Computer Networks

Christian Rinderknecht

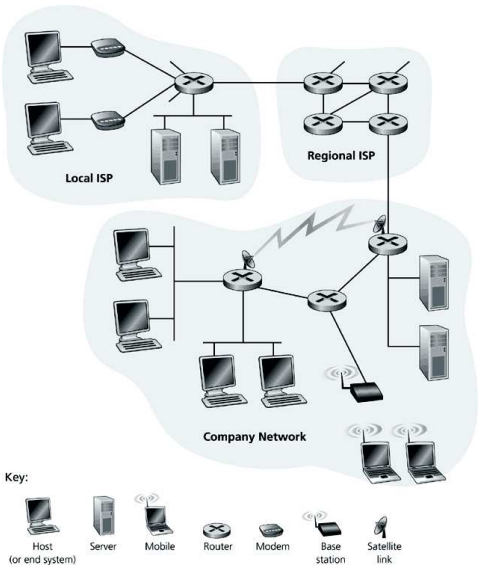
24 October 2008

What is the Internet?

Millions of connected computing devices, called **end systems** (PCs, workstations, servers, PDAs, phones, game consoles etc.), running network applications, i.e., software for accessing and using the network.

End systems are also called **hosts**. Hosts are sometimes divided into **clients** and **servers**.

What is the Internet? (cont)

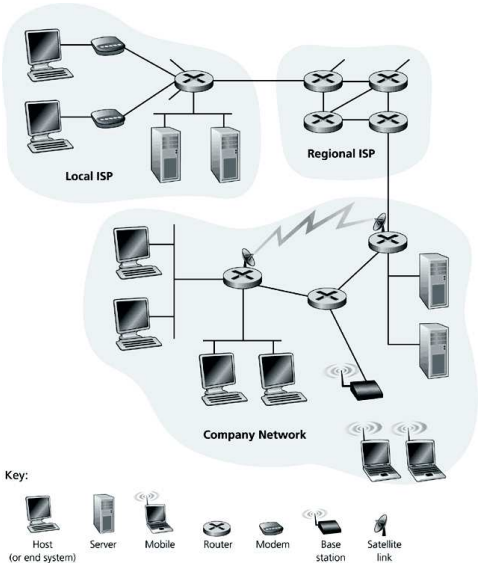


What is the Internet? (cont)

Communication links, e.g., fiber optics, copper wire, radio wave, satellite, connect hosts to each other; the transmission rate is called **bandwidth** (measured in bits per seconds, or "bit/sec").

The **routers** are located between hosts; they receive **packets** (chunks of data) from originating hosts and forward them to their destination hosts.

What is the Internet? (cont)

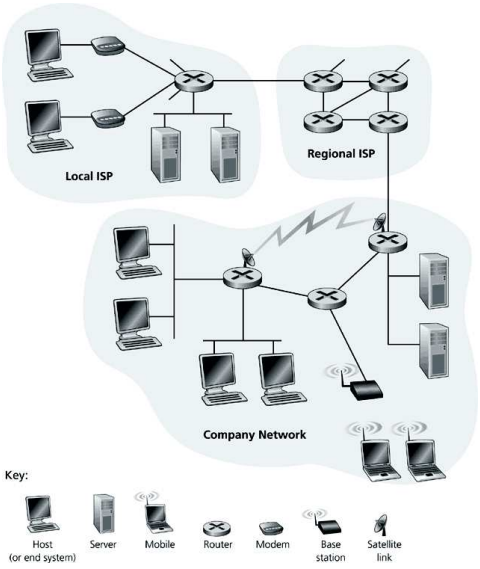


What is the Internet? (cont)

Hosts access the Internet through **Internet Service Providers** (ISP), which are networks proposing many different kinds of connections to the users: dial-up modem, cable modem (DSL), high-speed Local Area Network (LAN) access, wireless access.

There are ISPs in universities, companies and for residents.

What is the Internet? (cont)



What is the Internet? (cont)

ISPs are connected to others ISPs in a hierarchy: at the bottom lie the content providers ISPs and, at the top, the international ISPs (such as UUNet and Sprint) with high-speed routers interconnected with high-speed fiber-optics links.

So, what is the Internet?

- As the name tells it (*Inter*connected *net*works), it is a worldwide system of computer networks: **a network of networks** with a loose hierarchy.
- These networks share the same **protocols**, i.e., ways of representing information (packets) and rules for accepting, refusing or sending messages.

What is the Internet? (cont)

- This network of networks is called **public Internet**.
- Many companies have developed private networks based on the same hosts, links, routers and protocols as the public Internet: they are called **intranets**.
- These intranets are connected to the (public) Internet through **firewalls**, which filter and restrict the information flows in and out.

What is the Internet? (cont)

- Some Internet protocol names are HTTP (for the Web), TCP, IP, FTP (file transfer protocol), PPP (for modem connection), SMTP (e-mails) etc.
- Internet standards are developed by the **Internet Engineering Task Force** (IETF).
- The IETF standards documents are called **Requests For Comments** (RFC).

What is the Internet?/A service view

- The Internet allows applications that inter-operate to run on the hosts. These are called **distributed applications** and include remote login, e-mail, web surfing, instant messaging, audio and video streaming, Internet telephony, distributed games, peer-to-peer (P2P) file sharing, voting, databases etc.
- The network provides services to the distributed applications, i.e., networking-oriented features that programmers can use when they write such an application. Typically there are **connection-oriented services** and **connectionless services**.
- A connection-oriented service is **reliable**: it guarantees that the data is delivered in order and entirely. A connectionless service is **unreliable**: it guarantees nothing about delivery.

What is the Internet?/Protocols

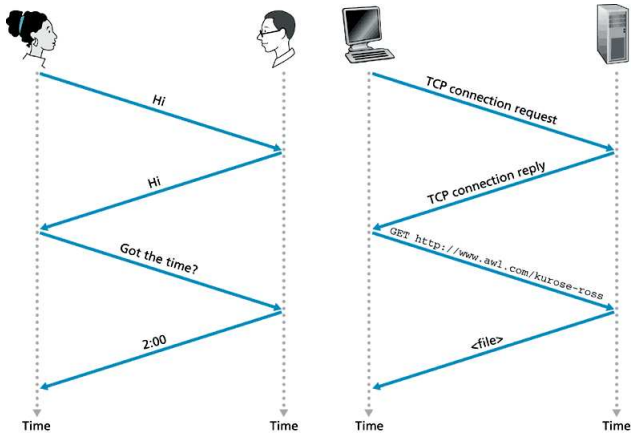
What is a protocol?

- Protocols are part of human relationships. When we meet a friend we start greeting him ('Hi!') and wait for the similar greeting. If it does not come or if some unpleasant words come back, we understand that the communication will not be possible or not good. Otherwise, we go on talking.
- The machine counterpart of this introduction is a connection-oriented service: first the sending application informs the remote application that it wants to communicate (i.e., exchange data). The remote application must acknowledge that before data is sent.

What is the Internet?/Protocols (cont)

- The conversation between friends is similar to the data transmission, from one application to another.
- So, in human protocols, some specific messages are exchanged (e.g., greetings, goodbyes) and some specific actions are taken when messages are received or other events happen.
- Similarly network protocols define format and order of messages exchanged between hosts as well as actions to be taken upon message receipt or transmission.

What is the Internet?/Protocols (cont)



A closer look at the network structure

- The **network edge** is made of the hosts and their applications.
- The **network core** is made of the routers and the protocols that enable the network of networks.
- The **access network** is the communication links.

Plan

1. Computer Networks and the Internet
    - What is the Internet?
    - **The network edge**
    - The network core
    - Network access and physical media
    - ISPs and Internet backbones
    - Delay and loss in packet-switched networks
    - Protocol layers and their service models

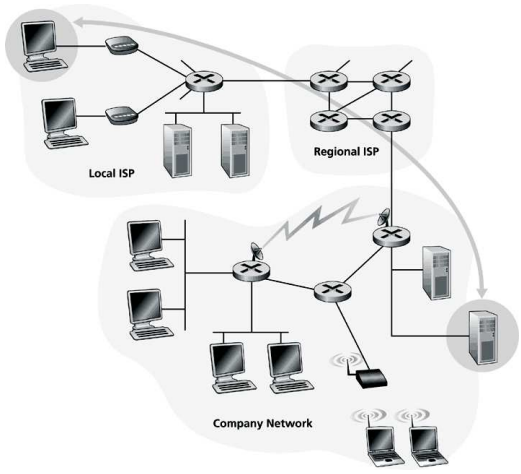
The network edge

End users may interact directly with the host (e.g., a Mac) or indirectly (e.g., a web server).

A **client program** runs on a host that requests and receives a service from a **server program** running on another host. This is the **client/server model**.

In the **peer-to-peer model**, there is little or no use of servers.

The network edge (cont)



The network edge/Connection-oriented service

The goal is to transfer data between two end systems in the following manner:

- First, **handshaking** takes place: the two systems agree on the forthcoming exchange. This is like the 'Hi/Hi (back)' in human protocols. Both hosts set their internal state in accordance, i.e., they record the fact that they are communicating with a known peer. Then data is transmitted.
- This is summarized in figure page 6: the two first messages consist in the handshaking and the two following (`GET` and the response containing the file) are the data communication itself.
- In the Internet the connection-oriented service is the **Transmission Control Protocol** (**TCP**), used by most of the applications (like telnet, SMTP, ftp, http).

The network edge/TCP added-services

The TCP has been designed to carry *more* than connection-oriented service, but also

- **reliability**: the (byte stream) data delivery, in order and in its entirety is guaranteed. As a coarse approximation, reliability is achieved by way of **acknowledgment** and **retransmission**: each time a packet is received, a special packet is sent back to acknowledge the receipt; when such acknowledgment is missing, the sender assumes the packet got lost and retransmits it.
- **flow control**: the sender slows down and avoids overwhelming the receiver by sending too many packets too fast;
- **congestion control**: the sender slows down when the routers start loosing packets because they are congested by a too heavy traffic.

The network edge/Connectionless service

In connectionless services, the goal is still data transfer between hosts but there is no handshaking.

In the Internet, the **User Datagram Protocol** (**UDP**) provides a connectionless service to the applications. This means:

- no reliable transfer (the data can arrive too soon, i.e., when the receiver is not expecting it),
- no flow control,
- no congestion control.

The applications must handle themselves these aspects.
Internet phone and video conferencing, streaming, DNS rely on UDP.

Plan

1. Computer Networks and the Internet
    - What is the Internet?
    - The network edge
    - **The network core**
    - Network access and physical media
    - ISPs and Internet backbones
    - Delay and loss in packet-switched networks
    - Protocol layers and their service models

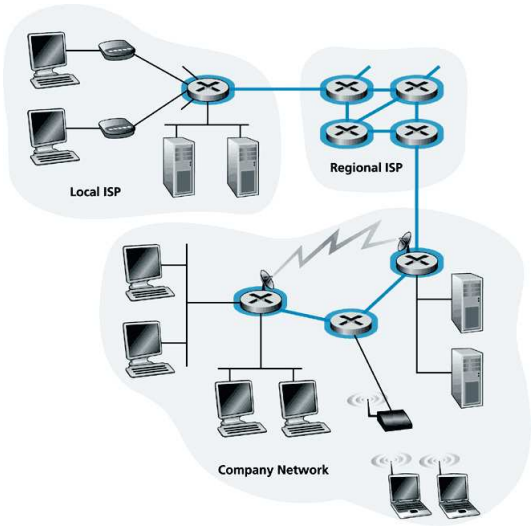
The network core

The network core is a mesh of interconnected routers.

They are of two kinds:

- **Circuit-switching** networks: the resources (buffers, link bandwidth) needed along a path (or **circuit**) to provide a communication between hosts are *reserved* for the duration of the **session**.
- **Packet-switching** networks: the resources are *not reserved* and thus the data may have to wait and are cut into packets.

The network core (cont)



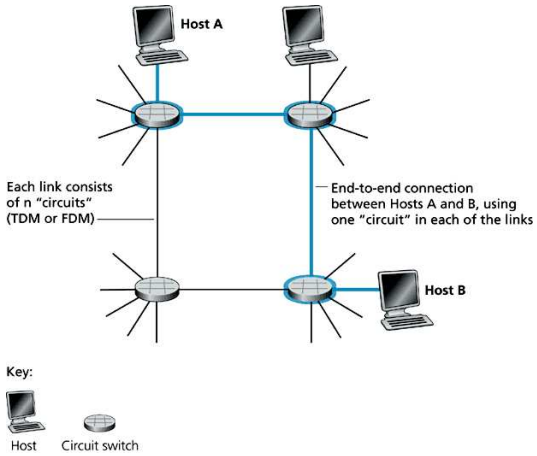
Circuit-switched networks

The **telephone network** is of this kind: even if no one talks on the phone, the resources are still reserved until one hangs up.

Since bandwidth has been reserved, the data (voice) rate is guaranteed.

Circuits are not shared and must be setup first: the routers (called here **circuit switches**) along the path "know" that they are part of a given circuit (i.e., keep a state in memory).

Circuit-switched networks (cont)

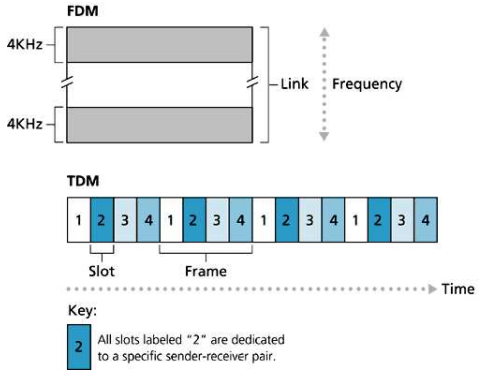


Circuit-switched networks/Multiplexing

Each link divides its data rate equally between all the active circuits it supports. This is called **multiplexing**. There are two kinds of multiplexing.

- **Frequency-division multiplexing (FDM)** divides and *shares the frequency spectrum of the link* among all the connections established across it. In telephony, the bandwidth allocated to a circuit is 4 kHz. FM radio also use FDM.
- **Time-division multiplexing (TDM)** defines a period of time, called **frame**, divides it into equal **slots** which are reserved to a given circuit. In other words, each time slot is periodically repeated and, *during each slot, the full bandwidth is available to the current circuit only.* Modern trend in telephony is to replace FDM with TDM.

Circuit-switched networks/Multiplexing/Figures



Circuit-switched networks/Multiplexing/TDM examples

For example, if a TDM link transmits 8,000 frames per second and each slot allows sending 8 bits, then the transmission rate of a circuit on this link is 64 kb/s (also noted 64 **Kbps**).

Now, suppose

- we want to send a file of size 640,000 bits;
- all links in the network use TDM with 24 slots/frame and have a bit rate of 1.536 Mbps (mega-bits per second);
- it takes 500 ms (milliseconds, also noted **msec**) to establish an end-to-end circuit.

How long does it take to send the file?

Circuit-switched networks/Multiplexing/Solution

1. 24 slots means there are 24 circuits along each link.
2. TDM implies that each circuit has the full bandwidth in turn. Therefore, each circuit has the full bandwidth 1/24 of the time.
3. Thus the rate for each circuit is the link rate by its time using it:

   1.536/24 = 64 kb/s = 64,000 b/s
4. The file is transfered in 640,000/64,000 = 10 seconds.

5. But we have to add the time to setup the circuit, so the total transfer time is

   $10 + 0.5 = 10.5$ seconds.

Note that **the transfer time is independent of the number of links.**

Packet-switched networks

The source breaks the messages into smaller pieces known as **packets**. The packets are directed towards their destination by **routers**, also called **packet switches**.

Packets are transmitted over each link at the *full* transmission rate: *only one packet travels a link at a time.*

Most packet-switches use **store-and-forward transmission**: the switch must receive the entire packet before it starts to retransmit the first bit of it.

Packet-switched networks/Queuing

Each router has multiple links attached to it. For each link, the router has an **output buffer** (also called **output queue**), which stores packets that are about to be transmitted.

If an incoming packet finds the next link busy, it is stored (queued) in the corresponding output buffer until the link is available: this delay is called **queuing delay**.

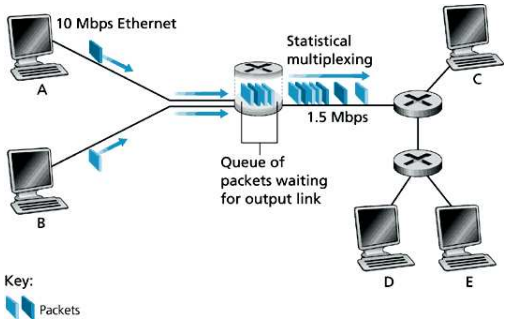
If the queue is already full (because of network congestion), the incoming packet is discarded (dropped): this is a **packet loss**.

Packet-switched networks/Statistical multiplexing

The packets are retransmitted whenever they are available and the output link is free. This leads to a random ordering, called **statistical multiplexing**.

This very different from TDM in circuit switching, where each host gets the same time slot in a revolving frame.

Packet-switched networks/Statistical multiplexing (cont)



Warning! In this figure, several packets for the same connection are drawn on the same link, but, in fact, there is only one at a time.

Packet-switched networks/Delay

Let us consider on a very simple case how long it takes to send a message of L bits on a series of Q links whose rate is R bits/sec.

Assume that there is no queuing delay and no connection establishment.

The packet must be transmitted on the first link: this takes L/R seconds.

Then it must be retransmitted on the following $Q-1$ links, that is to say $Q-1$ times.

Thus the total delay is simply QL/R.

Packet switching versus circuit switching

Let us compare packet switching and circuit switching.

- Opponents to packet switching say that packet switching is not suitable for **real-time services** (e.g., telephone calls and video-conference) because of its variable and unpredictable delays (mainly queuing delays).
- Proponents of packet switching argue that
  - it offers better sharing of the bandwidth than circuit switching,
  - it is simpler, less costly to implement and more efficient.

Packet switching versus circuit switching (cont)

Why is packet switching more efficient?

Let us consider a simple example. Suppose

- users share a 1 Mbps link;
- each user alternates periods of activity and of inactivity:
  - when he is active, he generates data at the constant rate of 100 Kbps;
  - when he is idle, he produces no data.
- each user is active during 10% of the time.

With circuit switching, 100 Kbps must be reserved for *each* user. Thus the link can only support 10 (1 Mbps/100 Kbps) users simultaneously.

Packet switching versus circuit switching (cont)

If there are 35 users, the probability that there are 11 or more simultaneously active users is around 0.0004 (we don't show the computation).

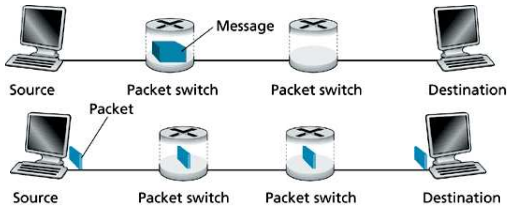
- When there are 10 or less active users at the same time, the total used bandwidth is less or equal to 1 Mbps. This is similar to the circuit switching situation discussed before.
- When there are 11 or more active users, the output queue of the routers will start to grow (this is congestion).

Because the probability of having more than 11 active users is very low, packet switching is almost as good as circuit switching, *while allowing more than three times the number of users* (35 users versus 10).

Message segmenting

We have seen some advantages of packet-switched networks. Now, what about the size of the packets?

If an application message is not cut into smaller pieces (packets), i.e., the message itself if a big packet, this is called **message switching**. Contrast this with small packets in the following figures:



Message segmenting (cont)

When a message is segmented, the packet-switched network **pipelines** the transmission, i.e., portions of the message are transmitted in parallel by the source and the two packet switches.

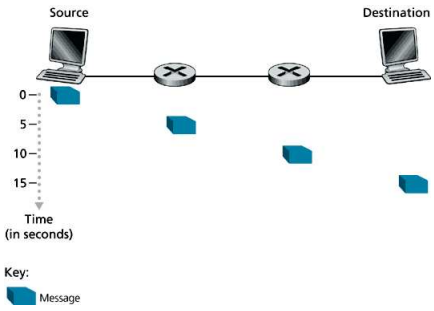
The advantage of message segmentation with packet switching is that the end-to-end delay is lower than with message switching. Let us see why.

Consider a message of size $7.5 \cdot 10^6$ bits, three links of rate 1.5 Mbps connecting two hosts and assume that there is no congestion.

How much time is required to send and receive the message with message switching?

Message segmenting (cont)

This is what happens:



It takes 15 seconds (3 links $\times$ $7.5 \cdot 10^6$ bits/1.5 Mbps). Even a bit more, in fact, due to store-and-forward delays.

Message segmenting (cont)

Now suppose that the source breaks the message into 5,000 packets.

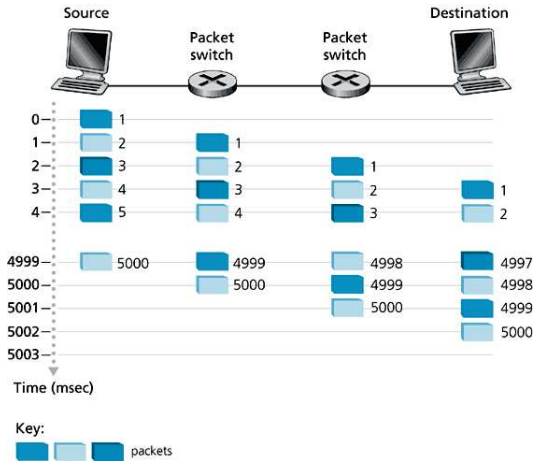
Then each packet is 1,500 $(7.5 \cdot 10^6/5,000)$ bits long.

Assume that there is no congestion.

How long does it takes to move the 5,000 packets to destination?

The next figure shows what happens.

Message segmenting (cont)



Message segmenting (cont)

As shown in the previous figure, the first packet reached the first switch after 1 ms. The second packet reaches it after 2 ms etc.

But, while the first packet is moving from the first switch to the second, the second packet is *at the same time* traveling from the source to the first switch.

The last packet reaches then the first switch after 5,000 ms = 5 seconds. But it still has to cross two more links, therefore the total delay is 5,002 ms (in fact, a little bit more due to store-and-forward delays).

This is much less than with message switching!

The reason is pipelining: once the first packet reaches the destination, the source and the two switches are transmitting simultaneously.

Message segmenting (cont)

There is another interest in message segmentation.

Bit errors can be introduced into the packets, when moving along the networks.

When a switch detects a bit error, it discards the whole packet.

- If the entire message is a packet (i.e., we do message segmenting), then the entire message is discarded.
- If the packets are small, only a small part of the message, corresponding to the erroneous packets, has to be retransmitted.

Message segmenting (cont)

One drawback of segmentation is that the packets do not contain only the application data but also some control information, called **header**, that helps routing.

This situation is comparable to sending mail: the envelope contains (written on itself) the control information for routing and the letter contains the application data.

If we cut our message into pieces much smaller than the envelopes, then the postal service has to carry a heavier load of envelopes than of messages!

Consequently, the price would not be the same.

Packet forwarding in computer networks

Packet-switched networks can be divided in two kinds:

- **Datagram networks** routers forward packets according to host destination **address**. Similar to a home address, a network address is a unique number on the network. The Internet is a datagram network.
- **Virtual circuit networks** routers forward packets according to virtual circuit numbers. X.25 and ATM (Asynchronous Transfer Mode) use such numbers.

Virtual circuit networks

A **virtual circuit** (**VC**) consists of

- a path, i.e., a series of links and switches;
- virtual circuit numbers (VC numbers, in short), one for each link along the path;
- a VC number translation table in each switch along the path.

Once a VC is established, the source can send packets into the VC with the appropriate VC number.

Because a VC has a different number on each link, the switches must replace the VC number of each incoming packet with a new one.

This new number is in the translation table of the switches.

Virtual circuit networks (cont)

Example. The numbers in the figure at the next slide are **interface numbers**.

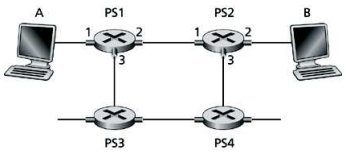
They are local to each switch.

Assume a packet starts from host A and must arrive to B: it has to cross three links.

It begins with the VC number of the first link, say 12, and enters PS1 at the interface 1.

There, its VC number has to be replaced by another one, say 22, and the packet must go out through interface 2 (towards PS2).

Virtual circuit networks (cont)



Virtual circuit networks (cont)

The VC number translation table at switch PS1 looks like

In interface	In VC#	Out interface	Out VC#
1	12	2	22
...	...	...	...

The network (actually, the switches) must maintain a **state information** about the VC.

Even if the VC numbers are all the same, the switches still must associate interfaces and VC numbers.

Each time a new circuit is created, a new entry must be added to the table of the switches along the path. Dually, each time a connection is released, the entries must be removed.

Datagram networks

Datagram networks are similar to postal services.

Each host has a unique address in the network.

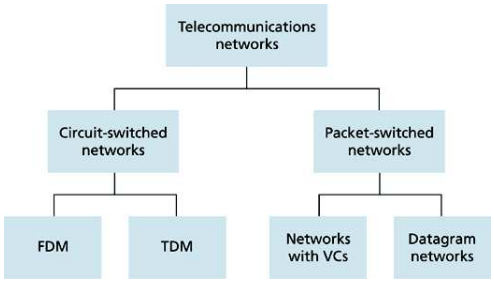
Elements in the address are hierarchical, in order to simplify the routing. This is like putting on an envelope country, city, town, district, street name, building number, apartment number.

We shall discuss later the packet structure.

For now on, remember that a datagram network do not maintain connection-related information in the routers: these do not know which connection they support.

Network taxonomy

Here is the taxonomy of the networking concepts we reviewed:



Warning! *This picture does not include the service view of the networks.* A datagram network is neither a connectionless nor a connection-oriented network. It can provide both services to different applications, through TCP or UDP.

Plan

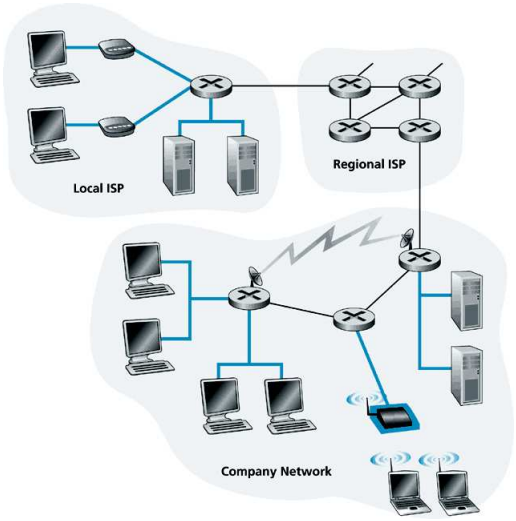
1. Computer Networks and the Internet
    - What is the Internet?
    - The network edge
    - The network core
    - **Network access and physical media**
    - ISPs and Internet backbones
    - Delay and loss in packet-switched networks
    - Protocol layers and their service models

Network access and physical media

The network access is the physical media that connect an end system to its **edge router**, which is the first router on the way to another end system.

The access links in the facing picture are in thick blue.

Network access and physical media (cont)



Network access and physical media

Network access can be loosely classified into three categories:

- **Residential access**, connecting home end systems into the network;
- **Company access**, connecting end systems in a business or educational institution in the network;
- **Mobile access**, connecting mobile end systems into the network.

Note that these categories are not tight, for instance a company may use access technology we ascribe to residential access.

Network access/Residential access

Residential access refers to connecting a home end system, say a computer, to an edge router.

In Europe, a common way is the **dial-up modem** over a wired telephone line. The modem converts the digital output of the computer into the analog format of the phone line, which is a **twisted-pair copper wire** (same as a phone cable). On the other side of the phone line, at the ISP, there is another modem which converts back the analog signal into digital signal which is directed to the edge router.

The data rate can be up to 56 Kbps, with hardware compression and if the telephone line is of good quality.

Disadvantages are a slow data rate for nowadays applications and the fact that the phone line is entirely devoted to the modem.

Network access/Residential access (cont)

Other access technologies are **digital subscriber line** (**DSL**) and **hybrid fiber coaxial cable** (**HFC**).

DSL is a new modem technology running over existing twisted-pair telephone line. But by shortening the distance from home to the ISP, the data rate is much higher. The rate is not symmetric: the rate from the ISP to home (called **downstream**) is higher than from home to the ISP (called **upstream**). Indeed, the standard assumes that the user wants more probably to get information than produce data.

In theory, DSL can provide rates of more than 1 Mbps from ISP to home and more than 1 Mbps from home to ISP, but in practice the real rates are much lower.

Network access/Residential access (cont)

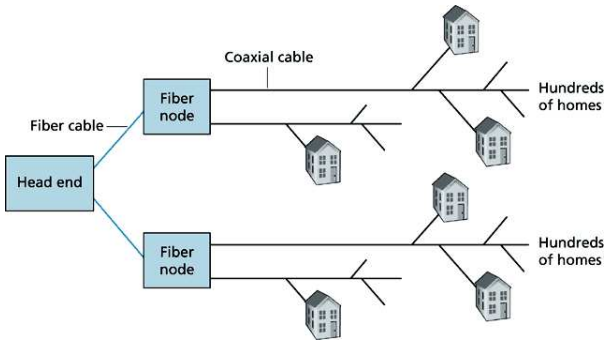
DSL uses frequency division multiplexing (FDM), see page 11. In particular, DSL divides the communication link into three non-overlapping frequency bands:

- a high-speed downstream channel, in the 50kHz-1MHz band;
- a low-speed upstream channel, in the 4kHz-50kHz band;
- an ordinary two-way telephone channel, in the 0-4kHz band.

This allows to keep the same link available for conference calls while being connected to the network.

Network access/Residential access/HFC

HFC are extensions of the existing cable network for television broadcasting.



Network access/Residential access/HFC (cont)

As with DSL, HFC requires special modems, called **cable modems**. This is an external device that is connected to the home computer through a **10-BaseT Ethernet** port (discussed later).

Cable modems divide the HFC network into downstream and upstream channels. As with DSL, the former is faster than the latter.

There is an important difference with DSL: HFC is a **shared broadcast medium**.

Network access/Residential access/HFC (cont)

Every packet sent by the head end is sent (on the downstream channel) to *all* the connected homes (this is broadcasting, a useful feature for television).

So if users download at the same time, the actual rate is slowed down. But if they are surfing the web, the pages will be likely to arrive at full speed, since it is improbable they all click at the same time.

Since the upstream channel is also shared, two packets sent from different homes at the same time will collide (but there is no broadcasting on the upstream channel).

Network access/Company access

On corporate and university campuses, a **local area network** (**LAN**) is used to connect an end system to the edge router. This means that the end system has to be connected first to the LAN.

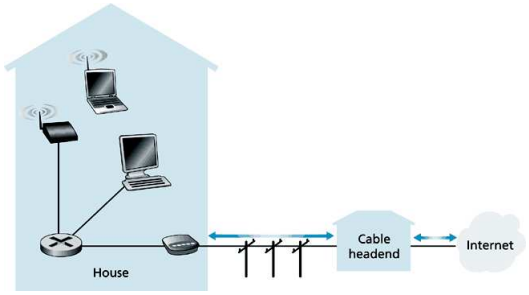
The most access technology to do so is the **Ethernet**. It uses either twisted-pair copper wire or coaxial cable with each other and with an edge router. The rates vary depending on the version: 10 Mbps, 100 Mbps, 1 Gbps or 10 Gbps.

Like HFC, Ethernet uses a shared medium, so end users share the transmission rate of the LAN.

There is now **switched Ethernet**, which uses multiple twisted-pair Ethernet segments connected at a special switch.

Network access/Mobile access

The most popular wireless access to the Internet is through a **wireless LAN** (or **Wi-Fi**). Mobile users are connected to a radio **base station**, also called **wireless access point**, which is itself connected to the Internet. The IEEE 802.11b provides a bandwidth of 11 Mbps.



Plan

1. Computer Networks and the Internet
    - What is the Internet?
    - The network edge
    - The network core
    - Network access and physical media
    - **ISPs and Internet backbones**
    - Delay and loss in packet-switched networks
    - Protocol layers and their service models

ISPs and Internet backbones

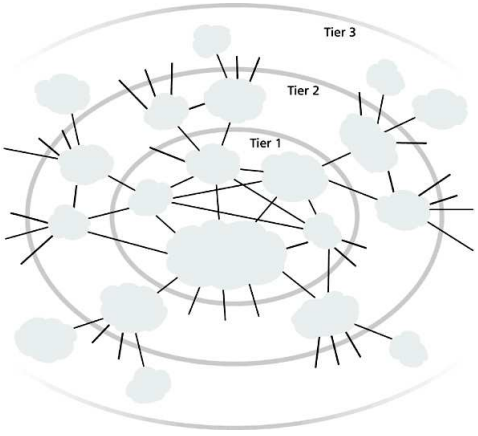
You may recall that ISPs are interconnected in a hierarchy. The most important level in this structure is the **backbone** or **tier-1 ISP** (level one is the top of the hierarchy).

Internet backbones are usual ISPs but

- its links speed is from 662 Mbps to 10 Gbps;
- it is directly connected to *each* other backbones;
- it is connected to a large number of tier-2 ISPs and other customers;
- it has an international coverage.

Hence their routers must be able to forward packets at a very high rate.

ISPs and Internet backbones (cont)



ISPs and Internet backbones (cont)

A tier-2 ISP has typically a regional or national coverage (depends on the size of the country, actually).

Thus, in order to reach a large portion of the Internet, a tier-2 ISP needs to route traffic through one of the backbones to which it si connected.

In this case, the tier-2 ISP is said to be a **customer** of the backbone and the backbone is said to be a **provider** to the tier-2 ISP.

Tier-2 ISP can be directly connected to each other. In general, when two ISPs are connected directly, they are said **to peer** each other.

Within an ISP, the points at which it peers another ISP are called **Points of Presence** (**POP**) and are a router or a group of routers.

Plan

1. Computer Networks and the Internet
    - What is the Internet?
    - The network edge
    - The network core
    - Network access and physical media
    - ISPs and Internet backbones
    - **Delay and loss in packet-switched networks**
    - Protocol layers and their service models

Delay and loss in packet-switched networks

Let us come back to the packets that travel a packet-switched network like the Internet.

When a packet crosses a link and a router, it suffers different kinds of delays:

- **nodal processing delay**,
- **queuing delay**,
- **transmission delay**,
- **propagation delay**.

All together, they give the **total nodal delay**.

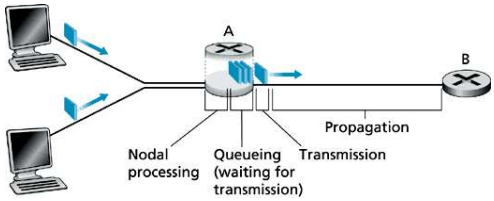
Let us now look with more attention to each of these delays, because it is necessary if we want to acquire a deep understanding of packet switching.

Delay and loss in packet-switched networks/Types of delay

- **Processing delay** is the time needed to examine the packet's header and find where to direct the packet. Also it can include the time for checking for bit-level errors in the packet.
- **Queuing delay** happens when some earlier-arriving packet is already waiting in the queue, else it is 0.

- **Transmission delay** is the time needed to "push" the packet out on the link (size of the packet/rate of the link).
- **Propagation delay** is the time needed to propagate a signal on the link (distance A-B/signal speed).

Delay and loss in packet-switched networks/Types of delay



Delay and loss in packet-switched networks/Types of delay

What is the difference between transmission delay and propagation delay?

Transmission delay depends on the packet's length (measured in bits) and the bit rate of the link (measured in bits per seconds), but has nothing to do with the distance between two routers.

Propagation delay is the time it takes a bit to propagate from one router to the next; it is a function of the distance between to adjacent routers, but has nothing to do with the packet's length or the transmission rate of the link.

Queuing delay and packet loss

The delay which can be the highest and is the most difficult to assess is the queuing delay.

The main reason is because it depends from packet to packet, contrary to the other kinds of delays. For instance, if ten packets arrive at very short intervals to an empty queue, the first one will suffer no queuing delay but the tenth will have to wait in the queue the other have been retransmitted.

That is why the usual tool to evaluate the queuing delay are **statistical models**, including average queuing delay and the probability that the queuing delay exceeds some specified value.

Queuing delay and packet loss (cont)

The queuing delay depends on

- the rate at which traffic arrives at the queue,
- the transmission rate of the (out-going) link,
- the nature of the incoming traffic:
  - is it periodical?
  - does it arrives in bursts?

Queuing delay and packet loss (cont)

Let us assume that the average rate at which packets reach the queue is a. It is measured in packets/sec.

Let R be the transmission rate, measured in bits/sec, which is the rate at which bits are pushed in the out-going link.

Suppose that all packets are L bits long.

Then the average rate at which bits arrive at the queue is La.

Assume the queue is unbounded.

The ratio La/R, called the **traffic intensity**, plays an important role in estimating the growth of the queue.

Queuing delay and packet loss (cont)

If $La/R > 1$ then the flow of incoming bits exceeds the capacity of the out-going link: the queue will grow forever. So the most important rule here is: *design your network so that the traffic intensity is no greater than 1.*

If $La/R \leqslant 1$ then the nature of the arriving traffic impacts the queuing delay. For example, if packets arrive periodically, one each L/R second, then every packet will arrive at an empty queue and there will be no queuing delay (this is the best case).

If packets arrive in bursts, i.e., if the number of packets per second varies, the delay can be significant (especially if the variation is not linear).

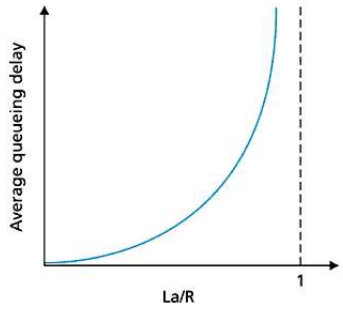
Queuing delay and packet loss (cont)

In practice, periodic arrival or linear bursts are improbable: the arrival is *random*.

Thus the quantity La/R is not enough to fully characterise the queuing delay, but it helps you to grasp the concept.

The curve in the facing column gives you more intuition on the average queuing delay. The important thing to note is that a little variation on La/R, when this quantity is close to 1, may lead to a huge augmentation of the average delay.

Queuing delay and packet loss (cont)



Queuing delay and packet loss (cont)

In our previous presentation, we assumed an unbounded queue, i.e., allowing an infinite number of packets to remain in the queue. That is why the traffic intensity could approach 1 at any distance.

But, in practice, routers have a finite memory, so a packet can arrive and find a full queue. In this case the packet is dropped, i.e., it is lost.

The packet losses increase as the traffic intensity increases.

Therefore, performance at a router is not only measured in terms of traffic intensity but also of packet loss.

Queuing delay and packet loss (cont)

Until now we considered the delay at one node (a router). What about the **end-to-end delay**, that is the cumulated delay of each node from the source to the destination? Assume there are N routers along the path. Let us suppose that there is no congestion (queuing delay is negligible).

- Let d_{proc} be the processing delay at each node (including the source),
- d_{prop} the propagation delay of each link,
- the transmission rate of each node is R bits/sec.

The nodal delays accumulate and give an end-to-end delay

$$d_{\text{end-end}} = N(d_{\text{proc}} + d_{\text{trans}} + d_{\text{prop}})$$

where $d_{\text{trans}} = L/R$, where L is the packet size.

Delays and routes in the Internet

Traceroute is a simple program that can run on any Internet host. It is given a destination host and this program sends special packets to this destination. Along their way, the packets pass through several routers, which send back to the source a message containing their name (if any) and their address, called **IP address** and whose structure we will discuss later.

This allows Traceroute to reconstruct the path and to show it to the end-user.

Imagine there are N routers along the path. Then the source will send N special packets, all addressed to the destination. They are numbered from 1 to N and sent by increasing numbers. The n-th router receives the n-th packet, destroys it and sends an identification message back to the source.

Delays and routes in the Internet (cont)

The source records the elapsed time between the moment it sent a packet and the moment the corresponding return message is received, coming from a router. This delay is called **round-trip delay**.

Traceroute repeats the experiment three times, because the round-trip delay can vary due to queuing delays, so the user gets an idea of the delay variation.

The web site `http://www.traceroute.org` provides a list of links to web sites, classified by countries, which provide an interface to Traceroute.

This way you can experiment by giving the address or name of a destination and the interface will report the path from the host running the web site to the destination host, as given by Traceroute.

Delays and routes in the Internet (cont)

Here is an example of output of Traceroute from a host in the Faroe Islands (Denmark) to a host in France (at INRIA):

```
traceroute to pauillac.inria.fr (128.93.11.35), 30 hops max, 38 byte packets
 1  L4-0-0.bone.olivant.fo (212.55.32.1)  0.977 ms  0.612 ms  0.766 ms
 2  feth1-0-0.utland1.bone2.olivant.fo (212.55.32.98)  1.108 ms  1.234 ms  1.601 ms
 3  ser4-0.45M.ldn2nxi2.ip.tele.dk (195.215.170.85)  30.455 ms  30.676 ms  29.567 ms
 4  ge1-2-2.1000M.ldn2nxg1.ip.tele.dk (195.249.13.121)  30.745 ms  30.134 ms  31.431 ms
 5  pos6-0.2488M.asd9nxg1.ip.tele.dk (195.249.2.134)  37.281 ms  37.332 ms  36.983 ms
 6  Ge7-1.AMSBB1.Amsterdam.opentransit.net (193.251.254.9)  37.103 ms  37.334 ms  37.726 ms
 7  * * *
 8  * * *
 9  * * *
```

Delays and routes in the Internet (cont)

```
10  P14-0.PASCR2.Pastourelle.opentransit.net (193.251.128.105) 58.415 ms  57.089 ms  57.764 ms
11  193.51.185.2 (193.51.185.2)  57.802 ms  61.880 ms  58.485 ms
12  inria-g3-1.cssi.renater.fr (193.51.180.174)  60.999 ms  58.497 ms  57.900 ms
13  royal-inria.cssi.renater.fr (193.51.182.73)  58.860 ms  63.028 ms  58.961 ms
14  193.48.202.2 (193.48.202.2)  63.017 ms  58.502 ms  57.889 ms
15  rocq-gw-bb.inria.fr (192.93.1.100)  60.721 ms  61.437 ms  61.078 ms
16  pauillac.inria.fr (128.93.11.35)  60.759 ms  62.540 ms  61.703 ms
```

The domain extensions (`.fo`, `.dk`, `.fr`) help you to follow the packets along the corresponding countries. Note that some routers have no name (only address). Routers 7, 8 and 9 do not send back their identification messages, that is why Traceroute prints an asterisk for each try.

Plan

1. Computer Networks and the Internet
    - What is the Internet?
    - The network edge
    - The network core
    - Network access and physical media
    - ISPs and Internet backbones
    - Delay and loss in packet-switched networks
    - **Protocol layers and their service models**

Layered architecture

Until now we have been presenting the different components (switches, hosts, links), their relationships, different switching policies and different protocols, or services.

We need now a model to describe the **architecture** of the services provided by the Internet.

A useful analogy is an airline system.

One way to guess a structure of it, is to describe the actions taken to make a trip: first we purchase a flight ticket, then we go to airport, we give our luggage to be checked, then we board the plane as the baggage is loaded, then we take off, the plane travels to destination, lands, we get off the plane, our baggage is unloaded, we claim it and we complain to the ticketing service if the trip was bad.

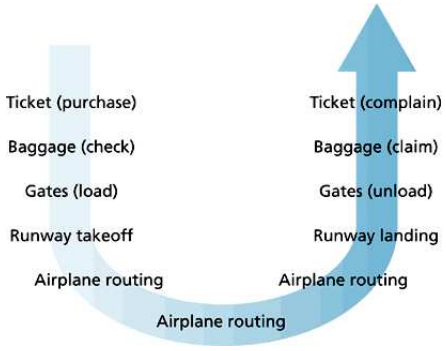
Layered architecture (cont)

The figure in the next slide summarises the series of actions undertaken to make an airplane travel.

It appears that each kind of activity, based on a functionality, in the departing airport has a corresponding counterpart in the arrival airport.

This suggests that we can look at the services in a *horizontal manner*, i.e., as belonging to same levels.

Layered architecture (cont)



Layered architecture (cont)

The following picture changes the focus according to the horizontal point of view on the service usages we described so far. The structure is divided into **service layers**.

Ticket (purchase)	Ticket (complain)	**Ticket**
Baggage (check)	Baggage (claim)	**Baggage**
Gates (load)	Gates (unload)	**Gate**
Runway takeoff	Runway landing	**Takeoff/Landing**
Airplane routing	Airplane routing	**Airplane routing**
Airplane routing		

Layered architecture (cont)

This layered architecture is of great value because it allows to reason on a specific part of the system, just by considering the upper layer (to which the current layer is a provider) and the lower layer (to which the current layer is a customer).

For instance, this model allows to change the service of one layer in a way that is not noticed by the surrounding layers.

The way a service is achieved internally can be changed without changing its observable behaviour (like reorganising the workflow of the staff at the airport gates: this does not change anything outside the service).

It is even possible sometimes to change some properties of the service that are not noticed by the surrounding layers, like imposing that boarding

should be done depending on the height of the passengers: this is not noticed by the baggage check and the runway takeoff services.

Layered architecture (cont)

Coming back to networks services, these are also structured into layers. The input and output behaviour of one layer (service) is the **protocol**. The way this behaviour is achieved internally to the layer is the **protocol implementation**.

As we just saw, these are different notions: you can change the protocol (as imposing more constraints on the output, like sorting by height the passengers), or independently change its implementation (as reorganising the staff at the gates).

Notice also that a protocol in a given layer is implemented by remote entities of the network, just as the services in the airplane system were *distributed* between the departing and arriving airports.

Layered architecture (cont)

In a network, the protocols can either be implemented by hardware or software, or even by a mixture of hardware and software.

Each implementation of layer n communicates with its peer at the same level n by exchanging messages whose structure is specific to this level, called **n-layer protocol data units** or, in short, **n-PDUs**.

On one entity, all the protocols taken as a whole is called the **protocol stack**.

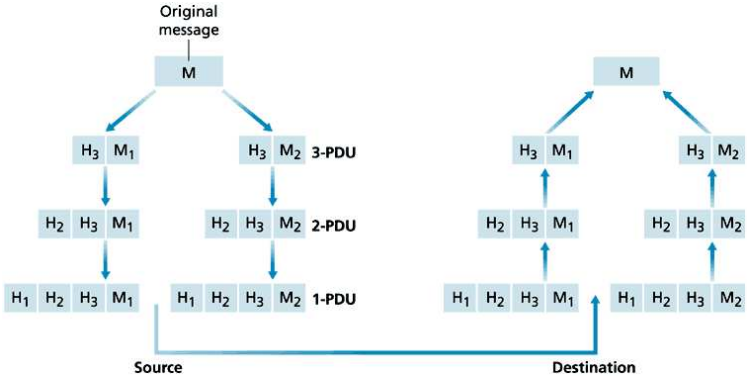
Layered architecture (cont)

When the layer n of host A sends an n-PDU to layer n of host B, it passes the PDU to the $n-1$ layer of host A and lets it deliver the n-PDU to the layer n of host B. Thus layer n is said to *rely* on layer $n-1$ for achieving the delivering, and, in turn, layer $n-1$ *offers services* to layer n.

Layer n is an **upper layer** with respect to layer $n-1$, which is then a **lower layer**.

Layered architecture (cont)

Let us give more insight about protocol layering. Consider a network with four layers in the following figure.



Layered architecture (cont)

The application message, M, is cut into two packets, M_1 and M_2. They are both sent to the layer below, which is layer 3. This layer does not know how to interpret the contents of M_1 and M_2, and does not care about it. His role is to add a **header** H_3 to each packet, whose structure is specific to layer 3, and then passes them to the layer below.

Layer 2, similarly, does not know the structure of the incoming 3-PDU, and does not care. It just adds a header H_2 specific to layer 2. The next layer behaves similarly.

The remote protocol stack works symmetrically, as the message goes up. Each level reads the outermost header (supposed to correspond, by construction, to the current level) and removes it.

Finally, the application message is reassembled back into M.

Layered architecture (cont)

In order for one layer to **interoperate** with the layer below it, the **interfaces** between the two layers must be precisely defined.

An interface contains the format of the PDU as well as the kind of functionality a layer provides (you can think to the types of functions in programming languages).

This allows to improve the implementation of a service to be unnoticed from adjacent layers, as long as the interface remains the same (this is similar to changing the code of a function, while keeping its type).

Layered architecture (cont)

In a computer networks, each layer may perform one or more of the following general tasks:

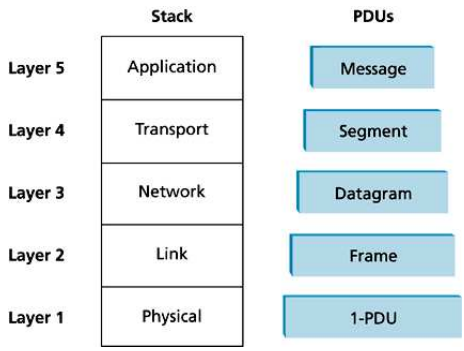
- **Error control**, which makes the logical channel (i.e., from one peer to another, at the same stack level, while ignoring the lower levels) more reliable.
- **Flow control**, which avoids overwhelming a slower peer with PDUs.
- **Segmentation and reassembly**, which divides data at the source and reassembles the pieces at the destination.
- **Multiplexing**, which allows several high-level sessions (i.e., full runs of a given service instance) to share a single lower-level channel.
- **Connection setup**, which provides handshaking with a peer.

Layered architecture (cont)

The **Internet protocol stack** is made of five layers: the physical layer, the data link layer, the network layer, the transport layer and the application layer (bottom-up).

It is common usage to give a name to the four upper layers, instead of n-PDU. They are, from data link layer to application layer: **frame**, **datagram**, **segment**, **message**.

Layered architecture (cont)



Layered architecture (cont)

The **application layer** supports network-oriented programs and may include many different protocols, like HTTP to support the Web, SMTP to support e-mail and FTP to support file transfer.

The **transport layer** provides the service of transporting application-layer messages between the client and the server. In the Internet, there are two protocols at this level: TCP and UDP.

The **network layer** routes datagrams from one host to the other. It has two main components. One defines the fields of the datagrams as well as how the hosts and the routers acts on these fields: this is the famous IP protocol. Another one contains routing protocols, which determines the paths between sources and destinations.

Layered architecture (cont)

The **link layer** is in charge of moving a packet from one router to another. By contrast, the network layer moves a packet along the whole path, not just two switches.

At each node, the network layer passes the datagram to the link layer, which delivers the corresponding frame to the next node. At this node, the frame is passed up to the network layer again.

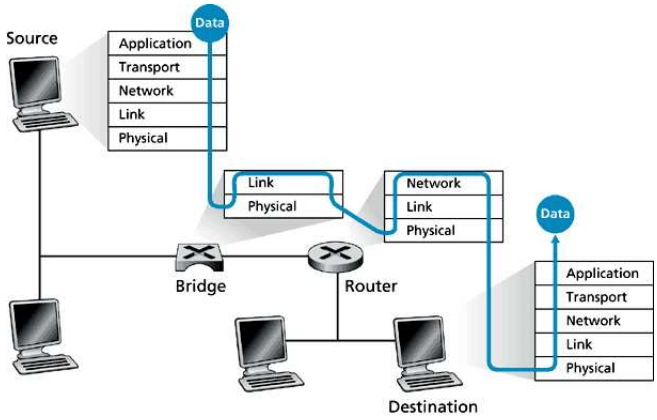
This is because the services provided by the link layer depends on the specific link protocols, which may be different for the incoming link and the out-going link (hence the frame must go up to the network layer at each node). For instance, one link can use PPP and the next one Ethernet.

Layered architecture (cont)

The **physical layer** is in charge of moving bits along a link, by propagating electromagnetic signals along a link. As with the link layer, the protocols at this layer are link-dependent and also depend on the actual transmission rate of the link.

For instance, Ethernet has many physical layer protocols: one for twisted-pair copper wire, another for coaxial cable, another for fiber optics etc. This is because in each case the way to move a bit across the link is different.

Layered architecture (cont)



Bridges are a special kind of switches (they do not support IP).

Plan

- **Application layer**
  - Application layer protocols
  - The Web and HTTP
  - File transfer: FTP
  - Electronic mail in the Internet
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Application layer

The network applications are the reason why we use the Internet, because they are the programs we interact with to connect our host to another host on the network.

There are many popular network applications:

- electronic mail, remote access to computers, file transfers, newsgroups, text chats (in the 1980's);
- Web (mid-1990's);
- instant messaging, peer-to-peer (end of 1990's).

Plan

- Application layer
  - **Application layer protocols**
  - The Web and HTTP
  - File transfer: FTP
  - Electronic mail in the Internet
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Application-layer protocols

A network application is made of several distributed software components, each of them running on different hosts.

Precisely, we do not say that programs communicate with each other, because programs are basically texts. The execution of a program gives birth to the notion of **process**. Processes are the communicating entities.

Network applications use application-layer protocols which define the format of the exchanged messages and the actions undertaken by the hosts on the receipt and sending of given messages.

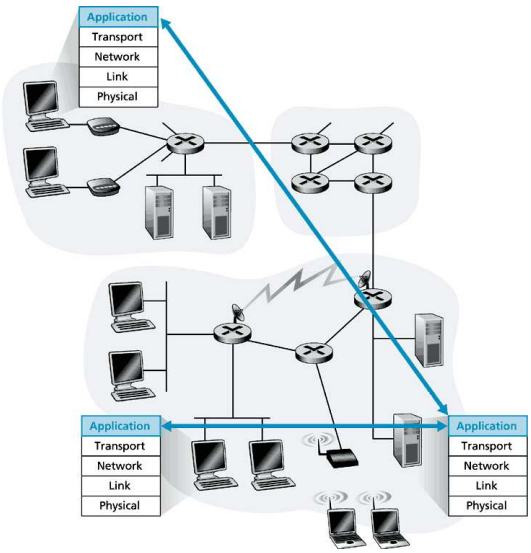
Application-layer protocols (cont)

The figure in the next slide shows the **protocol stacks** at each communicating host.

Applications are processes which interact, within the same layer, with the **operating system** and the user, and use the transport layer services for communicating across the network.

This behaviour is depicted using solid blue arrows.

Application-layer protocols (cont)



Application-layer protocols (cont)

Keep in mind the difference between a **network application** and an **application-layer protocol**:

- a network application is a process;
- an application-layer protocol is the definition of data formats and of the behaviour of communicating network applications.

In other words, a network application **implements** an application-layer protocol, but also do more, like offering to the user a graphical interface etc.

Some application-layer protocols are POP and SMTP for e-mail. The mail client, like Outlook, Thunderbird or Evolution, is a network application.

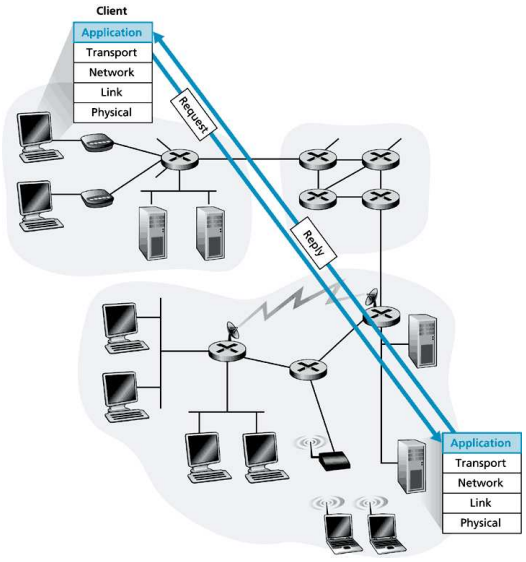
Application-layer protocols (cont)

The figure in the next page shows that a network application is made of a **client** and a **server**.

Usually a host implements both the client and the server, e.g., when an FTP session exists between two hosts, either host can transfer files to the other host.

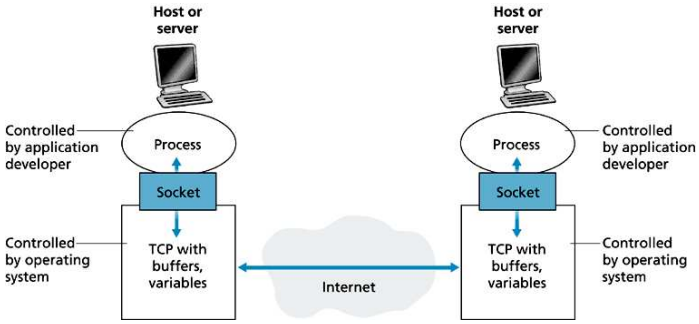
However, the host that initiates the session is labeled the client.

Application-layer protocols (cont)



Application-layer protocols/Sockets

A process sends and receives messages through its **socket**. By analogy, a socket can be considered as an application's door to the network, which is assumed (by the application) to provide a transport service for the messages entering it. Formally, it is an **API** (**Application Programmers' Interface**) between the application and the network.



Application-layer protocols/Addressing processes

In order for one process on one host to send a message to another process running on another host, it must identify the receiving process.

For this, two elements are needed on the Internet:

- the **IP address** of the receiving host, which is a 32 bits number globally unique;
- a **port number** within the receiving host, which allows addressing the message to the peer process.

Therefore, a network application must be assigned a port number, e.g., 80 for Web server processes (using HTTP protocol) or 25 for mail server processes (using SMTP protocol).

Application-layer protocols/User agents

A network application usually offers a graphical interface to the user. It is usually called **user agent** because it relays the wishes of the user to the core of the application network, which interacts in turn with the transport layer through sockets.

Web browsers and mail readers provide such user agents, while implementing specific application-layer protocols, respectively HTTP (web), SMTP (sending e-mails), POP3 or IMAP (retrieving e-mails).

Application-layer protocols/Transport services

The Internet offers several transport protocols to the network applications:

- **Reliable data transfer.** Some applications, such as e-mail, file transfer, remote host access, web document retrieving etc. require no data loss. Others, such as real-time or stored audio and video playing are **loss-tolerant applications**.
- **Bandwidth.** Some applications, such as Internet telephony, require a minimum bandwidth allocation: they are **bandwidth-sensitive applications**. Others, such as e-mail, can make use of any available bandwidth. They are **elastic applications**.
- **Timing.** Some applications, such as multi-player games, require timing constraints on the end-to-end delay: they are **time-sensitive applications**.

Application-layer protocols/Transport services (cont)

Application	Data Loss	Bandwidth	Time-Sensitive
File transfer	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Real-time audio/video	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Application-layer protocols/TCP services

Let us revisit the TCP services:

- **Connection-oriented service.** TCP has the client and the server exchange transport-layer control information before they exchange application-level messages: this is **handshaking**.

  After it, a **TCP connection** is said to exist between the sockets of the two processes.

  The connection is **full-duplex**, i.e., the two processes can send information to the other at the same time. After, the connection must be torn down.

- **Reliable transport service.** The processes rely on TCP to deliver their messages correctly, entirely and orderly.

Application-layer protocols/TCP services (cont)

TCP does *not* guarantee

- minimum transfer rate (due to flow and congestion control),
- end-to-end delay (due to routing protocols and queuing).

Applications	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 2821]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Remote file server	NFS [McKusik 1996]]	UDP or TCP
Streaming multimedia	Often proprietary (for example, Real Networks)	UDP or TCP
Internet telephony	Often proprietary (for example, Dialpad)	Typically UDP

Plan

- Application layer
  - Principles of application layer protocols
  - **The Web and http**
  - File transfer: FTP
  - Electronic mail in the Internet
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Application layer/The Web and http

The **Hyper-Text Transfer Protocol** (**http**) is the Web application-layer protocol.

HTTP is implemented in two programs: the client and the server, running on different and connected hosts, and exchanging HTTP messages.

A **Web page** (or **document**) consists of **objects**. An object is simply a file (e.g. an HTML file, a JPEG image, a Java applet, an audio file etc.) which is addressable by a single **Universal Resource Locator** (**URL**).

For instance, if a web page contains a **base html file** and five JPEG images, then the Web page contains six objects.

Application layer/The Web and http (cont)

An URL is made of

- a protocol name,
- a Web server name,
- a file path on the server.

For instance `http://www.ietf.org/rfc/rfc2396.txt` specifies

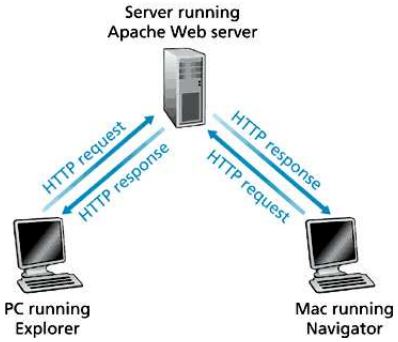
- the use of the HTTP,
- a unique computer named `www.ietf.org`,
- and the location of a text file or page to be accessed on that computer whose pathname is `/rfc/rfc2396.txt`.

Application layer/The Web and http (cont)

A Web server houses web objects, each addressable by a URL. Popular Web servers include Apache.] When a user clicks on an **hyperlink**, its browser sends HTTP request messages for the objects in the referred page to the server, which in turn responds with HTTP response messages that contain the objects.

Because it relies on TCP, HTTP does neither worry about data loss nor ordering.

Application layer/The Web and http (cont)



Application layer/The Web and http (cont)

It is important to understand that the server sends requested objects to the client without storing any state information about the client.

For instance, if the same client asks for the same file two times in a short lapse, the server never responds that it has just send it *because it does not remember*.

That is why HTTP is said to be a **stateless protocol**.

There are two kinds of TCP connections: **persistent** and **non-persistent**.

The Web and http/Non-persistent connections

Suppose a client uses a non-persistent connection to query a page made of a base HTML file and ten JPEG images, all objects being stored on the same server. The URL for the base HTML file is `http://www.school.org/dep/index.html`

- The HTTP client initiates a TCP connection to the server `www.school.org` on port number 80, which is the default port number for HTTP.
- The HTTP client send an HTTP request message to the server via the socket associated with the TCP connection that was established in step 128. This message includes the path `/dep/index.html`.

The Web and http/Non-persistent connections

- The HTTP server receives the request via the socket associated with the connection, retrieves the object `/dep/index.html` from its storage (RAM or disk), encapsulates the object in an HTTP response message and sends it to the client via the same socket.
- The HTTP server tells TCP to close the connection (but the TCP server waits for the client acknowledgement).
- The HTTP client receives the response message. The TCP connection terminates. The message indicates that the embedded object is an HTML file. The client extracts this file, parses the file and find references to the ten JPEG objects.
- The first four steps are repeated for the ten JPEG images.

The Web and http/Non-persistent connections (cont)

The steps described use **non-persistent connections** because each TCP connection is closed after the server sends the object — it does not persist for other objects.

Thus, in our example, 11 TCP connections are generated.

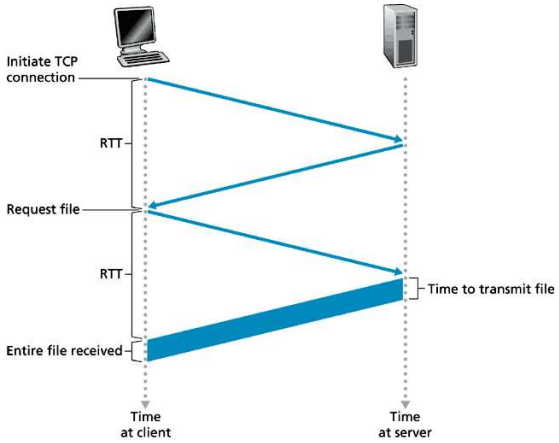
However, it is possible that the browser actually opens several TCP connections in parallel, but these connections still are non-persistent.

The Web and http/Non-persistent connections (cont)

Let us estimate the amount of time that elapses between the moment the client requests a base HTML file and the time it is received entirely.

The **round-trip time** (**RTT**) is the time needed for a small packet to travel from the client to the server and back to the client. It is roughly two RTTs plus the transmission time.

The Web and http/Non-persistent connections (cont)



The Web and http/Persistent connections

Non-persistent connections allow only one object to be transmitted over a given TCP connection, suffering two RTT delays.

With **persistent connections** the server let open the connection after sending a response. For instance, the eleven previous objects could have been sent during a single TCP persistent connection.

A persistent connection is usually closed after a configurable time interval. There are two kinds of persistent connections:

- **without pipelining**: the response must be received before another request is made;
- **with pipelining**: multiple requests can be gathered into the same TCP segment, so the server can fulfill them in parallel (leading to less Internet traffic).

The Web and http/http requests

As expected, there are two kinds of HTTP messages: requests and responses.

For example:

```
GET /dep/index.html HTTP/1.1
Host: www.school.org
```

```
Connection: close
User-agent: Mozilla/4.0
Accept-language:fr
```

A request is written in ordinary ASCII text (encoding characters with 7 bits).

The Web and http/http requests (cont)

This message is made of five lines, but can have less or more lines in general.

The first line `GET /dep/index.html HTTP/1.1` is called the **request line** and the following are the **header lines**.

The request line has three fields:

- the method field (`GET`),
- the URL field (`/dep/index.html`),
- the version field (`HTTP/1.1`).

The method field can take several values, `GET`, `POST`, `HEAD` etc. The `GET` method is used to request an object.

The Web and http/http requests (cont)

The header line `Host: www.school.org` specifies the host on which the object resides.

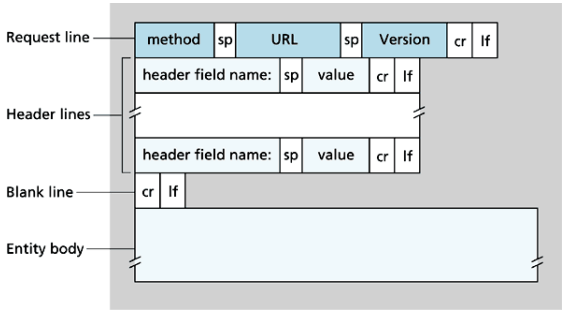
By including the line `Connection: close` the browser tells the server it does not want a persistent connection (thus it should be closed after the object has been sent).

Line `User-agent: Mozilla/4.0` specifies the browser type and version, a Mozilla user agent.

Finally, `Accept-language:fr` indicates that the user prefers to receive a French version of the object, if available, otherwise a default version.

The Web and http/http requests (cont))

Let us take a look to the general shape of an HTTP request:



There is another field in the message called **body**. It is empty when sending a `GET` request method but is filled when using `POST`. This method is used to request a page selected using some specific information (in the body field), as with a form.

The Web and http/http requests (cont)

In fact, not all forms use a `POST` method: sometimes the user's information is put in the URL itself and a `GET` method is used instead.

For instance, if a form consists in two fields whose given values are `monkeys` and `bananas`, the following URL is possible:

```
www.some-site.com/search?monkeys&bananas
```

The `HEAD` method is similar to a `GET` one, except the server does not return the requested object — even if it sends back a message. This method is often used by application developers for debugging purposes.

Protocol HTTP version 1.1 allows a `PUT` method to upload objects to a web server and a `DELETE` method to delete an object in a web server.

The Web and http/http responses

This could be a response message to the request we presented:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

... data ... data ... data ...
```

The Web and http/http responses (cont)

The first line `HTTP/1.1 200 OK` is the **status line**. It tells here that the object was found and is returned. The number `200` is the status number (equivalent to the status name).

The following six lines are the **header lines**.

The first one, `Connection: close` acknowledges the non-persistent underlying TCP connection.

The `Date` line gives the time when the server retrieved the object (not when the object was created).

The `Server` line states what kind of web server is running on what kind of operating system (here, an Apache server on Unix). It is analogous to the `User-Agent` line in the request.

The Web and http/http responses (cont)

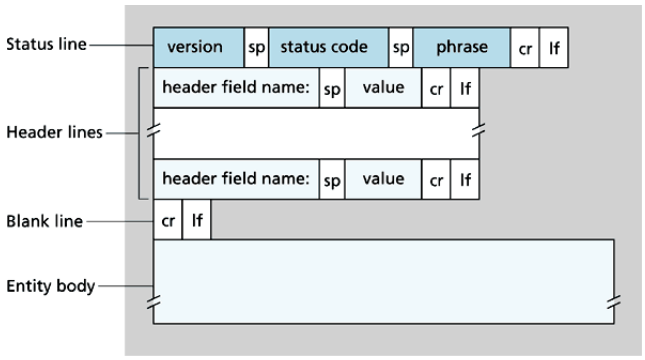
The `Last-Modified` line indicates the date when the object was created or last modified. The `Last-Modified` line is very important for object caching, both on the client side (browser cache) and on the network side (proxy servers).

The `Content-Length` line gives the number of bytes the object is made.

The `Content-Type` line records the kind of the object, in this case a HTML file (the file extension, like `.html` or `.htm` does not tell officially the content type).

The Web and http/http message format/Responses (cont)

The general structure of an HTTP response message is



The Web and http/http responses (cont)

The status code and its associated phrase can be of several kinds:

- `200 OK` means that the request succeeded;
- `301 Moved Permanently` means that the requested object has been permanently moved; the new URL is given in a `Location` header line in the response message and the client browser will automatically retrieve the new URL;
- `400 Bad Request` means that the request was not understood by the server;
- `404 Not Found` means that the object was not found on the server;
- `505 HTTP Version Not Supported` is self explanatory.

The Web and http/http responses (cont)

It is recommended that you create some small HTTP request messages, send them to some web server and examine the response messages.

If you get access to Unix machine (or Windows running Cygwin[1]), just type in a terminal

```
$ telnet konkuk.ac.kr 80
Trying 194.254.134.22...
Connected to konkuk.ac.kr
Escape character is '^]'.
HEAD /~rinderkn/index.html HTTP/1.0
```

(press the return key twice) and you will see the response from the server.

The Web and http/http responses (cont)

```
HTTP/1.1 200 OK
Date: Mon, 02 May 2005 05:25:56 GMT
Server: Apache/1.3.27 (Unix) mod_jk PHP/4.3.10
X-Powered-By: PHP/4.3.10
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Try to request a non-existent object, like `foo.html` and check the response again.

[1] `http://www.cygwin.com`

The Web and http/Authorisation and cookies

We mentioned before that an HTTP server is stateless. This keeps the design of the server simple.

But sometimes it is desirable to identify users, for instance in order to restrict the access or to save personal informations or to provide specific services to subscribers only.

Authorisation

Many web sites require users to provide a name and a password in order to grant the access to the services and objects. Let us follow the steps of such a process.

The Web and http/Authorisation and cookies (cont)

First the user sends an ordinary request. The server responds with a message with an empty body an with a `401 Authorization Required` status code. Also it includes a `WWW-Authenticate` header line telling the client how to authenticate (here: username and password).

The browser receives that response message and prompts the user for a name and a password. The client then resends the request with the name and the password.

After getting the first object, the client has to resend the username and password, but the user does not need to re-enter these data: the browser keeps them in a cache memory.

The Web and http/Authorisation and cookies (cont)

Cookies

Many commercial sites make use of **cookies**. These contain the following components:

- a cookie header line in the HTTP response;
- a cookie header line in the HTTP request;
- a cookie file kept on the user's system and managed by the browser;
- a database at the server site.

The Web and http/Authorisation and cookies (cont)

Let us describe an example of session using cookies.

- A user connects for the first time to an e-commerce site using cookies.
- The web server creates a unique identification number for him and create an entry in its customer database indexed by this number (or **cookie**).
- The server responds to the client browser including a `Set-cookie` header line containing the cookie.
- The browser reads the `Set-cookie` line and appends at the end of a special file a line containing the web site name together with the cookie.

The Web and http/Authorisation and cookies (cont)

- Now, each time the user requests a page on the server, the browser sends the cookie along as a `Cookie` header line. This way the server knows exactly the visited pages and the order and time of visit.

  This allows the site to provide a "shopping card" service: during a single visit, the server keeps track of all the intended purchases, so that the user pays for all of them once, at the end of the session.
- The customer pays by giving his name, address and bank card number. The site now associates the cookie (and all the details of the visit) to the personal data about the customer.

The Web and http/Authorisation and cookies (cont)

- After shopping, if the customer returns to the site, his browser will put again the cookie in the requests. Thus the e-commerce site can propose new products based on the previous visit and he can even by with one click since the site recorded his personal banking information during last session.

Now we understand how can cookies enable a stateful session between the client and the server on top of a stateless HTTP session.

The Web and http/Authorisation and cookies (cont)

Privacy

As you also understand, cookies are a controversial feature because they can provide a lot of private information to commercial web sites.

These sites can even record the behaviour of a specific user *across several web sites*.

Many pages of commercial site request advertisement banners (GIFs or JPEG images) from an advertising company. These requests may contain a cookie, and if the user visits several sites which share the same advertisement company, these company can track the user behaviour across all the sites he visited.

You can visit `http://www.cookiecentral.com/` for news about the cookie controversy.

The Web and http/Conditional `GET`

By caching the previously retrieved objects, the traffic on the internet is decreased: in case the client requests several times the same object, the copy in the cache memory is delivered instead of relaying the request again to the server.

This caching can take place either at the client side (i.e. in the browser) or at a special node in the network.

But caching can create a new problem: the cached objects can be *stale*, i.e. the original object (at the server side) may have changed since the last request.

The mechanism in HTTP which ensures that all retrieved objects are up-to-date despite caching is called **conditional `GET`**.

The Web and http/Conditional `GET` (cont)

An HTTP request message is a conditional `GET` if and only if

- the request message uses the `GET` method,
- the request message includes an `If-Modified-since` header line.

Let us consider an example.

First, a browser requests an *uncached object* from some web server:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
```

The Web and http/Conditional `GET` (cont)

Second, the web server sends a response message with the object:

```
HTTP/1.0 200 OK
Date: Wed, 12 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif

... data ... data ... data ...
```

The client displays the object and stores it in its local cache *along with the last modification date.*

The Web and http/Conditional `GET` (cont)

One week later, the client requests the same object, but this may have changed in the server. In order to get an up-to-date version the user-agent issues a conditional `GET`:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon, 22 Jun 1998 09:23:24
```

Note that the value of the `If-modified-since` header line is exactly the same as the one in the previous (and first, here) request. This conditional `GET` instructs the server to send back the requested object if and only if the object has been modified since the specified date.

The Web and http/Conditional `GET` (cont)

Suppose the object has not been modified in the meanwhile.

Then, fourth, the server sends back he following response message:

```
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

The body is empty: the object has not been sent (again). Note also the status line `Not Modified`.

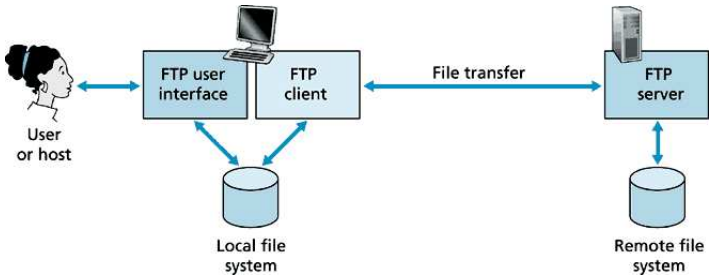
As a final note on HTTP, let us mention that this protocol can be used without any browser or human user and also can carry objects that are not part of web pages, such as XML files.

Plan

- Application layer
  - Principles of application layer protocols
  - The Web and HTTP
  - **File transfer: ftp**
  - Electronic mail in the Internet
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

File Transfer Protocol (ftp)

The **File Transfer Protocol** (**ftp**) follows a client/server model, as HTTP.



The user wants to access a remote account in order to retrieve or send some files. Therefore he must possess an identification and a password on the remote machine. FTP runs on top of TCP.

File Transfer Protocol (ftp) (cont)

First, he enters the name of the remote host to the FTP client through the FTP user agent (under Unix, it is simply a shell command-line but in Windows it is a graphical interface).

Once the server has authorised the user, he can copy files in his local filesystem to the remote host or vice versa.

Both HTTP and FTP are file transfer protocols but there is an important technical difference: FTP uses two different TCP connections to transfer a file: a **control connection** and a **data connection**.

File Transfer Protocol (ftp) (cont)

The control connection is used for exchanging control information between the two hosts, like identification, password, commands to change remote directory, commands to `get` and `put` files etc.

The data connection is used to actually exchange the files.

Because FTP uses a separate control connection, it is said to send its control information **out-of-band**. Dually, HTTP sends its control information **in-band**.

File Transfer Protocol (ftp) (cont)

First, the user starts an FTP session with the remote host by initiating a control TCP connection on server port 21.

The FTP client sends user identification and password to the server over this control connection.

After authentication, the client then sends commands to change remote current directory.

When the server receives a command for file transfer to the client over the control connection, it opens a TCP data connection on client port 20.

File Transfer Protocol (ftp) (cont)

After sending exactly one file over the data connection, the server closes it.

So, during an FTP session, only the control connection remains open, one data connection is open for one file at a time, then closed, i.e. data connections are non-persistent.

Throughout a session, the FTP server must maintain a **state** about the user:

- the control connection must be associated with a specific user account;
- the server must keep track of a given user's current directory on its file-system.

By contrast, HTTP is a stateless protocol — it does not keep any user state.

File Transfer Protocol (ftp) (cont)

Common commands and replies

The commands from client to server, and replies, are sent over the control connection in ASCII (i.e. non-accentuated characters and digits encoded with seven bits). Hence, FTP commands, as HTTP ones, are readable by people (plain text).

One command lies on one line and it consists of four characters and some optional arguments. The more common are

- `USER username` is used to send the user identification to the server;
- `PASS password` is used to send the user's password to the server;
- `LIST` is used to ask the server the list of all files in the current directory; this list is sent back on a new and non-persistent data connection;

File Transfer Protocol (ftp) (cont)

- `RETR filename` is used to retrieve (i.e. `get`) a file from the current remote directory;
- `STOR filename` is used to store (i.e. `put`) a file into the current directory of the remote host.

There is typically a one-to-one correspondence between the commands the user enters (by means of the user agent) and the commands the FTP client sends, as exactly one client's `RETR` for one user's `get`.

File Transfer Protocol (ftp) (cont)

The replies consist in a three-digit number followed by an optional message — this is similar to the status code and phrase in the status line of the HTTP response messages.

Some typical replies are

- `331 Username OK, password required`
- `125 Data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

Plan

- Application layer
  - Principles of application layer protocols
  - The Web and HTTP
  - File transfer: FTP
  - **Electronic mail in the Internet**
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Electronic mail in the Internet

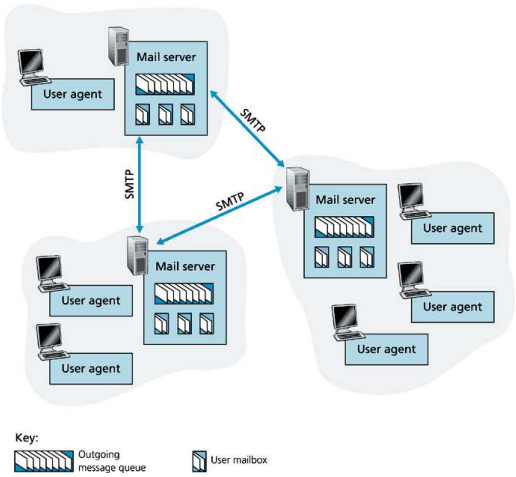
Electronic mail (or **e-mail**) is one of the first application of the Internet, and the most successful.

It is an **asynchronous** communication means, i.e. the two correspondent do not need to be using the e-mail application at the same moment.

The next figure provides a high-level view of the e-mail system. The three major components are

- **user agents** (also called in this context **mail readers**),
- **mail servers**,
- **Simple Mail Transfer Protocol** (**smtp**).

Electronic mail in the Internet (cont)



Electronic mail in the Internet (cont)

We will illustrate how the e-mail system works through the example of a user Alice which sends an e-mail to its recipient Bob.

1. When Alice is finished composing her message, her user agent sends it to its mail server, where it is put in the outgoing **message queue**.
2. In turn, Alice's mail server will send it to Bob's mail server, where it will be stored in Bob's incoming message queue (also called **mailbox**).
3. When Bob will check his e-mails, his user agent will download Alice's message from his mailbox in his mail server and display it.

Electronic mail in the Internet (cont)

Mailboxes

Each recipient, such as Bob, has a mailbox located in one of the mail servers. Bob's mailbox manages the messages that have been sent to him.

When Bob wants to read his new messages, he has to connect to his mailbox and the hosting mail server will authenticate him, by requesting a user name and a password.

Delivery failures

In case Alice's mail server fails to deliver her e-mail to Bob's mail server, the message is kept in Alice's mail server outgoing queue and the server will try to re-send it every 30 minutes or so. If there is no success after several days (often 5 days), the server removes the messages and informs Alice of the situation.

smtp

The Simple Mail Transfer Protocol (SMTP) is the main application-layer protocol for Internet e-mail. It relies on TCP to transfer the e-mails.

SMTP is divided into a **client** and a **server** entity: the client executes on the sender's mail server and the server runs on the recipient's mail server. Both client and server run on every mail server. A mail server becomes an SMTP client when he sends some e-mail, and when it receives some e-mail it behaves as an SMTP server.

Contrary to HTTP, SMTP does require the mail content to be encoded in ASCII. This implies that multimedia content must be encoded by the sender's agent and decoded by the recipient's agent.

smtp (cont)

Let us illustrate a mail exchange between Alice and Bob in a more detailed way.

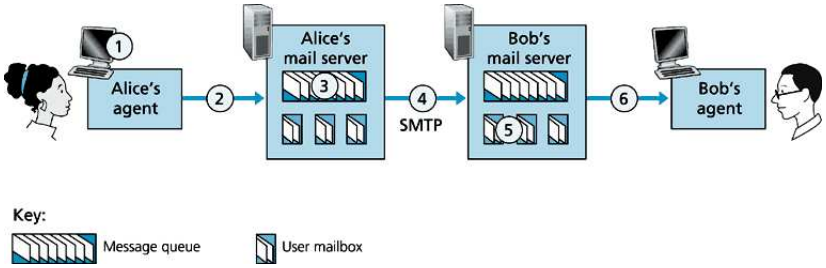
- Alice invokes her mail agent, provides Bob's **e-mail address**, composes a message and order the agent to send the message.
- Alice's agent sends the message to her mail server, where it is placed in a queue.
- The client side of SMTP, running on Alice's mail server, detects the message in the queue. It opens a TCP connection to an SMTP server running on Bob's mail server, on port 25.
- After some SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.
- At Bob's mail server, the server side of SMTP receives Alice's message and places it in Bob's mailbox.

smtp (cont)

- When Bob decides to read his messages he runs his mail agent, which connects to his mail server.

  Bob's mail server authenticates him and delivers to his mail agent Alice's message.

This situation is summarized in the following picture:



Note that there is a direct TCP connection between the two mail servers.

smtp (cont)

Let us detail the SMTP session between Alice's client and Bob's server.

In the following transcript, lines prefixed by **C:** are sent verbatim by the client agent to its TCP socket, and lines starting with **S:** are exactly the lines send to the TCP connection by the server. Client hostname is `crepes.fr` and server hostname `hamburger.edu`.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
```

smtp (cont)

```
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Note that SMTP uses persistent TCP connections, therefore if the sending mail server has several messages to send to the same receiving mail server, it can send all the messages over the same TCP connection.

smtp (cont)

It is recommended to use Telnet to send some e-mail:

```
$ telnet mail.konkuk.ac.kr 25
```

where `mail.konkuk.ac.kr` is the name of the *remote* server. This command establishes a TCP connection between the local host and the remote mail server.

After invoking Telnet at the prompt, the client receives:

```
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
220 konkuk.ac.kr ESMTP Postfix
```

and can type now SMTP commands.

Comparison with http

Both SMTP and HTTP transfer files (objects vs. mails) from one host to another use persistent TCP connections (mandatory for SMTP). The differences are:

1. an HTTP client pulls files (objects) from a server, whereas an SMTP pushes them (mails) to a server: we say that HTTP is a **pull protocol** whereas SMTP is a **push protocol**;

2. SMTP requires the body of each message to be encoded in ASCII, contrary to HTTP, thus accentuated characters or binary data must be encoded and decoded;

3. when a file consists of several parts (like text and images), SMTP places all of them in the same message, contrary to HTTP which puts them in separate messages.

Mail message formats and MIME

The body of an e-mail can be preceded by a blank line and a **header** made of header lines. Each of these lines is made of a keyword followed by a value.

Header lines starting with keywords `From:` and `To:` are mandatory. The header line starting with `Subject:` is optional. For instance:

```
From: Alice
To: bob@hamburger.edu
Subject: Some topics
```

A blank line follows the header and then the contents.

Warning! The header lines are **not** SMTP commands, but are part of the mail body. The header is interpreted by the *receiving mail agent* (not the SMTP server), which then displays it.

The MIME extension for non-ascii data

The header can be used to tell the receiving mail agent how to handle multimedia (i.e. non-ASCII) and multi-part (i.e. attached files) e-mails. These extra header lines and their interpretation constitute the **Multipurpose Internet Mail Extensions** (**MIME**).

The two header keywords enabling MIME are

- `Content-Transfer-Encoding:`
- `Content-Type:`

The first field instructs the receiving mail agent what kind of encoding to ASCII has been used, so it can reverse the process. Then, the second field enables the receiving mail agent to take an appropriate action on the message, as launching an image viewer if the content is JPEG.

The MIME extension for non-ascii data (cont)

Let us take an example. Imagine Alice wants to send a JPEG image to Bob. Her mail agent generates a MIME message, which might look like this:

```
From: Alice
To: bob@hamburger.edu
Subject: Picture
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

... base64 encoded data ...
```

Alice's mail agent encoded the JPEG file using base64, which is one possible technique to produce ASCII from binary.

The MIME extension for non-ascii data (cont)

When Bob's mail agent reads this message, it reads the

```
Content-Transfer-Encoding: base64
```

header, so it proceeds to decode the message body with a base64-decoder.

The header line

```
Content-Type: image/jpeg
```

tells Bob's mail agent that the message should be interpreted (*after decoding*) as a JPEG image. Thus, it may propose to Bob to launch a JPEG viewer if he would like to.

The following header line tells the version of MIME used by Alice's mail agent:

```
MIME-Version: 1.0
```

The MIME extension for non-ascii data (cont)

There are two formats of MIME types:

```
Content-Type: type/subtype
Content-Type: type/subtype; parameters
```

In the previous example, the type was `image` and the subtype was `jpeg`. The subtype restricts the meaning of the type, thus allowing a more precise interpretation of the content.

Most parameters are associated with a specific subtype and are similar to a variable definition.

The set of types and subtypes, as well as parameter, is open and increasing. New types should be registered at the Internet Assigned Numbers Authority (IANA)[2].

The MIME extension for non-ascii data (cont)

Currently there are seven types defined. The following are common.

- **text** tells the receiving mail agent that the body contains text. If the subtype is `plain`, the mail agent does not need any extra software to display the body, except font support. The character set can be specified using a parameter, like `charset=euc-kr`.

  Another popular pair is `text/html`, which informs the receiving agent that it can display the content as a web page.

- **image** tells the receiving mail agent that the body is an image. Two pairs of type and subtype are popular: `image/jpeg` and `image/gif`.

The MIME extension for non-ascii data (cont)

- **application** is used when the content does not fit any other type. It is often used for data that must be processed by an application before it can be viewed. For instance, `application/msword` instructs the receiving mail agent to launch Microsoft Word or OpenOffice to read the mail.

- **multipart** is used when the mail contains several parts because the sender wanted to send some other files together with his main message. If the attached files are not text, the pair `multipart/mixed` is often used.

The MIME extension for non-ascii data (cont)

Let us say a little more about multi-part e-mails.

[2] `http://www.iana.org/`

The mail agent needs to determine where each part starts and ends inside the body, how each non-ASCII part was encoded and what kind of content is it.

This is simply achieved by

- placing *boundary characters* between each part,
- repeating `Content-Transfer-Type:` and `Content-Type:` header line at the beginning of each part.

The MIME extension for non-ascii data (cont)

Assume Alice wants to send a message to Bob consisting of some ASCII text and a JPEG image. After she types the text and attach the image, the agent produces

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart

--StartOfNextPart
Dear Bob, look at my picture:
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
... base64 encoded data ...
```

Received Message header

The *receiving* SMTP server can also insert a specific line at the top of an e-mail: the `Received:` header line. This line specifies the originating SMTP server, the receiving SMTP server and the reception time. Like:

```
Received: from crepes.fr by hamburger.edu;
          12 Oct 98 15:27:39 GMT
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
.....
```

Received Message header (cont)

A user can instruct his mail server to forward his e-mail to another mail server. In this case each receiving mail server, i.e. the forwarding one and the final one, will add a `Received:` line to the original mail.

For instance, if Bob forwards his e-mails to the mail server `kimchi.kr`, the final messages will look like

```
Received: from hamburger.edu by kimchi.kr;
          3 Jul 01 15:30:01 GMT
Received: from crepes.fr by hamburger.edu;
          3 Jul 01 15:17:39 GMT
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: My picture
.....
```

Mail Access Protocols

We implicitly assumed until now that Bob reads his e-mails by directly logging in his mail server.

For a long time, this was the only way but since the mid 90's users have a mail agent that connects to the user's mail server, downloads the e-mails on the local disk and displays their contents to the user.

Using a mail agent allows to store locally the e-mails but also to easily visualise their possible multimedia attachments (images, videos, music etc.) — the purpose of a mail server is to serve mails, not to provide fancy features for display.

Note: the idea to run the recipient's mail server on the same machine as the mail agent is not good. Indeed, the user may want to turn his machine off and so the mail server would be, leading to the delivery failure of subsequent incoming mails.

Mail Access Protocols (cont)

Remember that a mail server is always on, it is usually shared by several users and is maintained by the ISP.

So

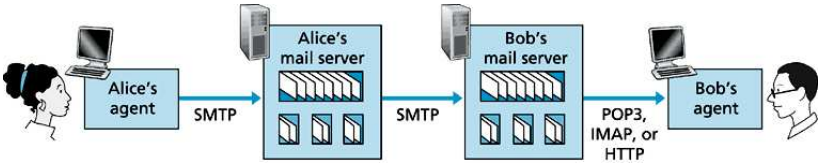
- Alice's mail agent has *to push* her e-mail to her mail server,
- Bob's user agent has *to pull* his e-mails to his local disk.

The first task can be done using SMTP because it is a push protocol by design. Why a two-step procedure (from Alice's mail agent to her mail server, then from her mail server to Bob's mail server)? Because otherwise, if Bob's mail server is down, Alice's mail agent should keep trying again to send the e-mails for days, impeding Alice to log out meanwhile.

Mail Access Protocols (cont)

The second tasks cannot be achieved by SMTP: we need a pull protocol, or **Mail Access Protocol**.

There are several possibilities: **pop3** (**Post Office Protocol** version 3), **imap** (**Internet Mail Access Protocol**) and HTTP:



Mail Access Protocols/POP3

POP3 is an extremely simple mail access protocol.

The session starts with the recipient's mail agent opens a TCP connection to its mail server on port number 110.

Then three phases occur in turn:

1. **Authorisation**: the user agent sends the user's name and password to the mail server, which authenticates him;
2. **Transaction**: the user agent retrieves the messages and the user can: mark the mails to be deleted on the server, unmark them and get statistics about them;
3. **Update**: the mails marked for deletion on the server are actually deleted.

Mail Access Protocols/POP3 (cont)

In a POP3 transaction, the user agent issue textual commands and the server responds to each command with a textual reply.

There are two possible replies:

- `+OK` (sometimes followed by server-to-client data) meaning the last client command was fine;
- `-ERR` meaning that something was wrong with the last command.

Mail Access Protocols/POP3 (cont)

Authorisation

There are two principal commands: `user` *`name`* and `pass` *`password`*.

Use `telnet` to try these commands:

```
$ telnet mail.konkuk.ac.kr 110
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
+OK <11115.1117125523@konkuk.ac.kr>
user rinderkn
+OK
```

Mail Access Protocols/POP3 (cont)

Example of failed authentication:

```
$ telnet mail.konkuk.ac.kr 110
Trying 202.30.38.143...
Connected to mail.konkuk.ac.kr.
Escape character is '^]'.
+OK <15247.1117126049@konkuk.ac.kr>
user foo
+OK
pass bar
-ERR authorization failed
Connection closed by foreign host.
```

Mail Access Protocols/POP3 (cont)

Transaction

The users can configure his mail agent to **download and delete** or **download and keep**, depending whether they wish to keep a copy of their mails on the server or not.

According to this configuration, the user agent sends different commands to the mail server, e.g. in the "download and delete" mode, the user agent will issue the `list`, `retr` and `dele` commands.

Imagine a user has two messages in his mailbox. In the following transcript, lines prefixed by **C:** are sent verbatim by the client agent to its TCP socket, and lines starting with **S:** are exactly the lines sent to the TCP connection by the server.

Mail Access Protocols/POP3 (cont)

```
C:  list
S:  1 498
S:  2 912
S:  .
C:  retr 1
S:  ... blah blah ...
S:  .
C:  dele 1
C:  retr 2
S:  ... blah blah ...
S:  .
C:  dele 2
C:  quit
S:  OK POP3 server signing off
```

Mail Access Protocols/POP3 (cont)

A problem with this "download and delete" approach is that the recipient, Bob, may be nomadic and want to access his mail messages from multiple machines, e.g. his office computer, his home computer and his portable computer.

If Bob reads his messages from his office machine, he will not be able to reread them from his home machine.

In the "download and keep" mode, the user agent leaves the messages on the server after downloading (a copy of) them.

During a POP3 session, the server keeps a client state, but not across sessions — which greatly simplifies the server implementation.

Mail Access Protocols/IMAP

With POP3, once the recipient has downloaded his messages, he can store them in a hierarchy of folders and later search information in it (e.g. by sender's name or subject).

But a nomadic user would get a hierarchy with different messages *for each machine* from where he downloads the mails. One centralised hierarchy would be preferable in this case. This is not possible with POP3.

To solve this problem, **imap** (**Internet Mail Access Protocol**) provides more features than POP3, at the cost of more complex implementations.

Mail Access Protocols/IMAP (cont)

An IMAP server will associate each message with a folder (initially, it is the "Inbox" folder).

The user can create and change folders on the server, as well as searching the messages they contain according to specific criteria.

This means that IMAP, contrary to POP3, maintain user state information *across sessions*.

Another useful IMAP feature is the possibility for the client to obtain only some components of a message. For instance, a user agent can request just the body of a message or one part of a multi-part MIME message. This is very useful in case of a low-bandwidth connection.

Mail Access Protocols/Web mail

A lot of users today access their e-mails through their Web browsers. This technique was introduced in the mid-1990's by the company Hotmail.

In this case, Alice's mail agent sends her messages using HTTP requests (like forms) to her web server which, in turn, uses SMTP to push the mail to the recipient's mail server.

Symmetrically, Bob retrieves his messages using HTTP requests to his web server which, in turn uses IMAP to get the messages from the sender's mail server.

This web-based e-mail system is very useful for mobile users since it only requires a web browser and an internet connection.

Plan

- Application layer
  - Principles of application layer protocols
  - The Web and HTTP
  - File transfer: FTP
  - Electronic mail in the Internet
  - **DNS — The Internet's directory service**

Internet Directory Service (dns)

The internet protocols use **IP addresses** to identify hosts. They are a series of numbers, between 0 and 255, separated by periods like `202.30.38.143`.

But these numbers have are difficult to remember, contrary to **host names**, which are made of a series of words separated by periods, like `mail.konkuk.ac.kr`.

The internet provides an application-layer service that allows applications to find the IP address of a host from its name: the **Domain Name Service** (**dns**).

Because of its functional similarity with the telephone directories, which map subscribers' names to their phone number, this service is also called **Internet Directory Service**.

Internet Directory Service (dns) (cont)

The DNS is

- a client/server protocol;
- a distributed database implemented in a hierarchy of **name servers**;
- an application-layer protocol that allows hosts and name servers to communicate in order to provide the translation service.

Name servers are often machines running the UNIX operating system and the Berkeley Internet Name Domain (BIND) software.

The DNS protocol runs over UDP and uses port 53.

DNS is commonly used by other application-layer protocols, like HTTP, SMTP and FTP.

Internet Directory Service (dns) (cont)

When a user wants to retrieve the web page whose URL is `http://konkuk.ac.kr/index.html`,

1. its browser extracts the hostname `konkuk.ac.kr` from the URL,
2. passes it to the client side of the DNS application,
3. this application sends to a DNS server the hostname,
4. after some delay, it receives back the IP address of the referred host,
5. this IP address is returned to the browser,
6. the browser opens a TCP connection to the host and to the HTTP server process located there.

Internet Directory Service (dns) (cont)

DNS provides a few other important services in addition to translating hostnames to IP addresses:

- **Host aliasing.** A host with a complicated name can have one or more (simpler) names. For example `relay1.west-coast.enterprise.com` may have two aliases, e.g., `enterprise.com` and `www.enterprise.com`. In this case, `relay1.west-coast.enterprise.com` is said to be the **canonical hostname**.
- **Mail server aliasing.** Mail server names can also be aliased in order to get a simple e-mail address, as `bob@hotmail.com` instead of `bob@relay1.west-coast.hotmail.com`.

Internet Directory Service (dns) (cont)

- **Load distribution.** Busy sites are replicated over multiple servers, hence an *ordered list* of IP addresses is associated with one canonical hostname. Each time a client requires an IP address in the list, the first is returned and the list is rotated. This way, the load is balanced between all the servers.

Internet Directory Service (dns) (cont)

What if there were only one name server in the internet?

- If the name server crashes, so does the internet.
- A single name server would have to handle all the DNS queries (from HTTP requests and e-mails at hundreds of millions of hosts).
- A single name server cannot be geographically close to all the hosts, imposing unfair delays to the hosts.
- A single name server would need a huge database to store all the hostname on the internet.

Therefore *a centralised database does not scale.*

Internet Directory Service (dns) (cont)

That is why

- there is a large number of name servers around the world
- which are organised in a hierarchy;
- no single name server has all the mappings for all the hosts;
- the mappings are distributed across the name servers;
- there are three kind of name servers:
  1. **local name servers**,
  2. **root name servers**,
  3. **authoritative name servers**.

dns/Local name servers

Each ISP has a **local name server** (also called a **default name server**), so it is relatively close to the users of this ISP:

- in an institution, it can be on the same LAN as the user;
- in case of a residential ISP, it is separated from the users by no more than a few routers.

The IP address of the local name server has to be set by the user in the network configuration of his host.

If the user requests a translation for a host which is part of the same ISP domain, the request will be *immediately* served by the local name server. As host `surf.eurecom.fr` requesting the IP address of `baie.eurecom.fr`.

dns/Root name servers

When a local name server can not serve a request, it then acts as a DNS client and queries a **root name server**.

If the root name server has the required mapping, it returns the IP address to the local name server which, in turn, delivers it to the querying (initial) host.

If the root name server has not the required record, it knows the IP address of an *authoritative name server* which has it.

dns/Authoritative name servers

Every host is registered with an authoritative name server.

This authoritative name server is a name server in the (*registered*) host's local ISP.

In principle, each host should be registered in two authoritative name servers, in case of failures.

By definition, a name server is authoritative for a given host if it always translates the host name into its IP address.

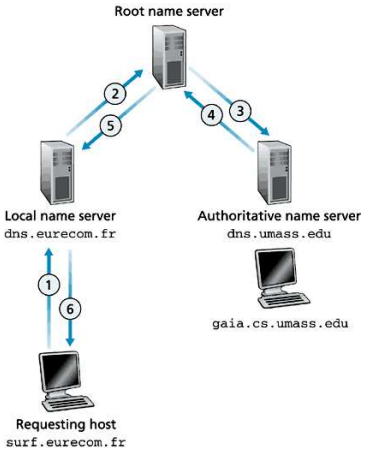
Many name servers act both as local and authoritative name servers.

dns/Example

Let us consider an example. Assume that host `surf.eurecom.fr` queries the IP address of host `gaia.cs.umass.edu`.

1. `surf.eurecom.fr` sends its query to its local name server `dns.eurecom.fr`,
2. which, in turn, forwards the query to a root name server,
3. which, in turn, forwards the query to an authoritative name server for the host `gaia.cs.umass.edu`, named `dns.umass.edu`.

dns/Example (cont)



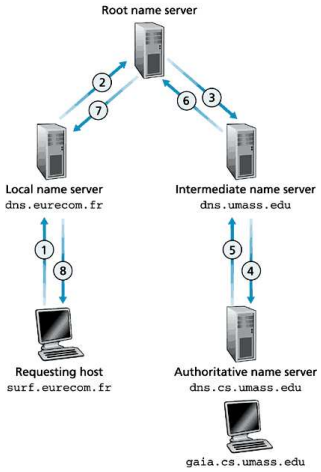
dns/Example (cont)

Finally, the IP address of host `gaia.cs.umass.edu` is forwarded back to host `surf.eurocom.fr`, through all the DNS clients (steps 4, 5 and 6).

Until now we assumed that a root name server always knows an authoritative name server for the requested host. In fact, this is not always true: it usually knows an **intermediary name server**, which may know an authoritative name server or another intermediary name server.

Let us reconsider the previous example and assume that the root name server does not know an authoritative name server for `gaia.cs.umass.edu` but, instead, a name server for the whole domain (the university) `umass.edu`, called `dns.umass.edu`. This intermediary name server knows all the name servers for all the departments, in particular `cs.umass.edu`.

dns/Example (cont)



dns/Recursive and iterative queries

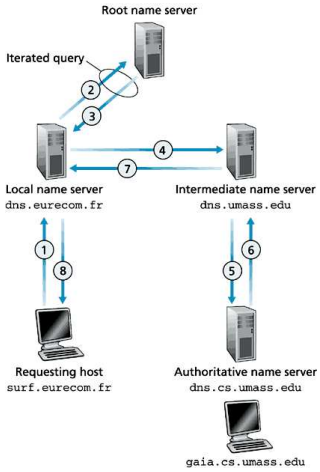
As in our example, when a name server A becomes a DNS client of name server B which, in turn, is client of C and that C sends the result to B, this is called a **recursive query** because B acts in behalf of A.

It is also possible that B responds to A with the IP address of name server C, leaving to A the task to *directly* query C: this is an **iterative query** — A makes all the queries.

It is possible to mix recursive and iterative queries for serving a single user's request.

Iterative queries save bandwidth and processing resources in some name servers (here, B), that is why it is a good idea to use them to ease the burden of root name servers (here, B).

dns/Recursive and iterative queries (cont)



dns/Caching

In order to improve the performances, name servers make use of **caches**.

When a name server receives a translation query and, later, the corresponding IP address, it copies the pair (hostname, address) in its memory or saves it on its local disk — as well as serving the result to its client.

Therefore, the next time the same query is received, the name server can answer just but by searching its cache.

In order to cope with ephemeral hosts, mapping pairs are kept in the caches only for a pre-configured period of time (often set to two days).

dns/Records

A (hostname, address) is embedded in a data structure called a **resource record** (**RR**). In turn, resource records are stored in databases in each name server.

Each DNS message (either query or response) carries one or more resource records.

More precisely, a resource record has the structure (*name*, *value*, *type*, *TTL*) where *TTL* is the **Time To Live** record; it determines the time at which a resource should be removed from the cache.

We will ignore the *TTL* field in the following description.

dns/Records (cont)

The meaning of *name* and *value* depend on the field *type*:

- if *type* is *A*, then *name* is a hostname and *value* is the corresponding IP address. This is the standard hostname-to-address mapping. For example (`konkuk.ac.kr`, 202.30.38.143, *A*).

- if *type* is *NS*, then *name* is a domain (such as `umass.edu`) and *value* is the hostname of an authoritative name server for all the hosts in *name*. This kind of record is used to route DNS queries along in the query chain. For instance, (`umass.edu`, `dns.umass.edu`, *NS*).

dns/Records (cont)

- if *type* is *CNAME*, then *value* is a canonical hostname for the alias hostname *name*. This record provide querying hosts the canonical name of a host. As an example, (`google.com`, `www.google.com`, *CNAME*).

- if *type* is *MX*, then *value* is the canonical name of the mail server *name*, like the record (`google.com`, `mail.google.com`, *MX*). A company can both have an *MX* record and a *CNAME* record, allowing the same alias both for its mail server and its web server: depending on the querying application, the corresponding record will be requested.

dns/Records (cont)

Now we can better understand what is being an authoritative name server or not.

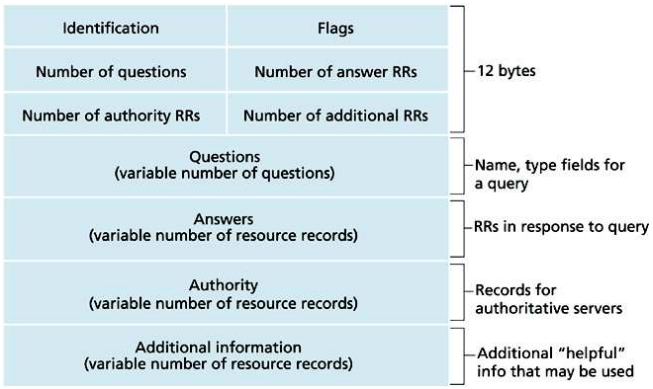
- If a name server is authoritative for a given hostname, then it must contain a *type A* record for this hostname.

- If a name server is not authoritative for a given hostname, it may contain a *type A* record in its cache. Otherwise it must contain a *type NS* record for the domain including the hostname, as well as a *type A* record giving the IP address of the domain name server given in the *type NS* record.

  For example, assume that a root name server is not authoritative for the hostname `gaia.cs.umass.edu` and no *type A* record is in cache for it. Therefore this name server must contain a record like (`umass.edu`, `dns.umass.edu`, *NS*) and (`dns.umass.edu`, 128.119.40.111, *A*).

dns/Messages

There are only two kind of DNS messages and both queries and replies have the same format:



dns/Messages (cont)

The first 12 bytes are the **header section**, which is made of successive fields:

1. the **identifier** is a 16 bits number that identifies a query and its answer;
2. the **flags** are one bit flags:
   - **query** is 0 and **reply** is 1;
   - **authoritative** name server or not;
   - **recursive query** required/not required (by client) or supported/not supported (by server);
3. the **numbers** are four numbers that count the number of occurrences of four kind of data in the next sections.

dns/Messages (cont)

There are now four **data sections**:

1. the **question section** contains information about the current query:
   (a) the **hostname** being queried;
   (b) the type of query, e.g., *type A* or *type MX*;

2. the **answer section** contains information about the current reply: possibly multiple records (because a host can be duplicated);

3. the **authority section** contains records of other authoritative servers;

4. the **additional section** complements other informations, e.g., if a reply is an *MX* record, this section provides the IP address of the canonical mail server.