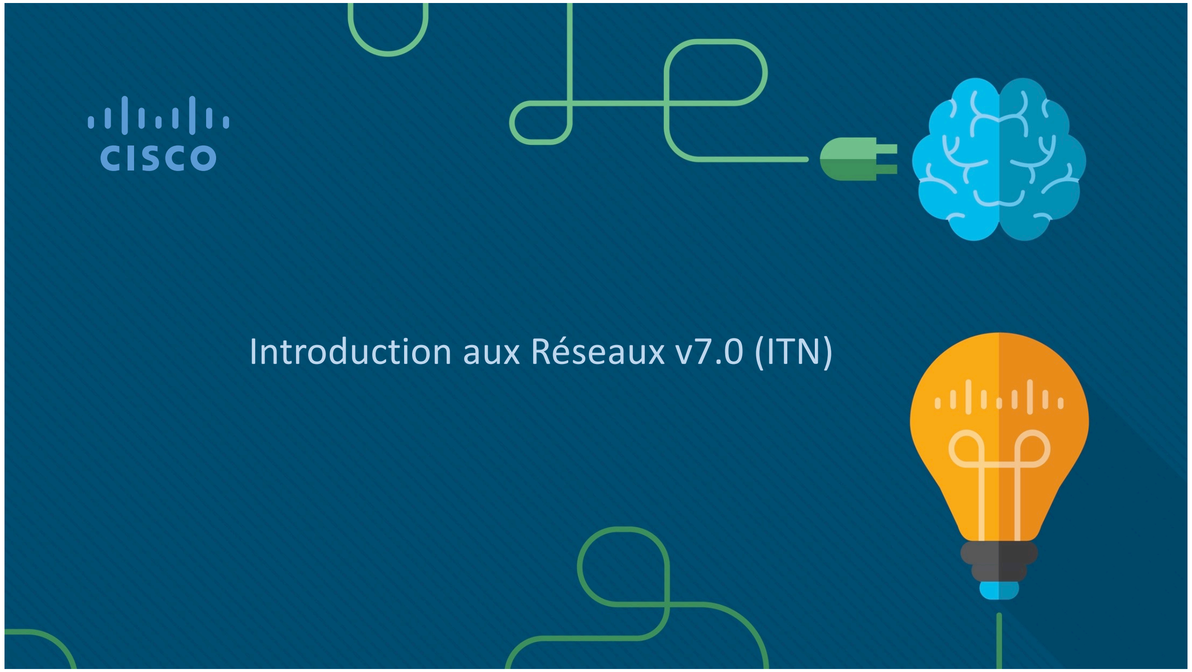
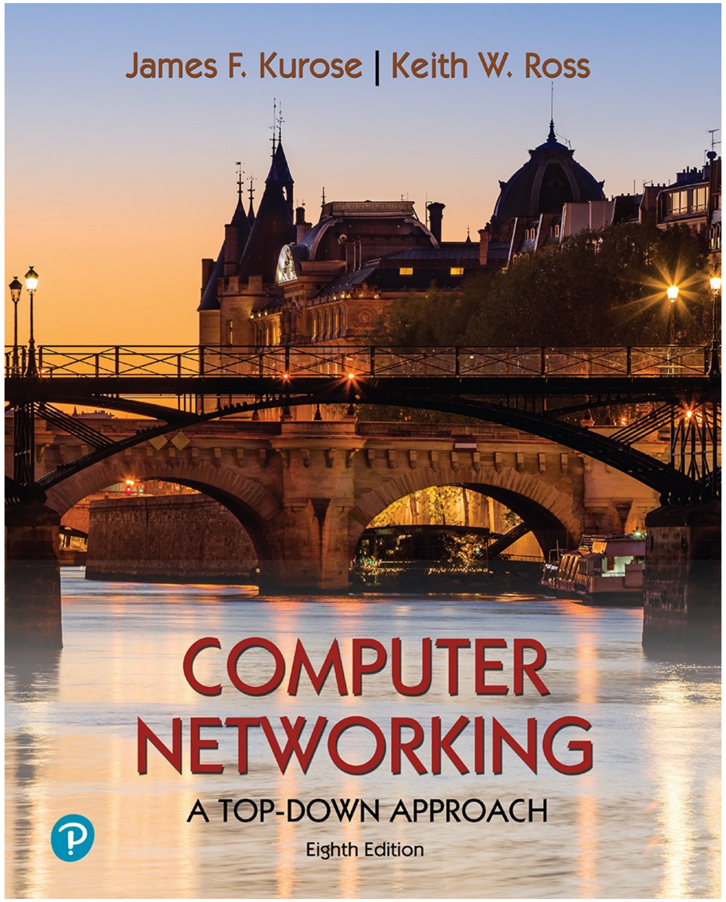
Réseaux fixes
1. Introduction

Luc Deneire

EII-5, Option Réseaux et Objets Connectés (ROC)

Basé (principalement) sur ....





Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Réseaux fixes : Le Syllabus

- **Durée** : 54h
- **Format :** 13,5 séances de 4 h qui incluent … tout ☺
- **Objectif** : structure et conception des réseaux de communications
  - (4 cours) **segmentation** (Routage classique , NAT, VLAN, routage inter-VLAN)
  - (3 cours) **redondance** (STP, redondance de routes/routeurs/liens, équilibrage de charge, …)
  - (3 cours) **sécurité** (SSH, politique de sécurité, firewall, …)
  - (2 cours) **services** principaux (DHCP, DNS, FTP, SSH, HTTP, …)
- Obtenir la certification CCNAv7 (CISCO)
  - « examens » en ligne + 1 examen en classe et 1 TP final en classe
  - Il y a 3 »niveaux », vous pouvez faire le 2 si vous avez déjà le niveau 1

Les cours : Principes et Pratiques

- Chaque séance :
  - 1 à 2 heures de cours « magistral »
  - 1 TP, soit sur matériel réseau, soit par simulation/analyse de trames

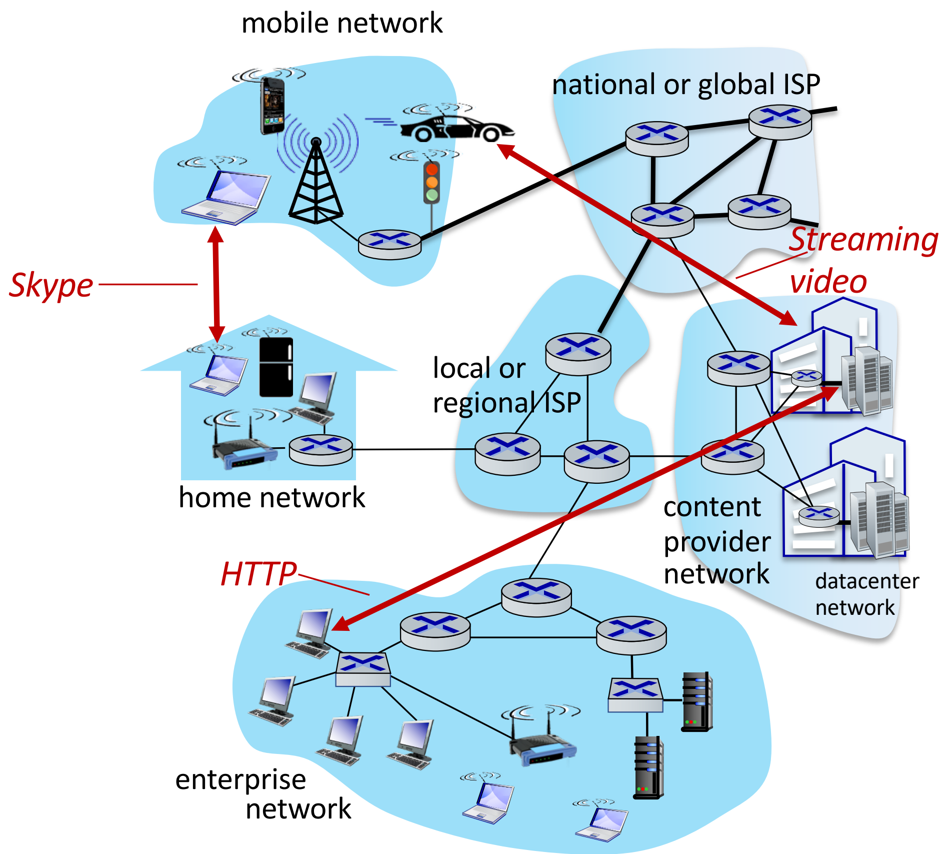
- En parallèle : la certification CCNAv7 (1)(voire 2)
  - 17 (!) chapitres, reprennent la structure de réseau par une approche « bottom up » (des couches basses vers les couches hautes).

Cours d’introduction

- Un petit voyage dans les couches
  - En partant de l’applicatif
  - En traversant la couche TCP/IP
  - En finissant par une application client/serveur et établissement d’un (tout petit) réseau complet.

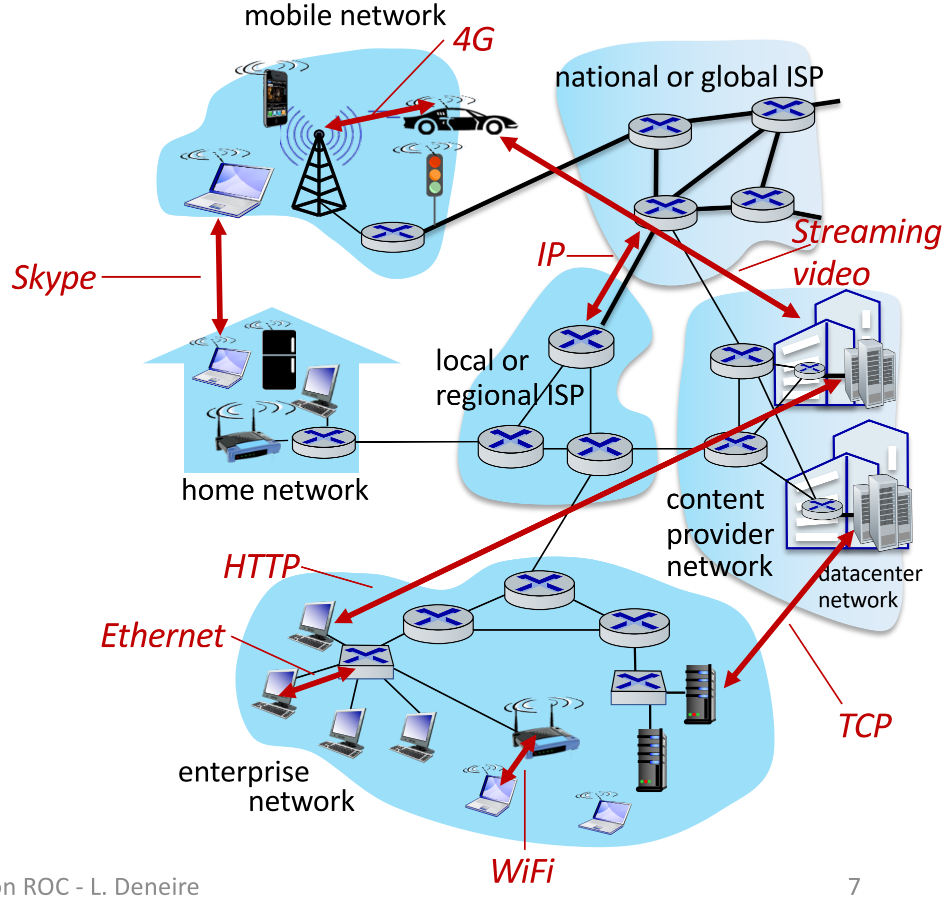
The Internet: a "service" view

- *Infrastructure* that provides services to applications:
  - Web, streaming video, multimedia teleconferencing, email, games, e-commerce, social media, inter-connected appliances, …
- provides *programming interface* to distributed applications:
  - "hooks" allowing sending/receiving apps to "connect" to, use Internet transport service
  - provides service options, analogous to postal service



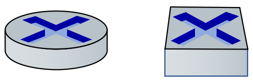
The Internet: a "Protocol" view

- *Internet: "network of networks"*
  - Interconnected ISPs
- *protocols* are *everywhere*
  - control sending, receiving of messages
  - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet
- *Internet standards*
  - RFC: Request for Comments
  - IETF: Internet Engineering Task Force



The Internet: a “nuts and bolts” view

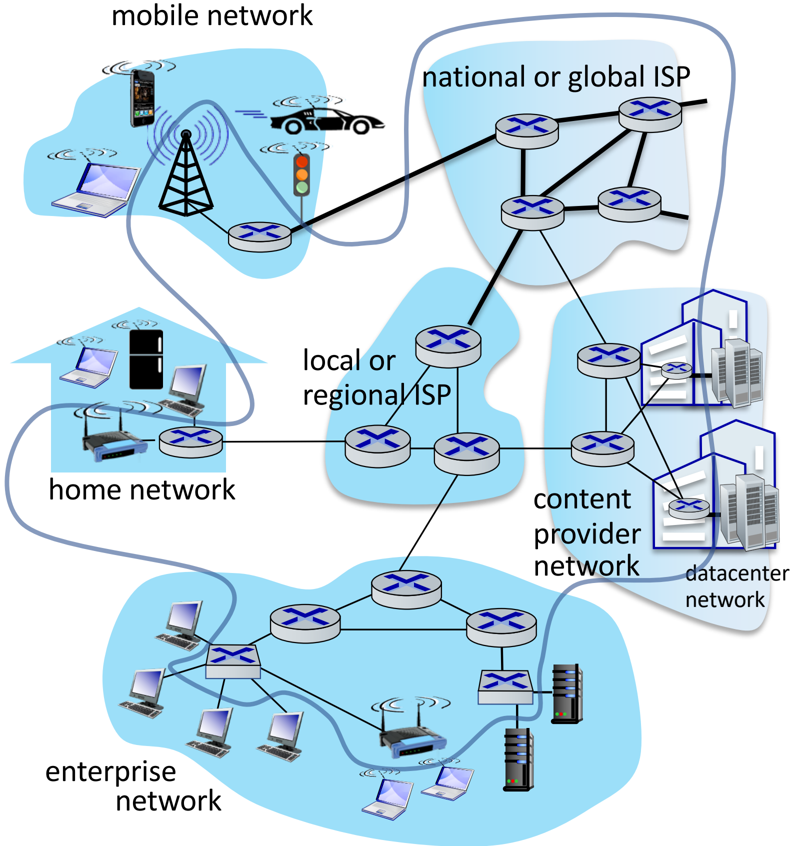
Billions of connected computing *devices*:
- *hosts = end systems*
- running *network apps* at Internet’s “edge”



Packet switches: forward packets (chunks of data)
- *routers, switches*

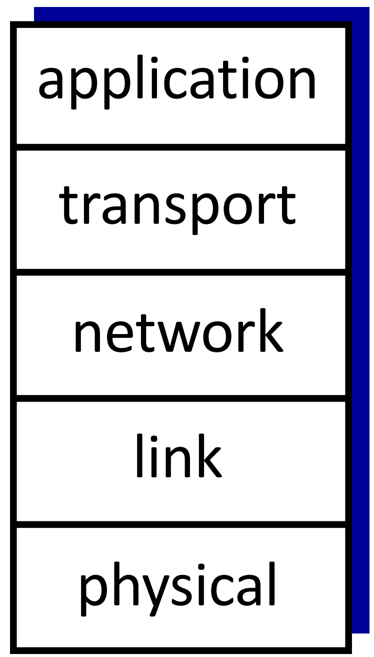
Communication links
- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

Networks
- collection of devices, routers, links: managed by an organization



Internet protocol stack

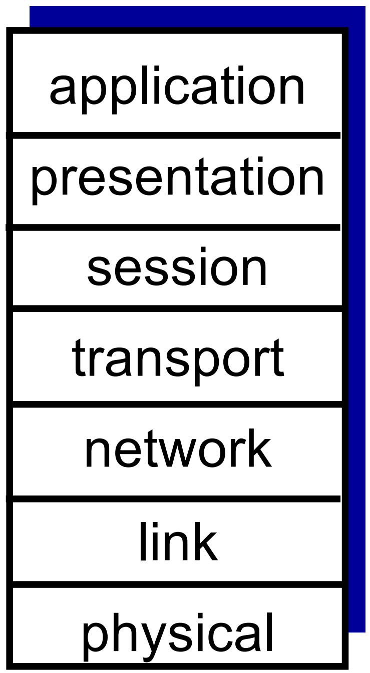
- *application:* supporting network applications
  - IMAP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits “on the wire”



ISO/OSI reference model

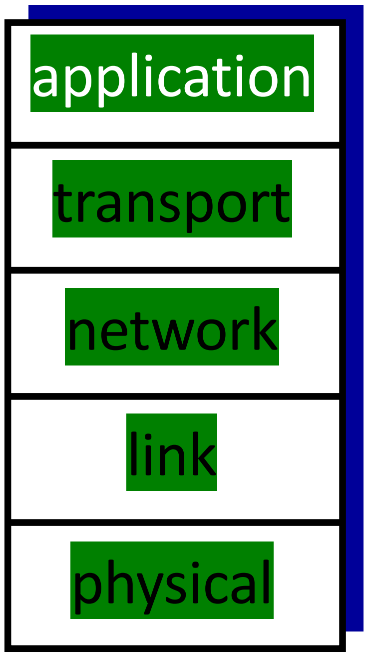
Two layers not found in Internet protocol stack!

- *presentation:* allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack “missing” these layers!
  - these services, *if needed,* must be implemented in application
  - needed?



The seven layer OSI/ISO reference model

Première étape : couche applicative



Application layer: overview

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

Application layer: overview

Our goals:

- conceptual *and* implementation aspects of application-layer protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-layer protocols
  - HTTP
  - SMTP, IMAP
  - DNS
- programming network applications
  - socket API

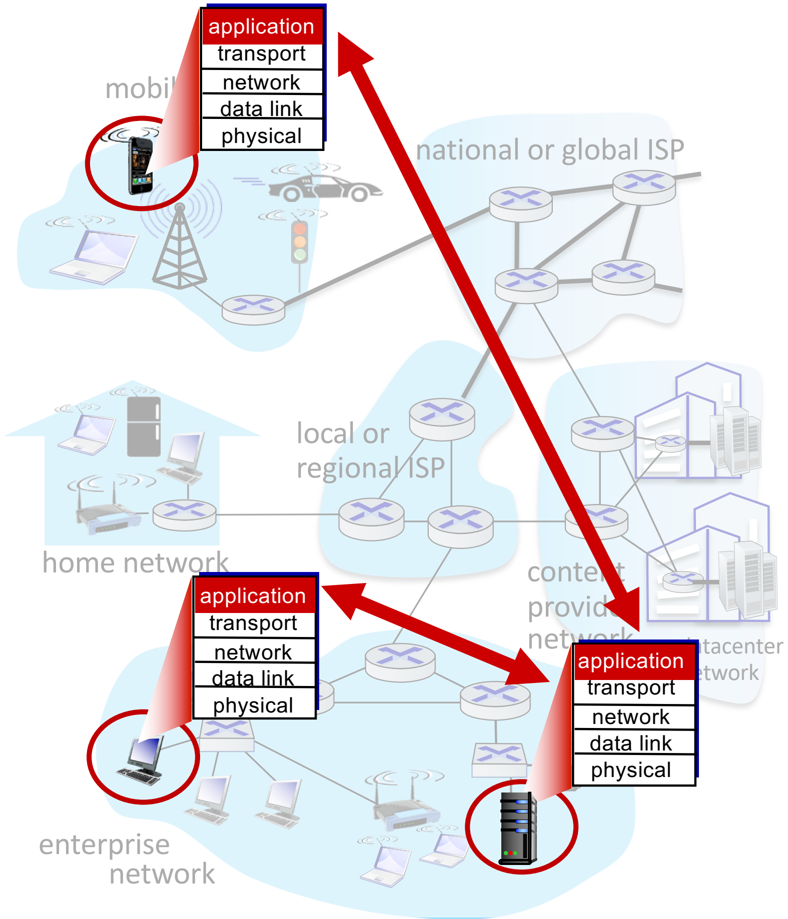
Creating a network app

write programs that:

- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Client-server paradigm

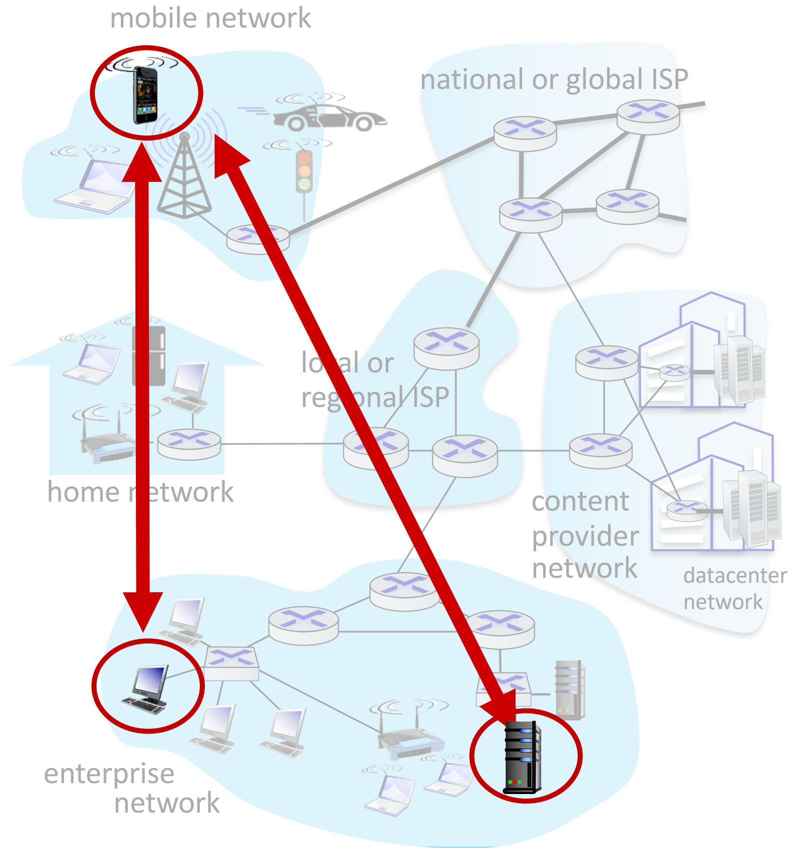
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

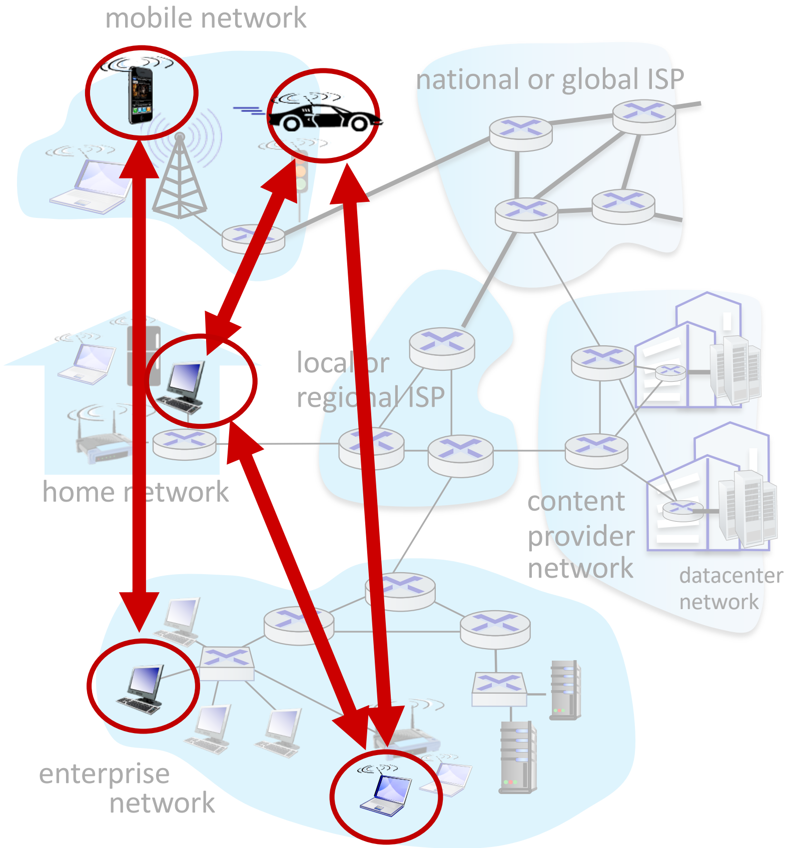
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- example: P2P file sharing



Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

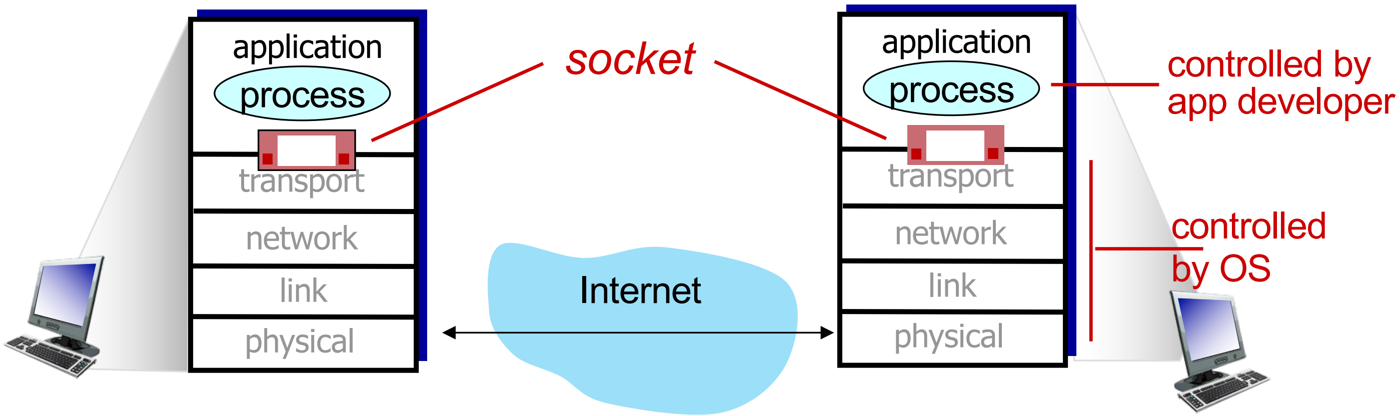
client process: process that initiates communication

server process: process that waits to be contacted

- note: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

An application-layer protocol defines:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Socket programming with UDP

UDP: no “connection” between client & server

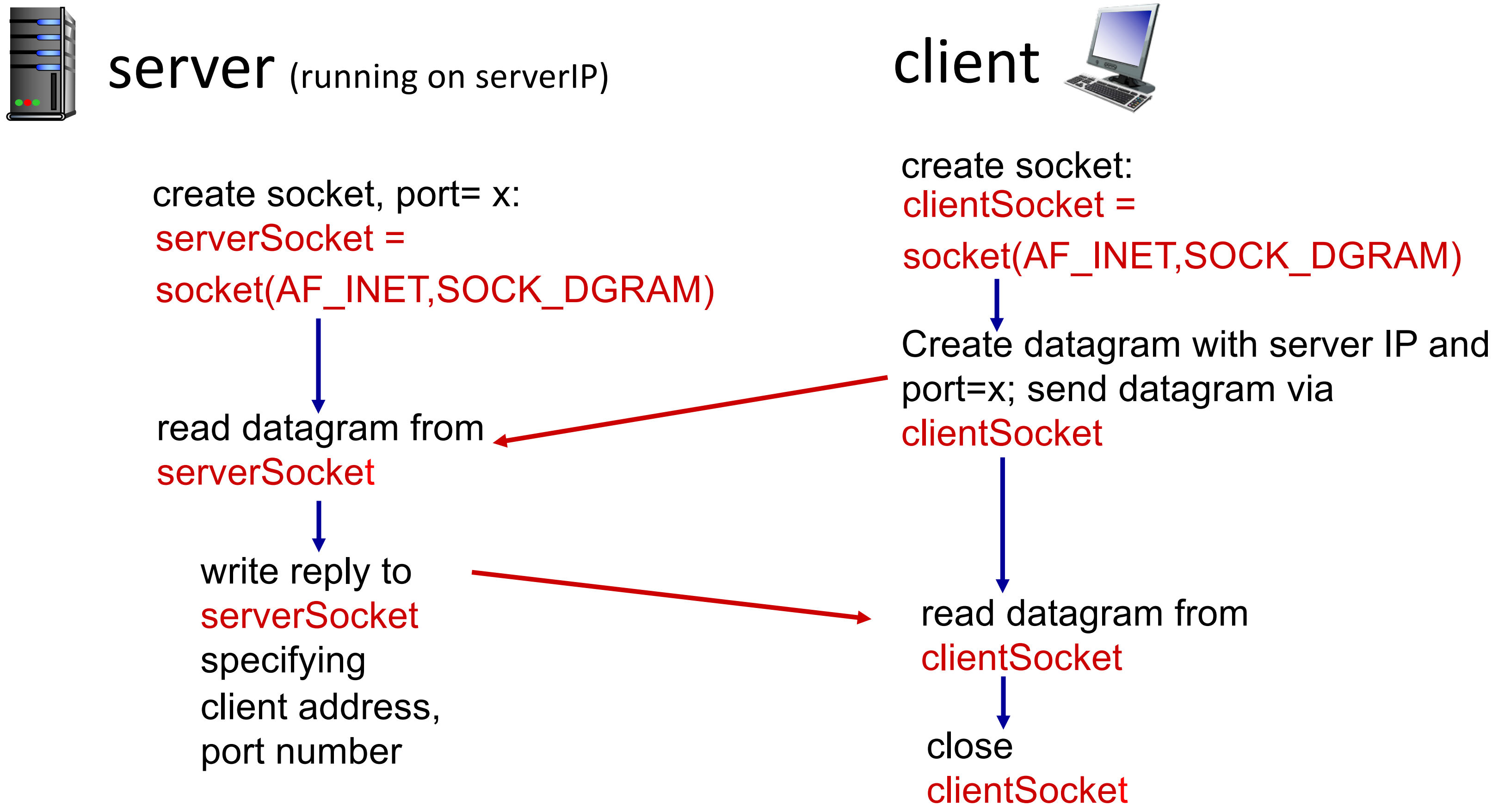
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

Client/server socket interaction: UDP



Example app: UDP client

Python UDPClient

include Python's socket library →
create UDP socket for server →
get user keyboard input →
attach server name, port to message; send into socket →
read reply characters from socket into string →
print out received string and close socket →

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

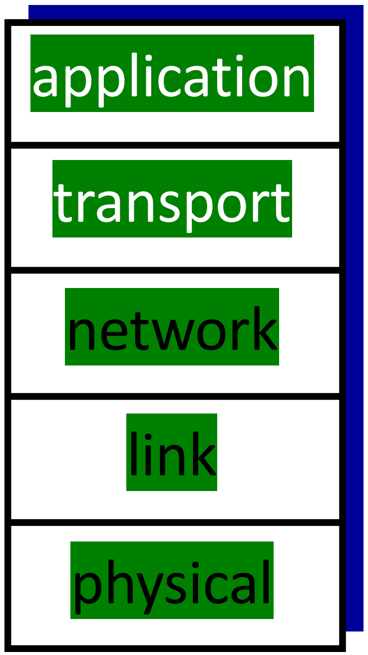
Example app: UDP server

Python UDPServer

create UDP socket →
bind socket to local port number 12000 →
loop forever →
Read from UDP socket into message, getting client's address (client IP and port) →
send upper case string back to this client →

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

Deuxième étape : couche Transport



Transport layer: overview

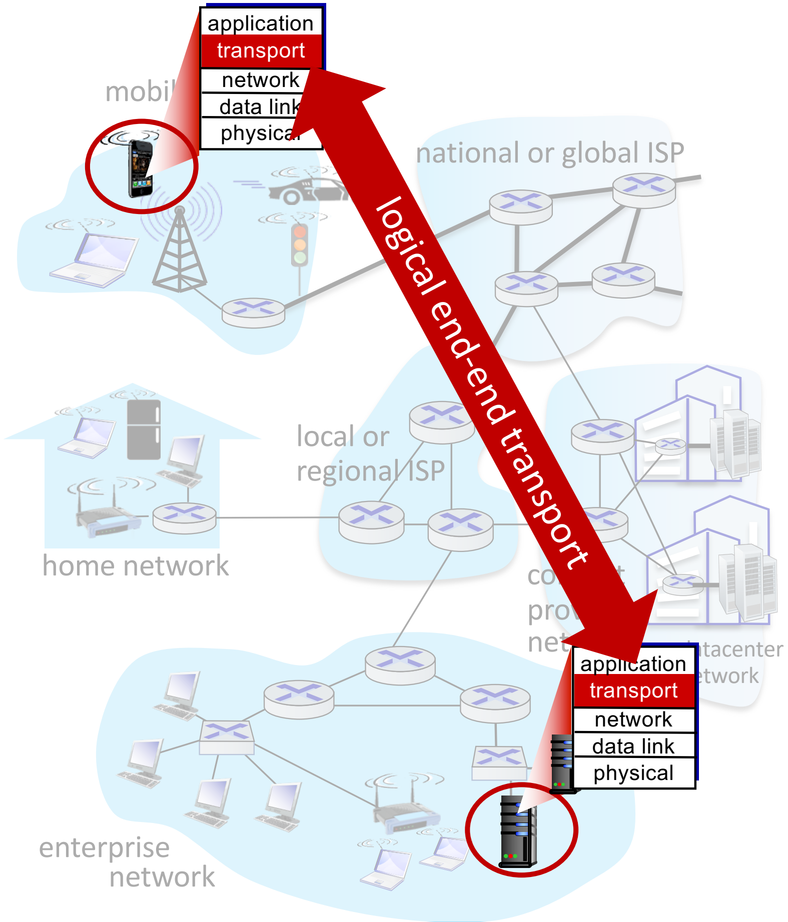
Our goal:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control

- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
  - TCP, UDP



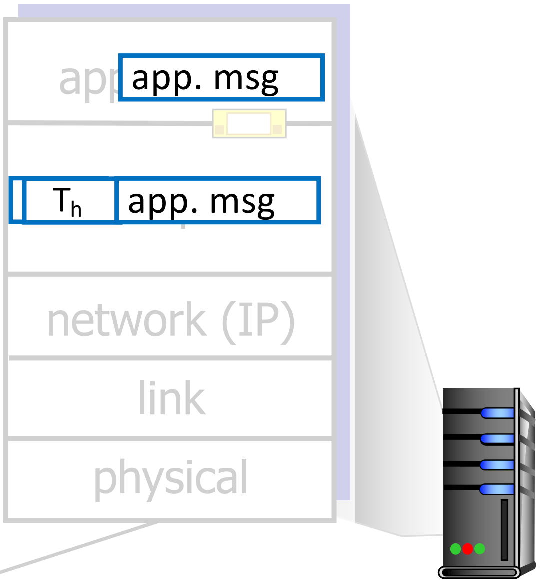
Transport vs. network layer services and protocols

- network layer: logical communication between *hosts*
- transport layer*:* logical communication between *processes*
  - relies on, enhances, network layer services

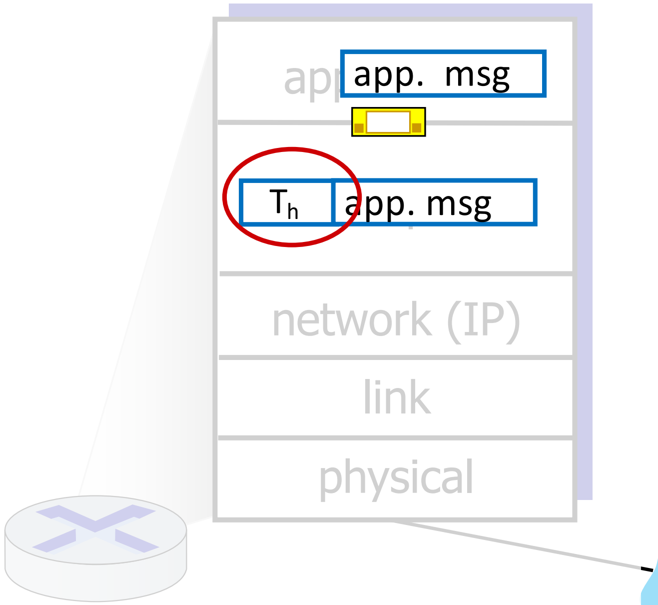
Transport Layer Actions

Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP

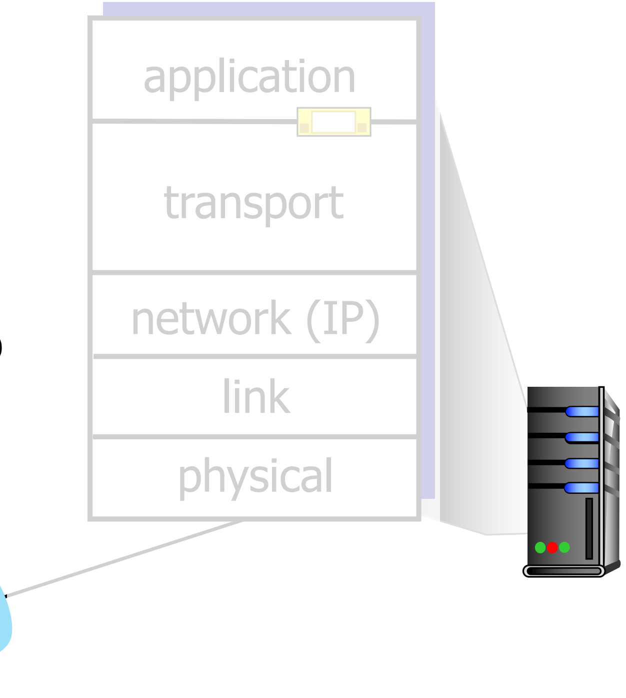


Transport Layer Actions



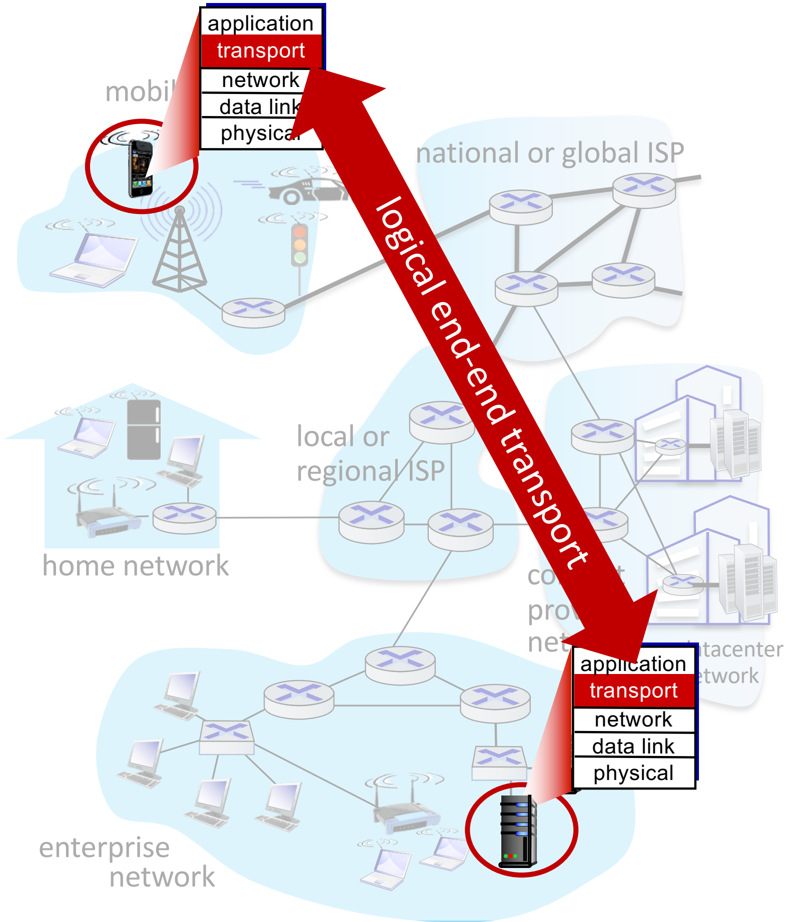
Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket

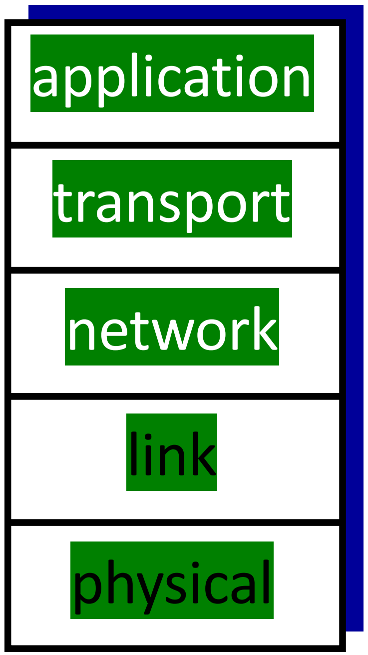


Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Troisième étape : couche Réseau

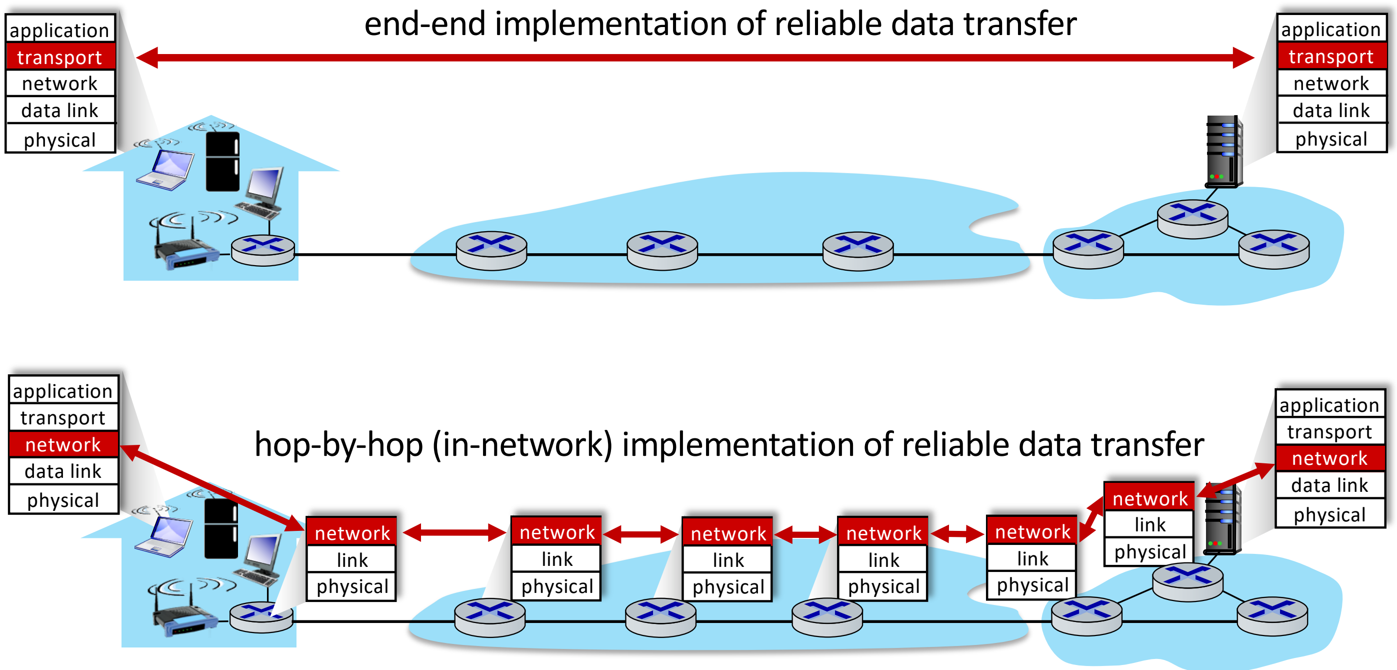


Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

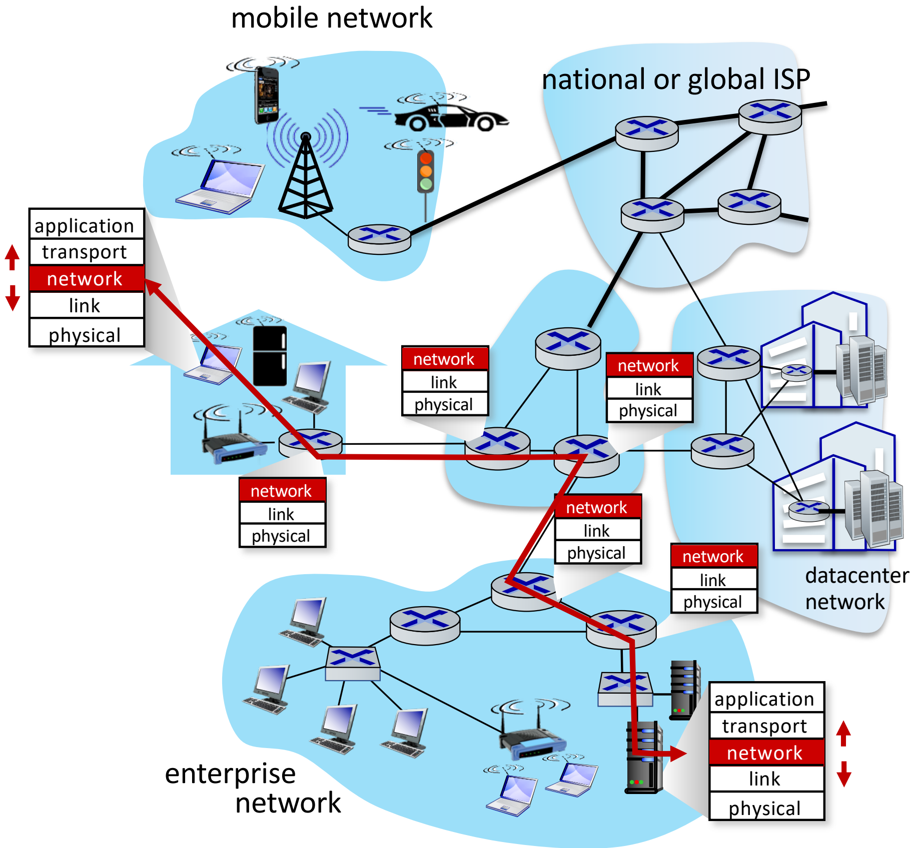
The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge



Network-layer services and protocols

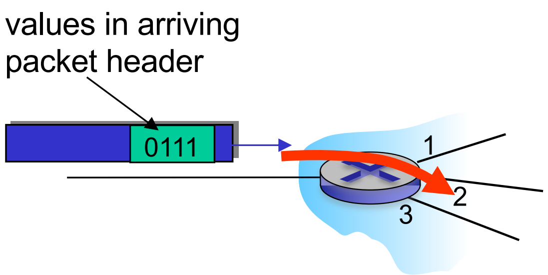
- transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer



Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

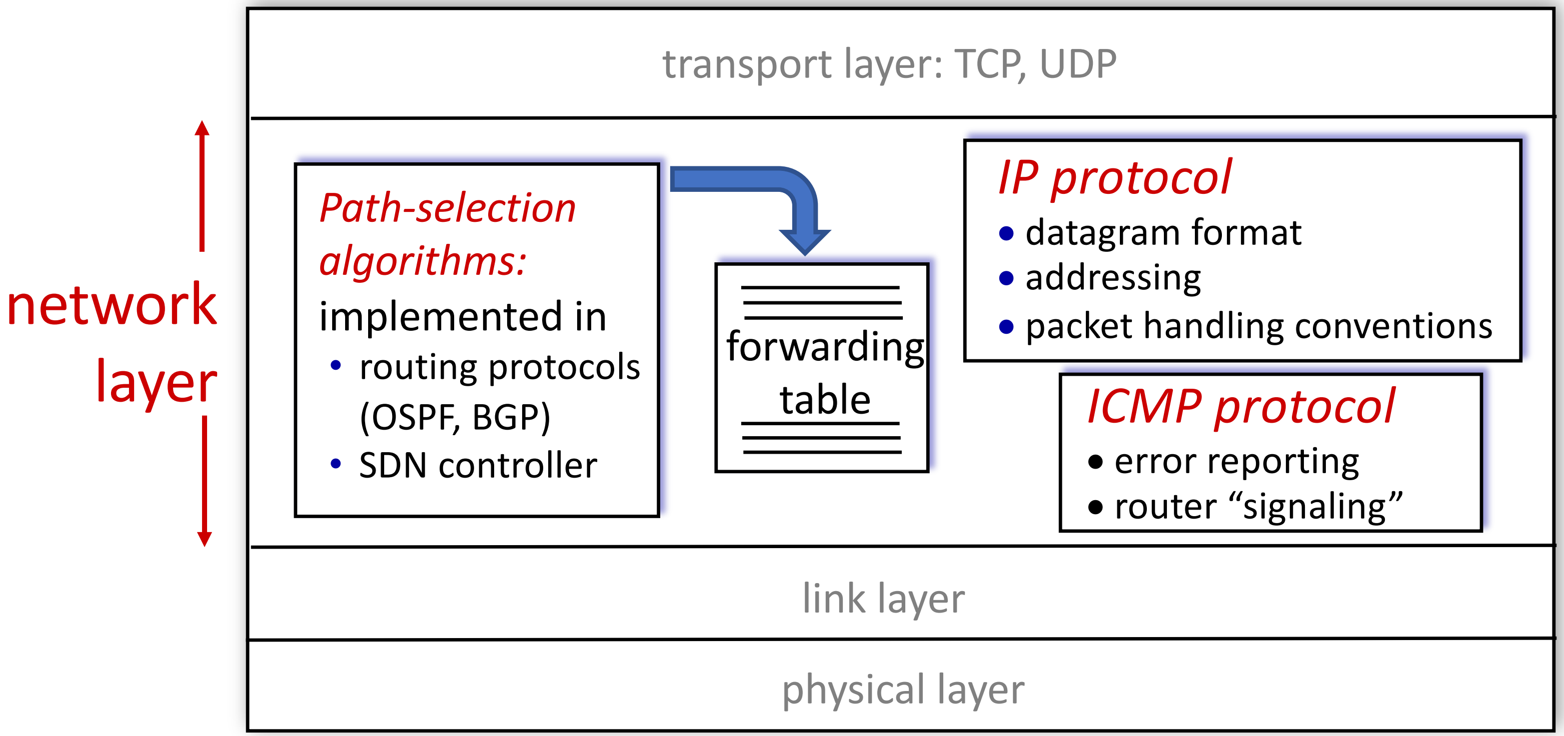


Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

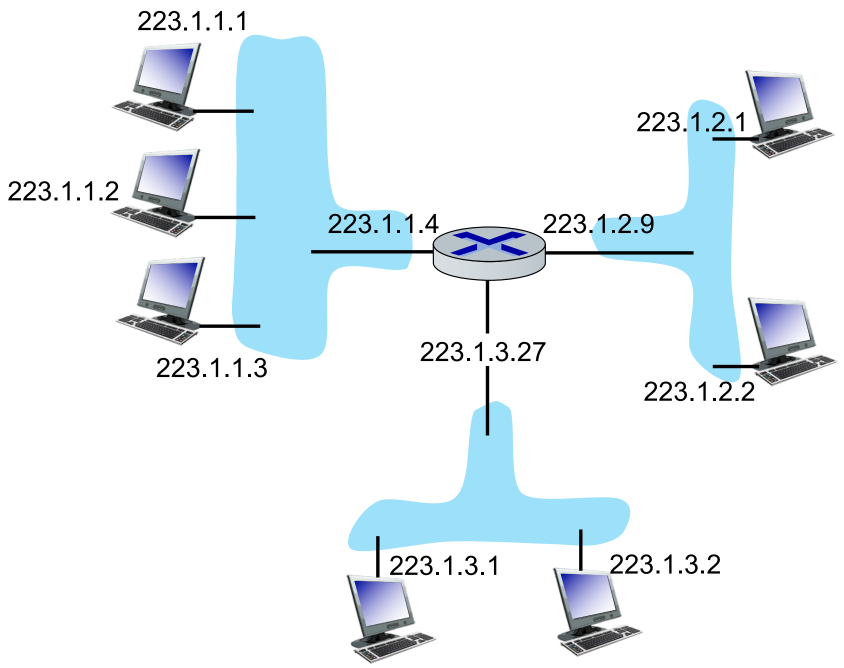
Network Layer: Internet

host, router network layer functions:



IP addressing: introduction

- IP address: 32-bit identifier associated with each host or router *interface*
- interface: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)



dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

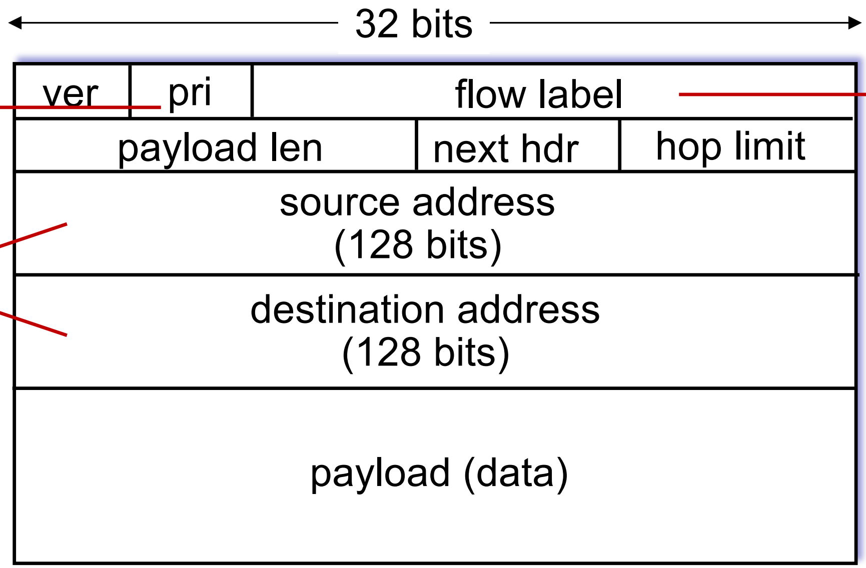
IPv6: motivation

- initial motivation: 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

IPv6 datagram format

priority: identify priority among datagrams in flow

128-bit IPv6 addresses

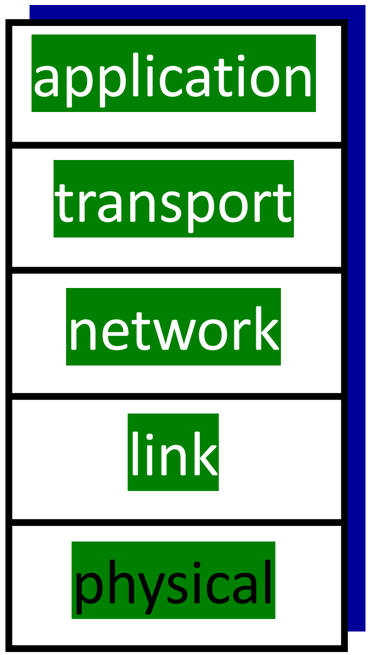


flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

Quatrième étape : couche liaison

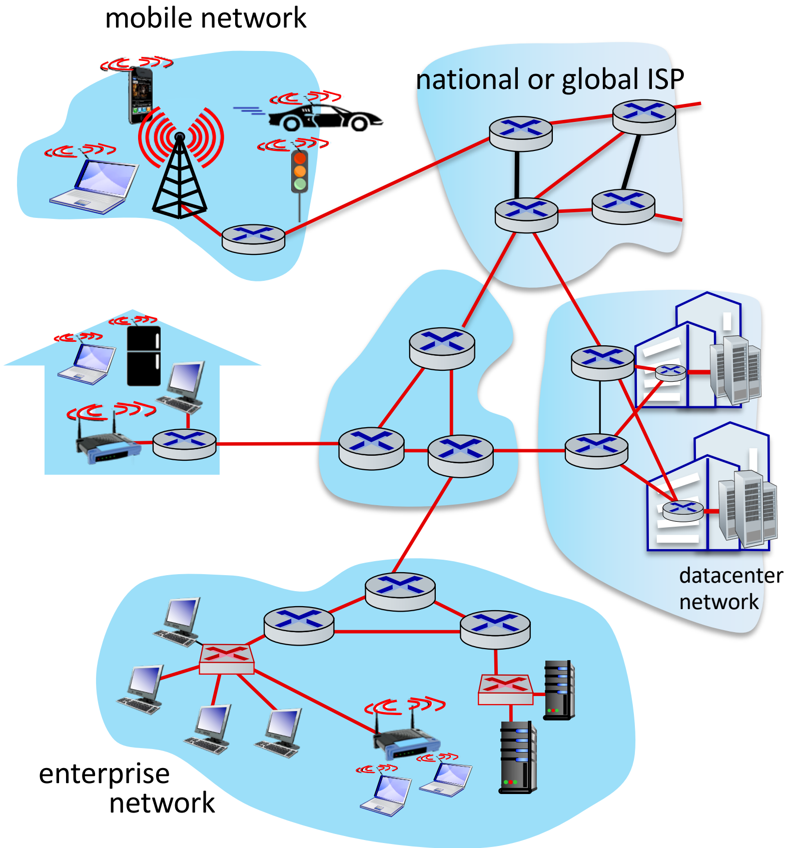


Link layer: introduction

terminology:

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired
  - wireless
  - LANs
- layer-2 packet: *frame*, encapsulates datagram

link layer has responsibility of transferring datagram from one node to physically adjacent node over a link



Link layer: context

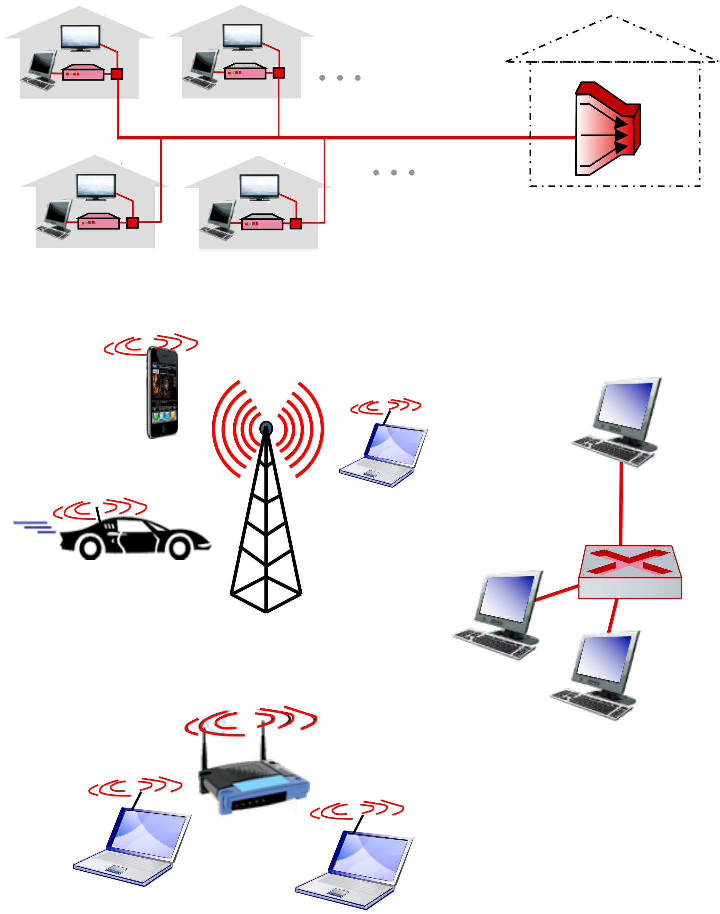
- datagram transferred by different link protocols over different links:
  - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link

transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link-layer protocol
- travel agent = routing algorithm

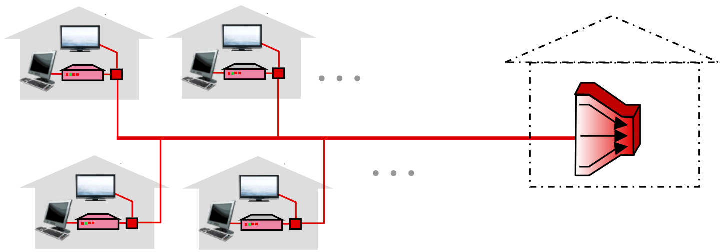
Link layer: services

- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?



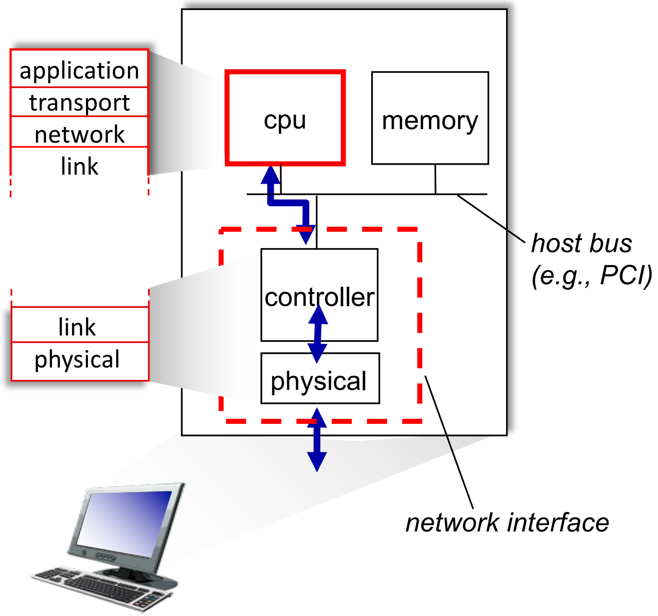
Link layer: services (more)

- **flow control:**
  - pacing between adjacent sending and receiving nodes
- **error detection:**
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- **error correction:**
  - receiver identifies *and corrects* bit error(s) without retransmission
- **half-duplex and full-duplex:**
  - with half duplex, nodes at both ends of link can transmit, but not at same time

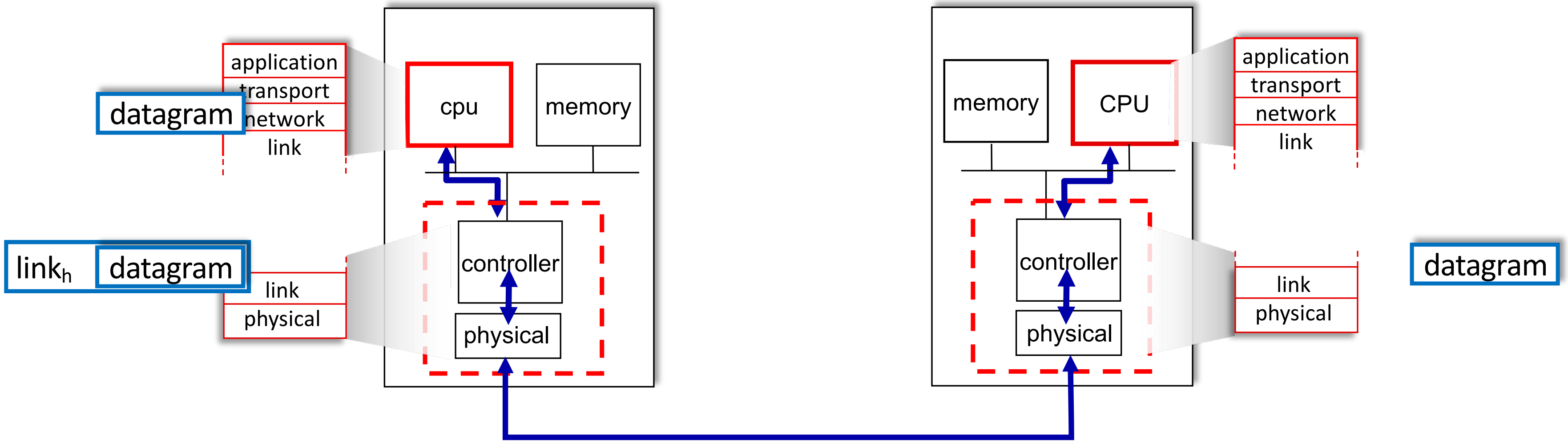


Where is the link layer implemented?

- in each-and-every host
- link layer implemented in *network interface card* (NIC) or on a chip
  - Ethernet, WiFi card or chip
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Interfaces communicating



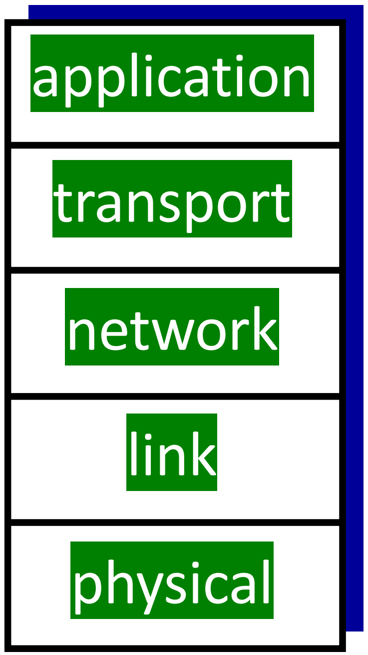
sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

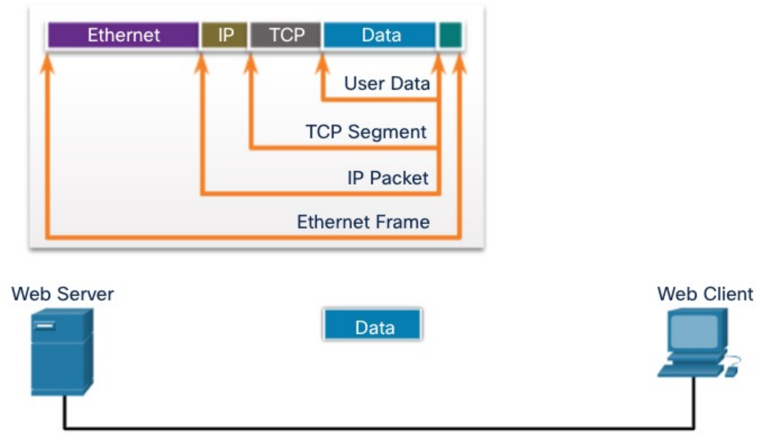
Cinquième étape : couche physique



Objectif de la couche physique

La couche physique

- Transporte des bits sur le support réseau
- Cette couche accepte une trame complète de la couche liaison de données et la code sous la forme d'une série de signaux transmis au support local.
- C'est la dernière étape du processus d'encapsulation.
- Le périphérique suivant dans le chemin d'accès à la destination reçoit les bits et re-encapsule le cadre, puis décide quoi en faire.



Terminologie de la bande passante

- **Latence**
  - Temps, y compris les retards, nécessaire pour que les données voyagent d'un point donné à un autre
- **Débit (Throughput)**
  - La mesure du transfert de bits à travers le média sur une période de temps donnée
- **Débit applicatif (Goodput)**
  - La mesure des données utilisables transférées sur une période donnée
  - Débit applicatif = Débit - frais généraux de trafic

Types de câblage en cuivre

Unshielded Twisted-Pair (UTP) Cable

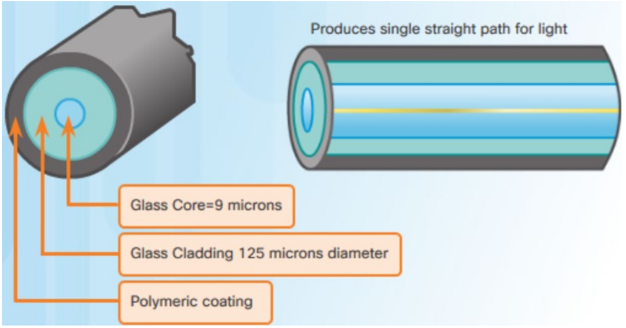
Shielded Twisted-Pair (STP) Cable

Coaxial Cable

Câblage à fibre optique

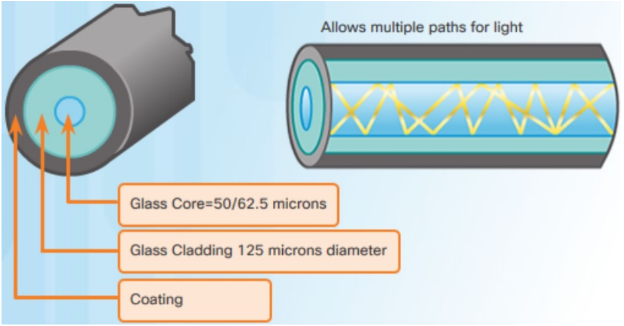
Types de supports à fibre optique

Fibre monomode



- Très petit noyau
- Utilise des lasers coûteux
- Applications longue distance

Fibre multimode



- Plus grand cœur
- Utilise des LED moins chères
- Les LED transmettent à différents angles
- Jusqu'à 10 Gbit/s sur 550 mètres

La dispersion correspond à la propagation d'une impulsion lumineuse au fil du temps. Une dispersion accrue signifie une perte accrue de puissance du signal. MMF a une plus grande dispersion que SMF, avec une distance de câble maximale pour MMF est de 550 mètres.

Fibre contre cuivre

- La fibre optique est principalement utilisée comme câblage de base pour un trafic élevé, point à point
- les connexions entre les installations de distribution de données et pour l'interconnexion des bâtiments
- dans les campus multi-bâtiments.

Problèmes de mise en œuvre	Câblage à paires torsadées non blindées (UTP)	Câblage à fibre optique
Bande passante	10 Mbit/s - 10 Gbit/s	10 Mbit/s - 100 Gbit/s
Distance	Relativement courte (1 à 100 mètres)	Relativement longue (1 à 100 000 mètres)
Résistance aux perturbations électromagnétiques et radioélectriques	Faible	Haute (résistance totale)
Résistance aux risques électriques	Faible	Haute (résistance totale)
Coûts des supports et des connecteurs	Moins élevé	Plus élevé
Compétences requises pour l'installation	Moins élevé	Plus élevé
Précautions à prendre concernant la sécurité	Moins élevé	Plus élevé

Propriétés des supports sans fil

•Il transporte des signaux électromagnétiques représentant des chiffres binaires en utilisant des fréquences radio ou micro-ondes. Cela offre la plus grande option de mobilité. Le nombre de connexions sans fil continue d'augmenter.

•Certaines des limites du sans-fil :

- **Zone de couverture** - La couverture effective peut être fortement influencée par les caractéristiques physiques du lieu de déploiement.
- **Interférence** - Le sans-fil est sensible aux interférences et peut être perturbé par de nombreux appareils courants.
- **Sécurité** - La couverture des communications sans fil ne nécessite aucun accès à un support physique, de sorte que tout le monde peut avoir accès à la transmission.
- **Support partagé** - Les réseaux locaux sans fil (WLAN) fonctionnent en semi-duplex, ce qui signifie qu'un seul appareil peut envoyer ou recevoir à la fois. L'accès simultané de nombreux utilisateurs au WLAN entraîne une réduction de la bande passante pour chaque utilisateur.