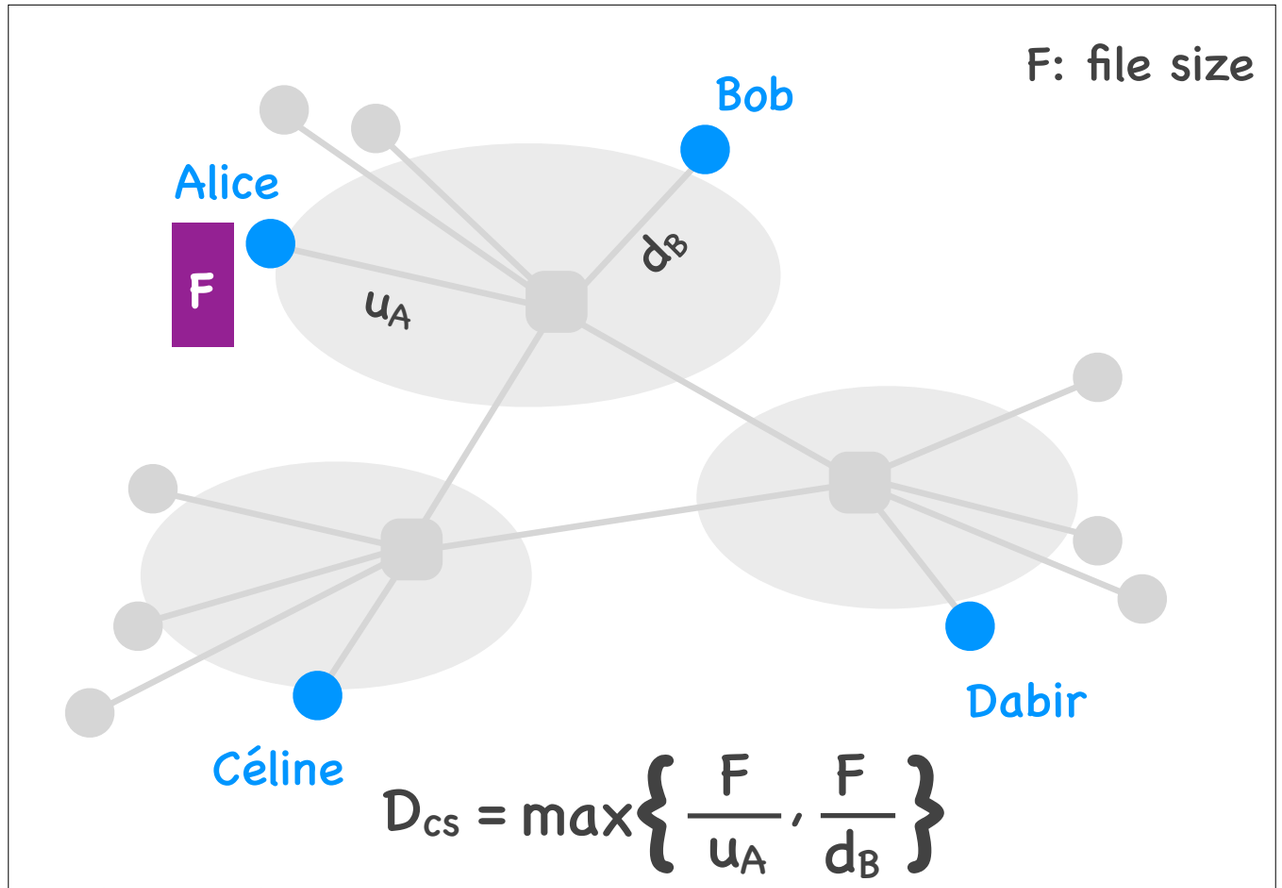
Suppose, for a moment, Alice sends the file only to Bob.
What is the file distribution time?

It depends on the bottleneck link between Alice and Bob.
Since there is no other traffic on the network, the bottleneck link between Alice and Bob is the one with the smallest transmission rate.

If it's the first link, then the file distribution time is the transmission delay of the file over the first link, plus some other components (the propagation delays of the links, the transmission delay of the last packet over the second link) that we can ignore.

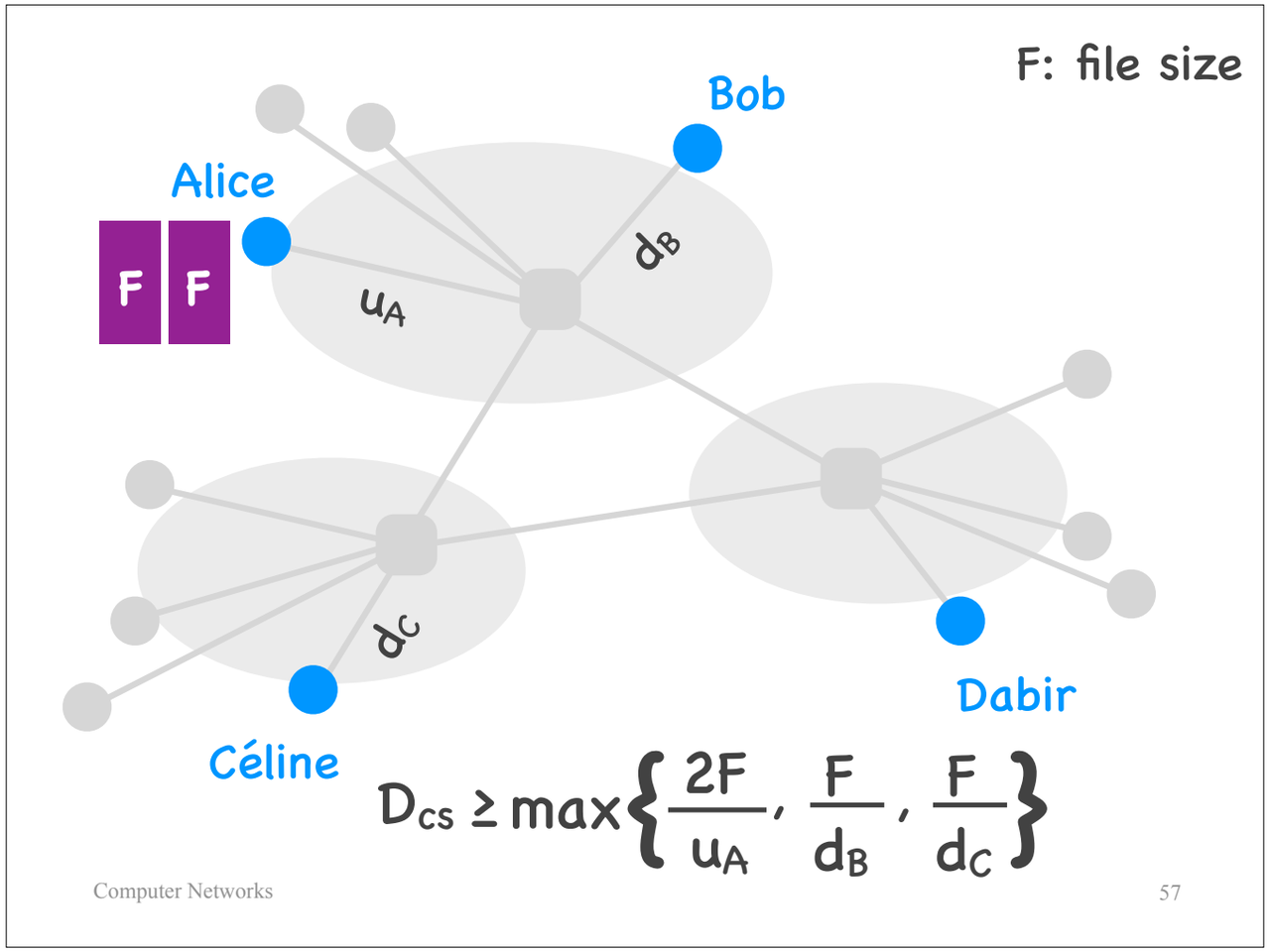


If the link with the smallest transmission rate is the second link, then the file distribution time is the transmission delay of the file over the second link plus some other components that we can ignore.



So, in general, the file distribution time is equal to the transmission delay of the file over the first or the second link, whichever one is bigger.

I am using u_A to denote Alice's upload capacity, i.e., the transmission rate of the first link,
and d_B to denote Bob's download capacity, i.e., the transmission rate of the second link.



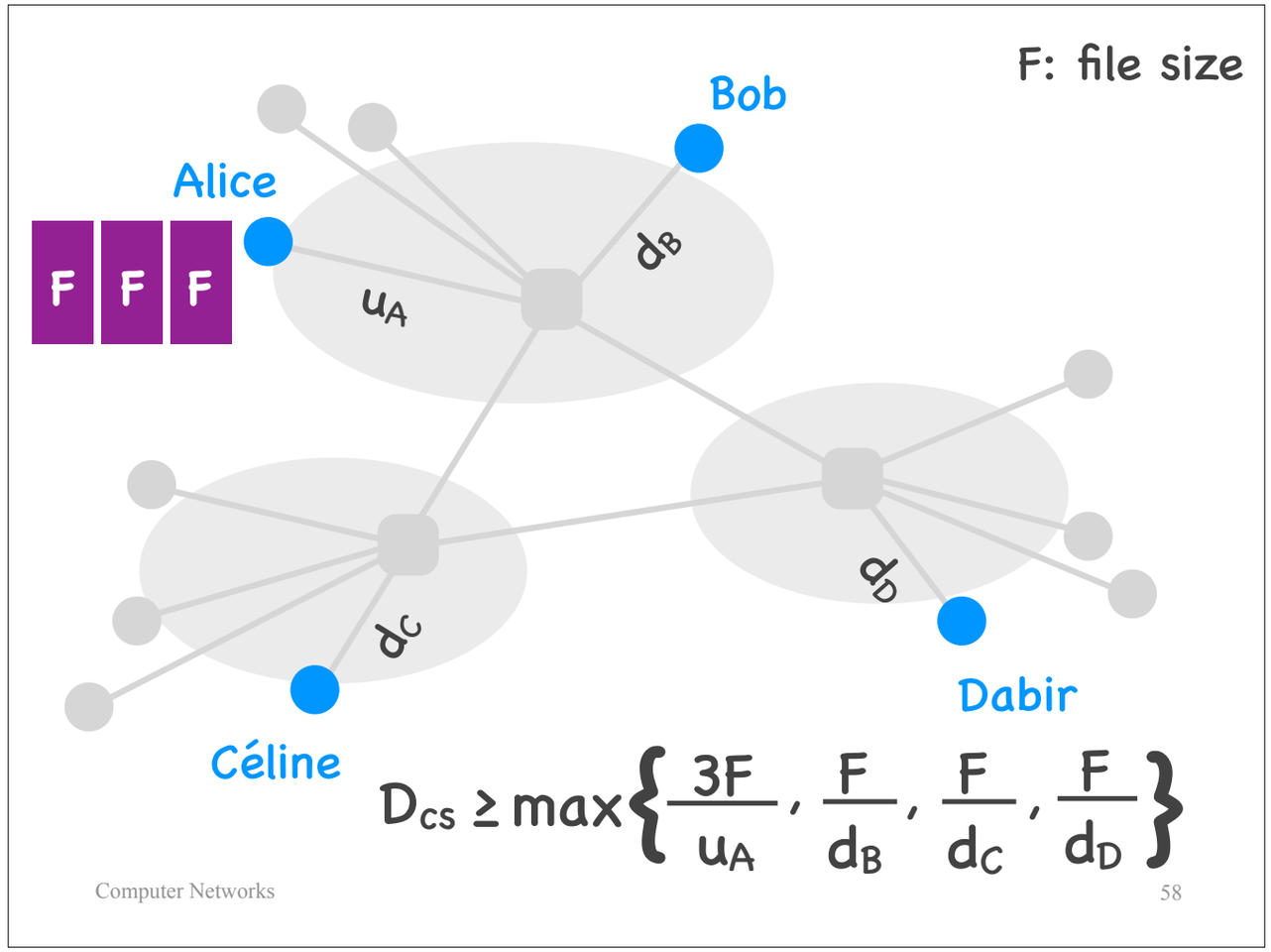
Now suppose Alice sends the file to Bob and Céline.

Assume that the packet switch that's closest to Alice, it has separate queues for the packets going to Bob and those going to Céline.
So, as Alice sends her two copies of the file, each copy is queued inside the packet switch in a separate queue.

The file distribution time is at least as large as the max of the following quantities:
- the transmission delay of the two copies of the file over the first link
- the transmission delay of Bob's copy over the last link to Bob
- the transmission delay of Céline's copy over the last link to Céline.

I would like you to pause the video and think about this formula.
Why am I not adding up the last two quantities, why am I taking the max over them?

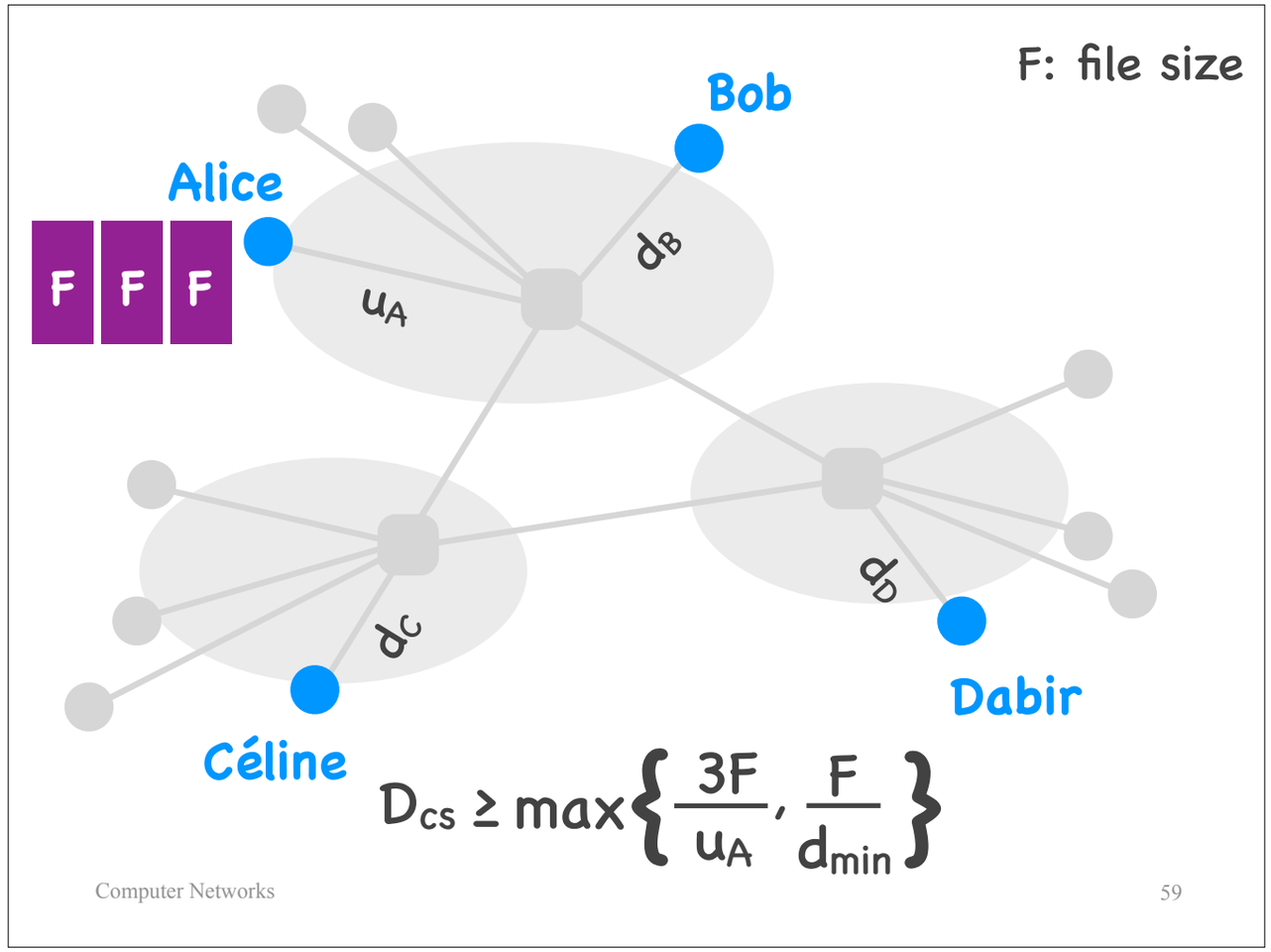
Because the copies to Bob and to Céline may end up traveling in the network at the same time, depending on the transmission rates of the links that are involved. This is the challenge, in general, with computing network delays, we must think about which operations may overlap in time.



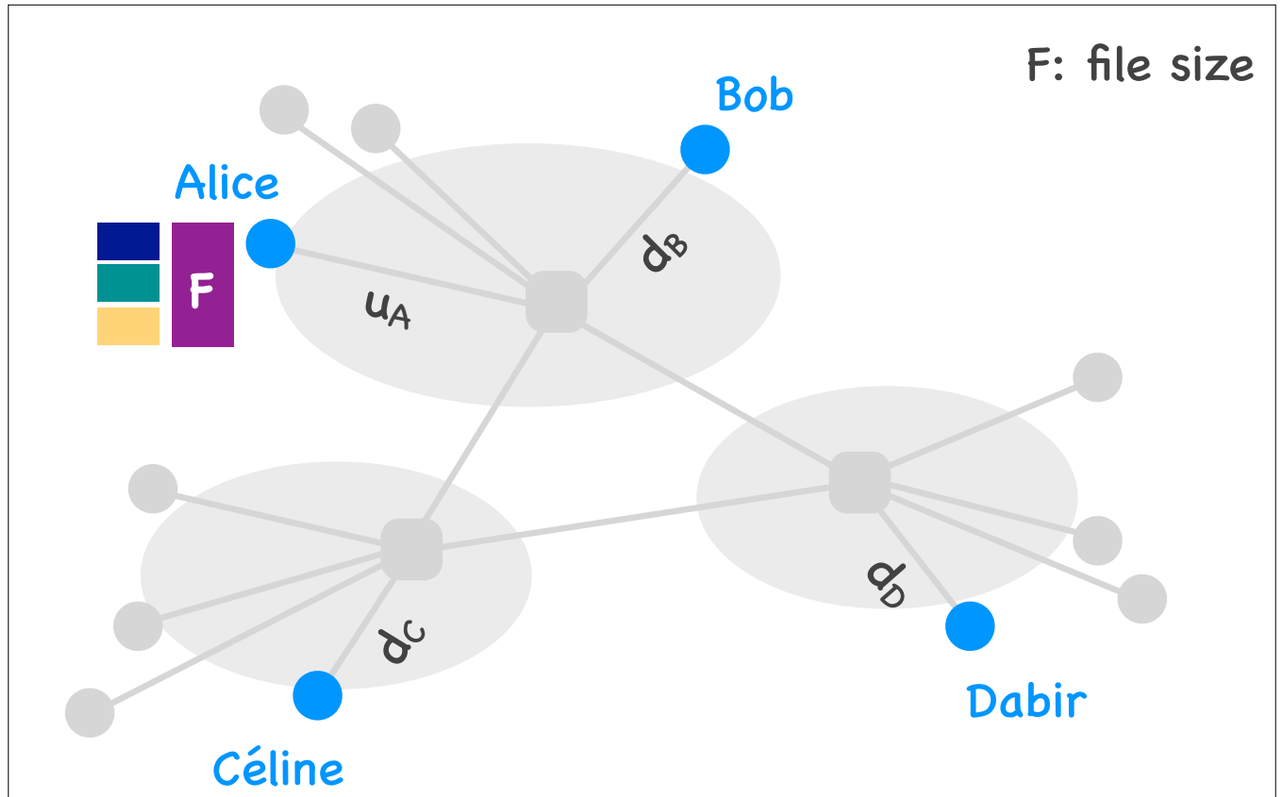
Finally, suppose Alice sends the file to all her three friends.

The file distribution time is at least as large as the max of the following quantities:

- the transmission delay of the three copies of the file over the first link
- the transmission delay of Bob's copy over the last link to Bob
- the transmission delay of Céline's copy over the last link to Céline
- the transmission delay of Dabir's copy over the last link to Dabir.

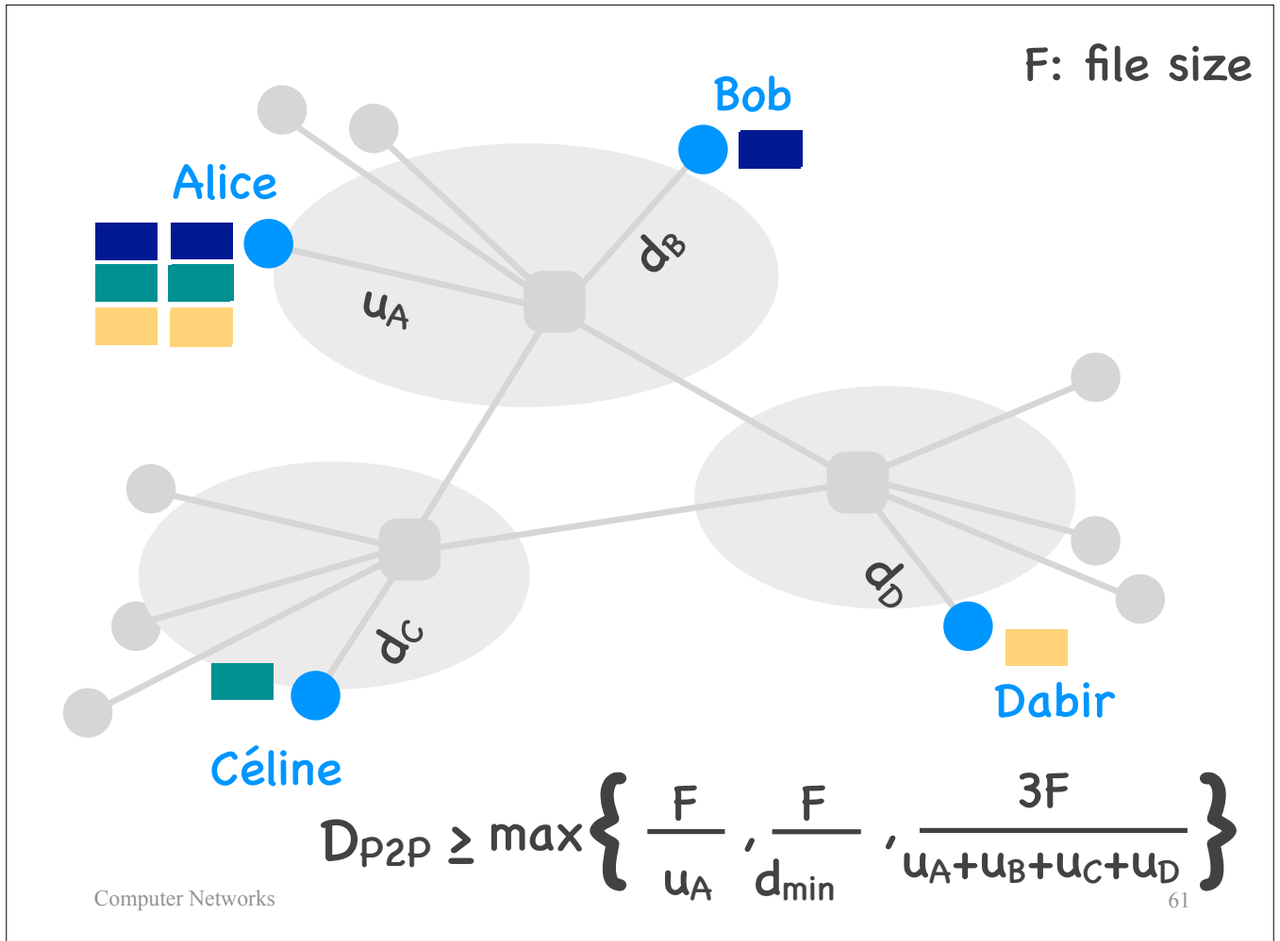


A more compact way to write this formula is this one.



Now, the P2P approach.

Alice does not create copies of the file.
She cuts the file into three pieces.



She sends one piece to each friend.
Then, each friend sends to the other friends the pieces that they are missing.
And this continues until all the friends have all the pieces of the file.

What's the file distribution time?
It seems awfully hard to compute.
So, we are going to be super approximate.

The file distribution time will be at least as large as the max of the following quantities:

- the transmission delay of one copy of the file over the first link; why one copy only? because Alice pushes each bit of the file once over her link;
- then, the transmission delay of one copy of the file over the last link of each friend; why one copy? again, because each friend pulls each bit of the file once over its link;
- there's one more quantity to consider: the transmission delay of three copies of the file over the links of all the peers together.

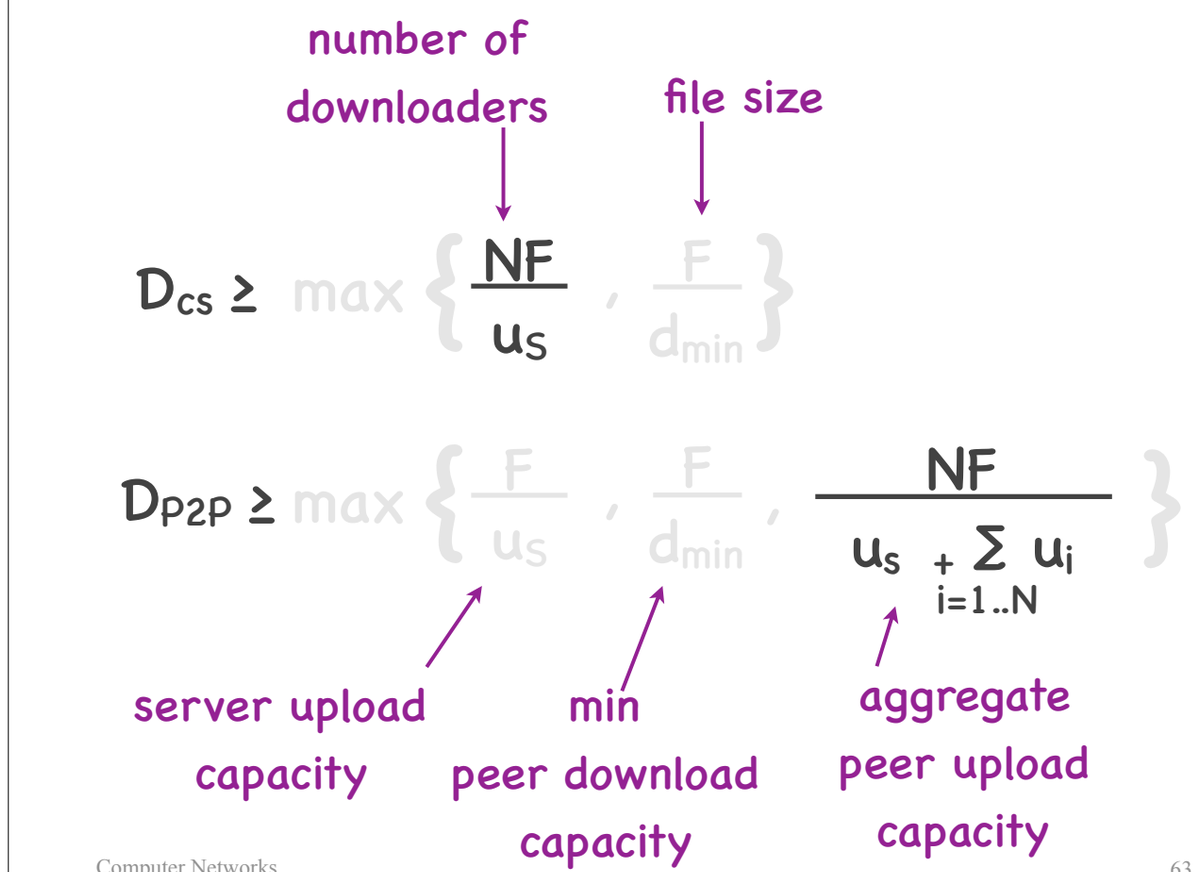
This last quantity captures the essence of P2P file distribution: three copies must be delivered to three friends; so, at some point, each of these 3F bits must be pushed into the network; who pushes all t these bits? It's not just Alice, who originated the file, but all the friends, all the peers together.

$$D_{cs} \geq \max\left\{\frac{3F}{u_A}, \frac{F}{d_{min}}\right\}$$

$$D_{P2P} \geq \max\left\{\frac{F}{u_A}, \frac{F}{d_{min}}, \frac{3F}{u_A+u_B+u_C+u_D}\right\}$$

So: here are the two bounds that we computed for the file distribution time, in client-server approach, and in the P2P approach.

If we replace 3 with N peers…

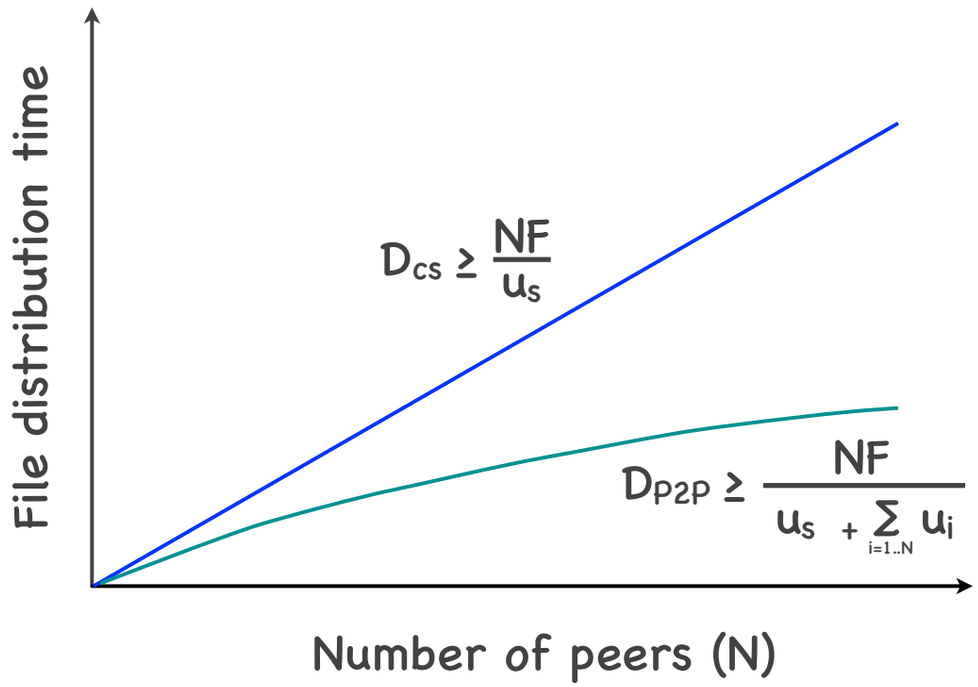


…then we get these bounds.

Let's remember what all these terms are:

- the file size F
- the number of downloaders N
- the server upload capacity u_s (server is the node that originates the file)
- the minimum peer download capacity d_min
- the aggregate peer upload capacity, which is this sum.

As N increases, the terms that include N become the dominant ones, and we can ignore the rest.



If we plug in some values and plot file distribution time as a function N, for the two approaches, we get these curves.

In the client-server approach, file distribution time grows, scales linearly with the number of downloaders N. Of course, since each downloader gets their own copy.

In the P2P approach, file distribution time grows, scalers sub linearly with the number of downloaders N. Which makes sense, because more downloaders require to push more bits into the network, but they also help push these bits into the network.

Scalability (informally)

- Ability to grow
- As the system grows,
  it maintains its properties
  at a reasonable cost

Let’s remember again that scalability…

File distribution

- Client-server: time increases linearly with the number of clients
- Peer-to-peer: time increases sub-linearly with the number of peers
- Peer-to-peer scales better than client-server

How to retrieve content from a P2P file distribution system?

Last topic of this lecture: …

Content

- Set of data files
- Stored in a peer

First of all, a piece of content is a set of data files that are stored in a peer.

Metadata file

- Special file that stores information about the data files
  - file identities
  - (optionally) location information
- May be on a web server or a peer
- BitTorrent: metadata file = .torrent file

Each piece of content is associated with a metadata file.

This is a special file that stores information about the data files, e.g., the identities of the data files.

In a P2P system, the metadata file may be stored in a centralised location, e.g., a web server, or in a peer.

For those of you who use BitTorrent, the metadata file is the .torrent file.

Steps to retrieve content

- (Learn metadata file ID)
- Find metadata file location
- Get metadata file (from web server or peer), read data file IDs
- Find data file locations
- Get data files (from peers)

Here are the basic steps to retrieve a piece of content: …

To complete these steps, you need a mechanism for …

How to find file location?

… finding the location of a file, whether that's a metadata file, or a data file.

In general, there are two mechanisms:

Tracker

- An end-system that knows the locations of the files
  - the IP addresses of the peers that store each file

First option: a tracker.

This is an end-system that knows the locations of the files, meaning, the IP addresses of the peers that store each file.

Distributed Hash Table (DHT)

- An distributed system that knows the locations of the files
  - the IP addresses of the peers that store each file

Second option: a distributed hash table (DHT).

This is a *distributed system* that knows the locations of the files.

Tracker vs. DHT

- Different implementations
  of the same service
  - input: file ID
  - output: IP(s) of peer(s) that have the file
- Tracker is centralized,
  DHT is distributed/decentralized
- You don't need both

A tracker and a DHT are different implementations of the same service.

You can view both of them as a black box that takes as input a file ID and produces as output the IP addresses of the peers that store this file.

A tracker is a centralised implementation of the service,
a DHT is a distributed, decentralized implementation of the service.

In general, you need one of these things, not both.

Steps to retrieve content

- (Learn metadata file ID)
- Find metadata file location
- Get metadata file (from web server or peer), read data file IDs
- Find data file locations
- Get data files (from peers)

Back to the basic steps to retrieve a piece of content.

Steps to retrieve content

- (Learn metadata file ID)
- Find metadata file location
- Get metadata file (from web server or peer), read data file IDs
- Find data file locations
- Get data files (from peers)

Let’s focus on the first three, which are about retrieving the metadata file.

Where is the metadata file?

- Option #1: on a web server
  - you download it from the web server
  - you don't need to learn any ID
- Option #2: on a peer
  - you learn its ID from a web server
  - you learn its location from a tracker or DHT
- BitTorrent: metadata file ID = magnet link
  - e.g., magnet:xt=urn:btih:c12fe1c06bba25…

Where could the metadata file be?

Option #1: On a web server.
In this case, you don't need to learn any ID for the metadata file.
You just google ".torrent file for game of thrones season 1".

Option #2: The metadata file is on a peer.
In this case, you need to learn the ID of the metadata file and learn its location by asking a tracker or a DHT.

For those of you who use BitTorrent, the metadata file ID is the magnet link.

And how do you learn the ID of the metadata file?
You have to get at least that from somewhere outside the P2P system itself.
You typically get it from a web server.
You google "magnet link for game of thrones season 1".

Steps to retrieve content

- (Learn metadata file ID)
- Find metadata file location
- Get metadata file (from web server or peer), read data file IDs
- Find data file locations
- Get data files (from peers)

Once you have retrieved the metadata file, you open it, read the data file IDs, find each data file's location (using a tracker or a DHT), and you retrieve all the data files.

Why use magnet links?

Here's a question for you to think about until Friday: why do people use magnet links to point to metadata files? Why not simply put all the metadata files (the .torrent files) on web servers?

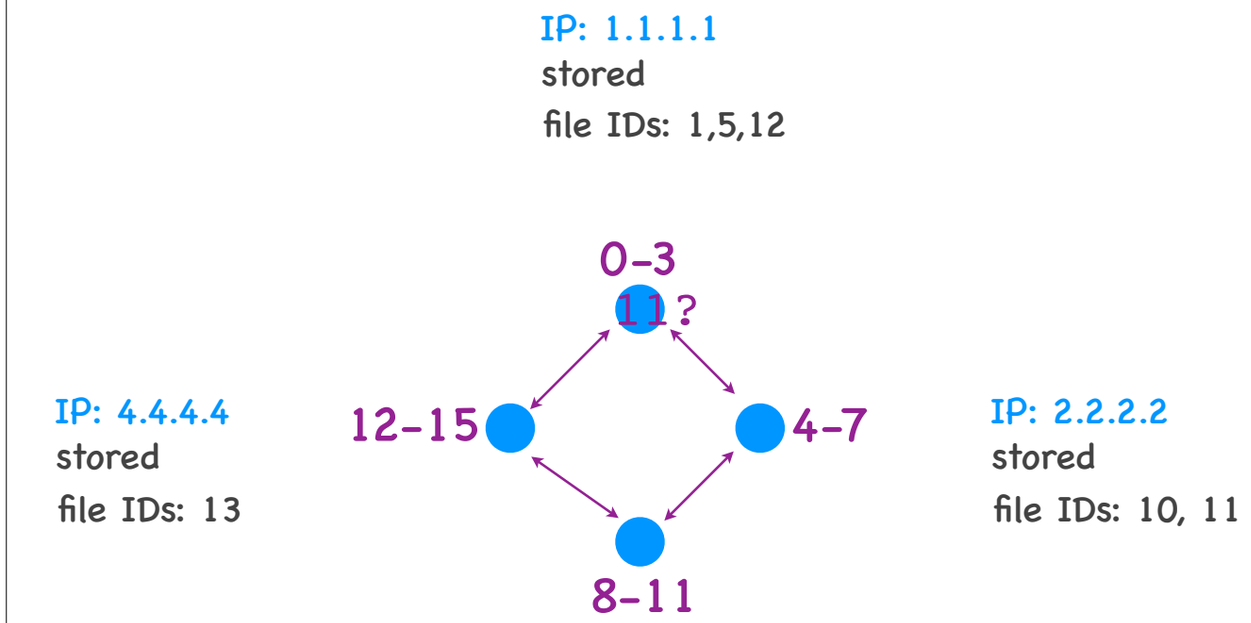
How does a DHT work?

We will close the lecture with a glimpse at how DHTs work.

Simplifying assumption

- We can have only 16 files
- File IDs are from 0 to 15

We will make a simplifying assumption:
there can be only 16 files,
and their file IDs are from 0 to 15.



Suppose this is a P2P system that consists of 4 peers, which have these IP addresses.
Each peer stores some files.
E.g., the peer with IP address 1.1.1.1 stores files with IDs 1, 5, and 12.

The peers partition among themselves the file name space.
This means that each peer “owns” some file IDs.
E.g., the top peer “owns” file IDs 0-3, the second one file IDs 4-7, and so on.

When I say that a peer “owns” a file ID, I do not mean that the peer necessarily stores that file. E.g., the top peer owns file ID 3, but it is the bottom peer that stores that file.

When I say that a peer “owns” a file ID, I mean that the peer knows where that file is stored. E.g., the top peer, who owns file ID 3, knows that this file is stored at the peer with IP address 3.3.3.3.

Moreover, each peer knows the IP addresses of a few other peers,
and it also knows which file IDs those peers own.
In this example, each peer knows the IP addresses of its two neighbors.

Now, suppose the top peer wants to find the file with ID 11.
It knows nothing about this file.
But it knows that its neighbor, the peer on the left, owns file IDs 12-15, and file ID 11 is close to that, so it asks that neighbor.
The peer on the left also knows nothing about this file.
But it knows that its neighbour, the peer at the bottom, owns file IDs 8-11, so it asks that neighbor.

The peer at the bottom owns file ID 11, so it knows that that file is stored at the peer with IP address 2.2.2.2.
So, the peer at the bottom responds to the peer at the top and tells it where to find the file with ID 11.

Basic DHT concepts

- File ID space partitioned:
  each peer "owns" an ID range
- Each peer knows the location
  of the files whose IDs it owns
- Each peer knows its own range
  + the ranges owned by its neighbors

Basic DHT concepts

- The DHT receives requests to locate a file ID
- Each peer forwards the request to the neighbor whose range is closest to the target file ID