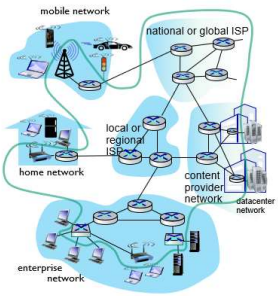
The Internet: a "nuts and bolts" view

Billions of connected computing *devices*:
- *hosts* = *end systems*
- running *network apps*

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

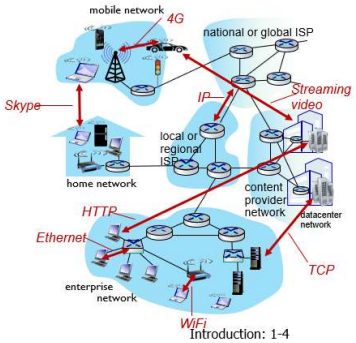
Communication links
- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

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



The Internet: a "nuts and bolts" view

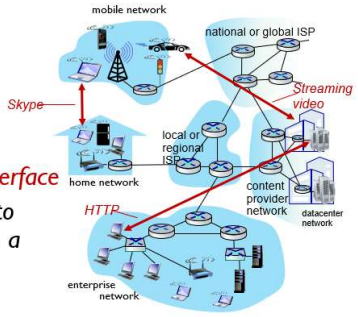
- *Internet*: "network of networks"
- *protocols* are *everywhere*
  - control sending, receiving of messages
  - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet



Internet has three ingredients: (1) hosts; (2) packet switch; (3) communication links. The three components are grouped together into a Network such as campus network, home network.	Different networks need to link together in order to communicate across different networks. Different devices have to use a common **language** in order to speak to each other. This common language is called a **protocol**. It describes what a device should do when receiving a message or command.

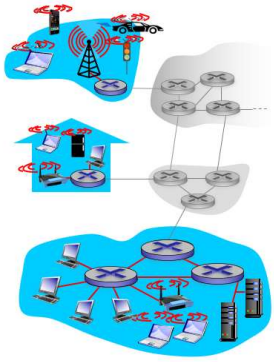
The Internet: a "services" view

- provide services to applications:
  - Web, streaming video, teleconferencing, email, games, e-commerce, social media, …
- provide *socket programming interface*
  - *Sender invokes this interface to send/receive message to/from a remote host*



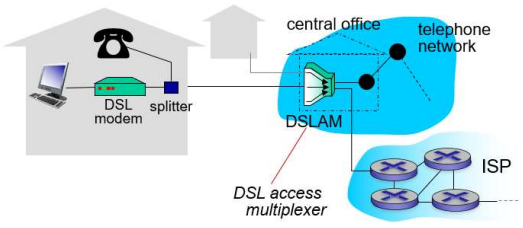
Access networks

Your connecting network
- residential access nets
- institutional access networks (school, company)
- mobile access networks



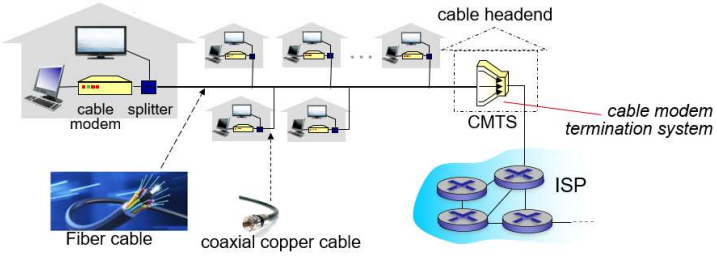
The internet can also be viewed in the service model: you use the internet by requesting a **service** from it. For programmers, it can use **socket** to take the services provided by Internet. Socket is an interface that allows to send or receive a message from remote host.	The network you first connect is called **access network**. At home, home network is your access network; on campus, the campus network is your access network. Your mobile phone uses mobile network (from phone company).

Access network: digital subscriber line (DSL)



- use *existing* telephone line to central office DSLAM

Access network: cable network

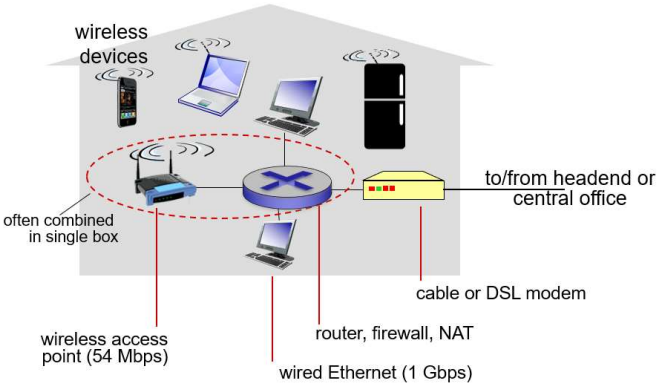


- network of cable, fiber attaches homes to ISP router
  - homes *share access network* to cable headend
  - unlike DSL, which has dedicated access to central office

DSL network: message from your computer goes through DSL modem, transforming into a modulated signal (suitable for transmission), mixed with telephone signal by splitter. It is then sent to tel company **DSL access multiplexer** (DSLAM), which separates	Cable network is different from DSL in that the combined signal from splitter is sent to the cable company device CMTS using a single cable (usually fiber cable). This cable is shared with many households.

the telephone data and internet data, and then combines internet data from other households and send them to the internet.

Access network: home network



Wireless access networks

- shared *wireless* access network connects end system to router
  - via base station aka "access point"

wireless LANs:
- within building (100 ft.)
- 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate

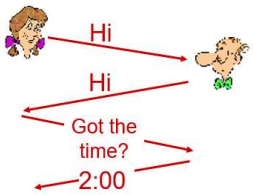
wide-area wireless access
- provided by telco (cellular) operator, 10's km
- between 1 and 10 Mbps
- 3G, 4G: LTE
- 5G

This is a home network sample. The internet data arrives at your home. It first goes through the (cable or DSL) modem. This transforms into a low frequency signal. It then tries to reach a correct destination device.
However, there are several internet devices. In this case, the router will determine which device is the correct recipient (if there is only one device, the router is not needed).
The host device could connect to router by a twisted-pair copper wire. In this case, it communicates with router using Ethernet protocol. If your host device is wireless device, it can connect to the router through a wireless access point through WIFI. The protocol from your wireless device to wireless access point is 802.11 or its variants. Sometimes wireless access point, router or even modem are integrated into one device.

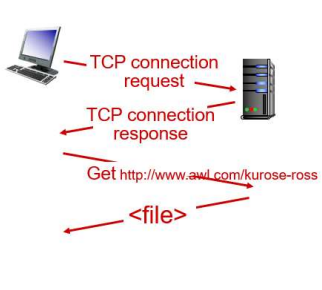
What's a protocol?

INTERNET PROTOCOL STACK

a human protocol



a computer network protocol:



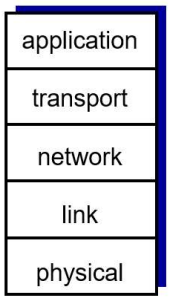
Q: other human protocols?

What's a protocol?

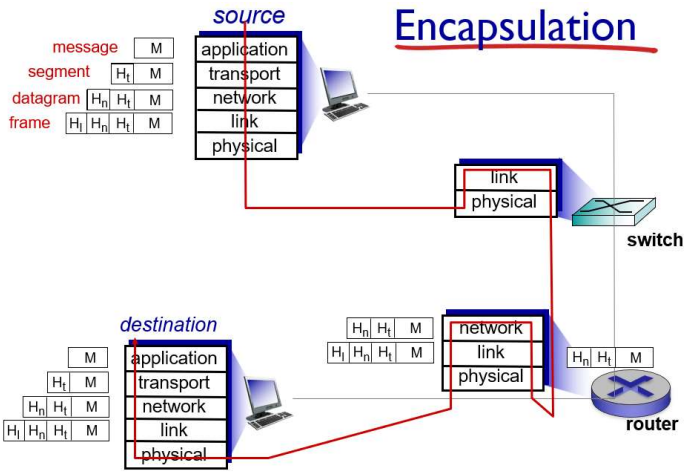
network protocols:
- machines rather than humans
- all communication activity in Internet governed by protocols
- *protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, receipt*

Internet protocol stack

- *application:* supporting network applications
  - FTP, 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, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



Encapsulation



Internet is a complex system. It is very difficult for one application to implement the whole system. So it is decomposed into five layers and each layer is communicating with the neighboring layer through the service interface. Under this view, an internet application (such as a webserver) only needs to implement the server itself. The communication with the client can be done by sending the message to the transport layer. The network specifies the routing path to the

This gives an overall picture how each layer is providing service to the lower below. Specifically, at the sending side, when one layer receives the packet from the above layer, it adds one header. At the receiver side, the

destination. Link specifies the communication between each neighboring link. Physical layer handles the physical communication (turning the binary string into the physical signal).	same layer will read this header and knows what to do.

Web page

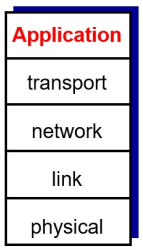
- web page consists of *base HTML-file,* which will reference to *several objects*
- object can be another HTML file, JPEG image, Java applet, audio file,…
- each object is addressable by a *URL,* e.g.,

```
www.someschool.edu/someDept/pic.gif
```

host name: `www.someschool.edu` — path name: `/someDept/pic.gif`

`/var/www/html/someDept/pic.gif`

HTTP



HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

Host name corresponds to the server's network location (in terms of IP address; the details will be given later). The path name is the directory of the file on the server. **/var/www/html** is the default location on the server.	This is the http format, send/receive through TCP connection.	When you visit www.google.com, you browser needs to first translate **www.google.com** into its IP address. This is done by **DNS protocol**. DNS is an application layer protocol.

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

/var/www/html/index.html

carriage return character

line-feed character

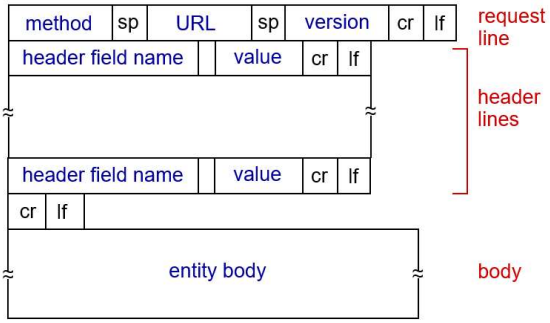
request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP request message: general format



HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this msg

301 Moved Permanently
- requested object moved, new location specified later in this msg (Location:)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

DNS: domain name system

people: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

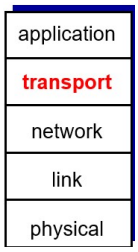
The need to translate host name to ip address.

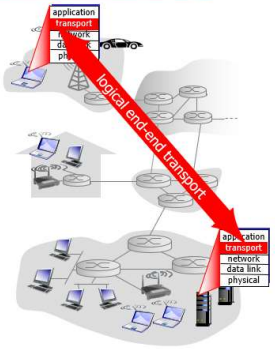
Domain Name System:
- name server
  - Resolve the name to ip address translation
- *DNS is an application-layer protocol:*
  - executed between host (your computer) and name servers to *resolve* names/ipaddress translation
  - Your host starts the DNS query at local DNS server.
  - We will go to the details in the future.

When you visit www.google.com, you browser needs to first translate **www.google.com** into its ip address.
This is done by DNS protocol.
DNS is an application layer protocol.

Transport services and protocols

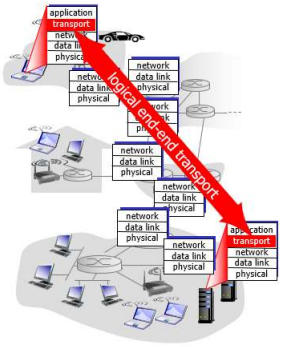
- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP





Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Between sender transport layer and receiver transport layer, you can look it as a virtual communication channel. The basic functionality for this channel is**: 1.** receive the message from application process and **2.** break it into pieces and **3.** send into the channel. At the receiver side, it reassembles the pieces into the original messages. Typically, we have two transport layer protocols: TCP and UDP. They provide different quality of services.	TCP and UDP provide different kind of services. **Congestion control** makes sure the mistake due to the communication channel can be recovered. **Flow control** makes sure the sender does not send messages too fast so that the receiver can not handle it timely. UDP almost does nothing. All these properties will be clear by looking at the packet header soon. Delay and bandwidth can not be guaranteed as it really depends on the channel usage by other users (e.g., more users in the channel indicates longer channel delay).

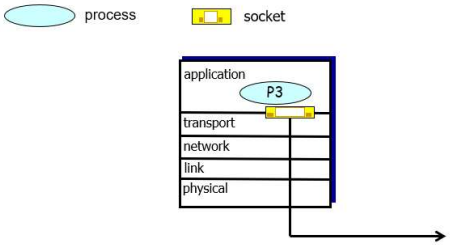
Process is a **running** application. Application process access the transport layer service using socket interfaces: **socket.send()** to send a message and **socket.recv()** to receive a message.

When you run **socket.send(M)**, message M will be put at the sending buffer and is available to transport layer program while you run **M=socket.recv(),** you will retrieve the message M from receive buffer that is delivered by transport layer.

Different application has its own socket and its own send/receive buffer. The transport layer's job is to **take messages from each application's buffer** and **add header** and **send out**, at the receiver sends remove headers and place message at the correct receive buffer.

Sockets

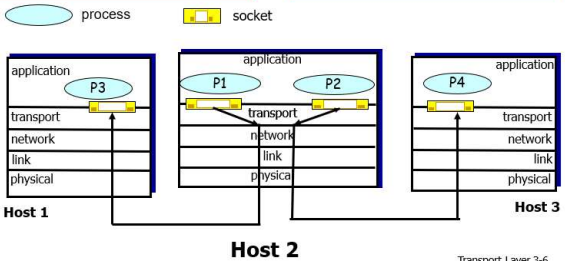
- Application processes sends messages to (or receives messages from) transport layer through socket.
- socket programming is for this purpose: socket.send(M) and M=socket.recv()



Multiplexing/demultiplexing

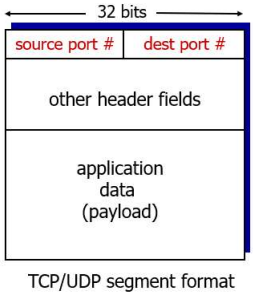
multiplexing at sender: handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver: use header info to deliver received segments to correct socket



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

UDP demultiplexing

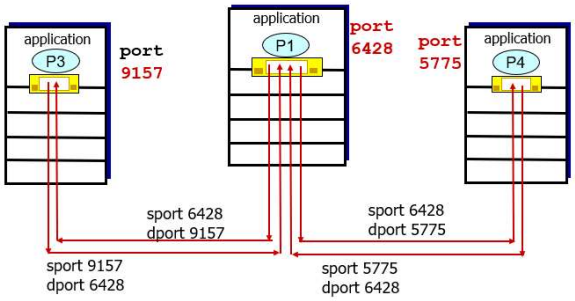
- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

IP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

- a UDP server (e.g., DNS) with two clients is an example

The question is how the transport layer decide which socket is the **correct** receiver. This answer is to use (**sender IP, receiver IP, sender port, receive port**) to determine the correct receiver socket.	Different UDP socket has a different port #. So the transport layer can use the port # to decide which socket is the **correct** receiving socket.	TCP de-multiplexing requires the four tuple. **For example**, HTTP server assigns a **different** socket for each client but all theses socket use the same port 80. So you can not only use port 80 to determine the receiving socket.

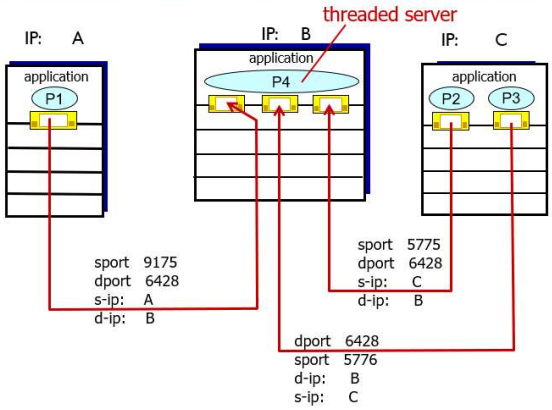
UDP demux: example



TCP demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client

TCP demux: example



UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

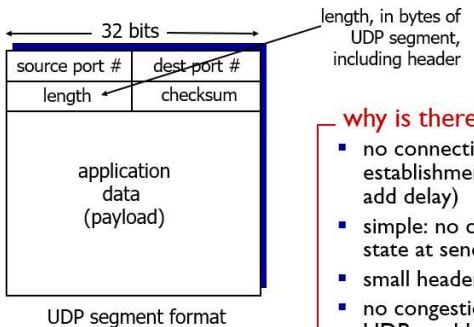
This is the properties for UDP. They will be clear by looking at the UDP header.

Now you can check why the UDP has the properties mentioned before:

- **connectionless**, because no field is concerned with connection;
- **could be lost or out of order**: no field to recover the sending order
- **no error correction**: no field is used to correct error.

UDP checksum is used to **detect** error but not **correct** error. The sender calculates the checksum put it in the UDP header; the receiver recalculates the checksum to verify if it matches the checksum in UDP header. If the packet has error, the checksum should be different. **Detection** is quite different from **correction**. It is quite similar to the scenario: You can easily verify someone's solution to a problem is wrong but you do not know how to correct.

UDP: segment header



UDP segment format

why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

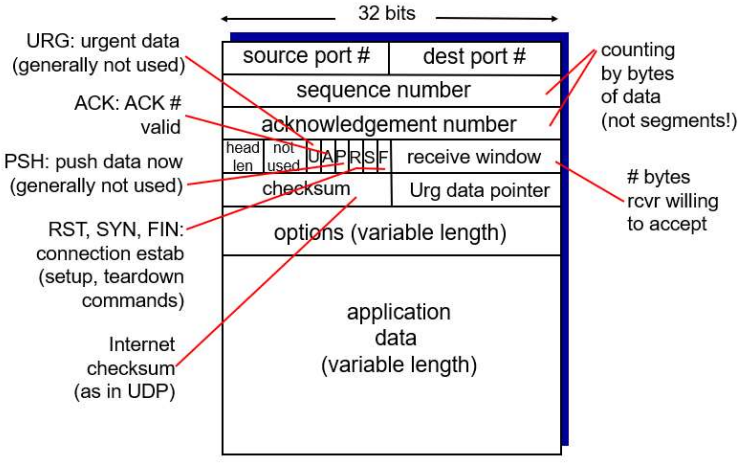
receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors*

TCP segment structure

TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver



TCP has these properties. Again, you can verify the properties based on the TCP header only. **No message boundary**: when application invokes socket.send(M1), socket.send(M2) and socket.send(M3), M1|M2|M3 will be put on sending buffer. In this case, M1, M2, M3 are put together with no boundary between them. So TCP could just take M1M2M3 to make **one** packet to send out.

Other properties will be explained in details with the TCP header.

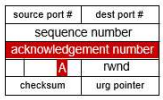
seq. # and ACK

sequence numbers:

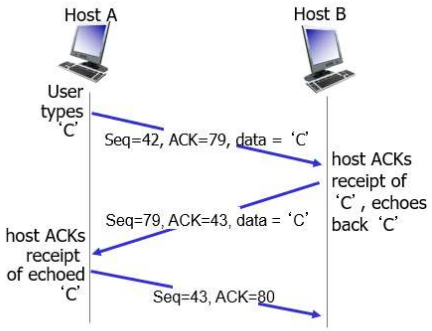
- Initial seq # can be any
- Seq # (**next** packet) =seq# (**current** packet)+ #databytes (**current** packet)
- nex seq# >current seq #.
- seq# can be used to recover the packet order.

Acknowledge number:

- ack_num=seq# of next packet expected from other side
- **ack_num=502**:
  please send your packet with seq#=502.
- means packets with seq#<502 have been received.



TCP seq. numbers, ACKs



simple telnet scenario

Sequence number: The sequence number in the format is used to make sure the packets are received by the recipient in order. At the sender side, the intial sequence # is picked randomly. If the current outgoing packet has a sequence # **s** and the data size of **Data** is **N bytes**, then the sequence # of the next outgoing packet is **s+N.** It should emphasized that due to the difference of routing paths for different packets, a packet will larger sequence number might arrive at the destination earlier than a packet with a smaller sequence #. But looking at the packet sequence numbers, the receiver can restore the original packet order.

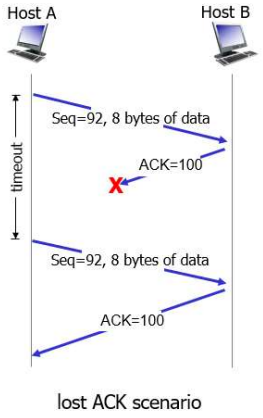
Acknowledge Number: The acknowledge number. The acknowledgement number is to tell the other side the sequence # of the next packet expected. **ack_num=next_seq#** of the **received** packet

timeout	If the segment is lost in the transmission, the sender has to retransmit the packet (to maintain the reliability). **Ques.** : How can the sender realize that the packet is lost? **Ans.** The answer is timeout. You set a timeout when sending out a packet. When the acknowledgement is not returned after timeout, you can think it is lost (even if it actually is not) and do the retransmission.
• If a segment is not acked, then the packet might be lost and so sender needs to retransmit it. ▪ Sender needs to set a *timeout* (e.g., 5s). After waiting for this length of time, he needs to retransmit the segment.	

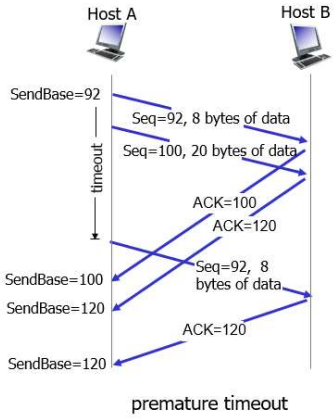
When a packet arrives at IP layer, IP layer processes it and removes the IP header. This results in the TCP segment and will be given to transport layer. Transport layers then processes this segment and removes the TCP header and delivers the resulting application data to TCP socket at socket's receive buffer. When application program wants to read data from socket. It invokes read function such as recv() to get it. In this case, the data will be removed from buffer. It is possible that the reading speed is

slower than the delivering speed. In this case, the receive buffer will overflow. So we need a flow control to tell the sender to slow down the transmission. The **receive_window** field in TCP header in the packet replying the sender is for this purpose.

TCP: retransmission scenarios

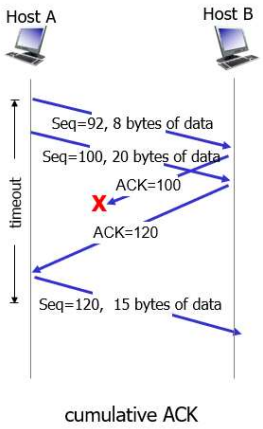


lost ACK scenario



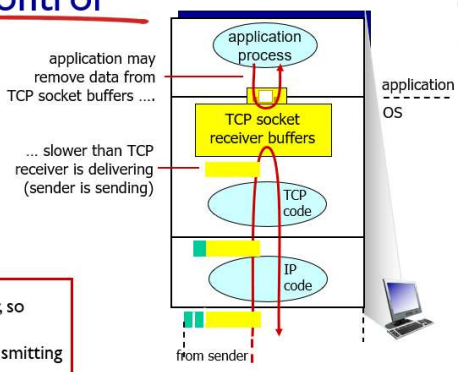
premature timeout

TCP: retransmission scenarios



cumulative ACK

TCP flow control

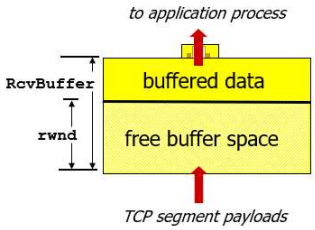


receiver protocol stack

flow control
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP flow control

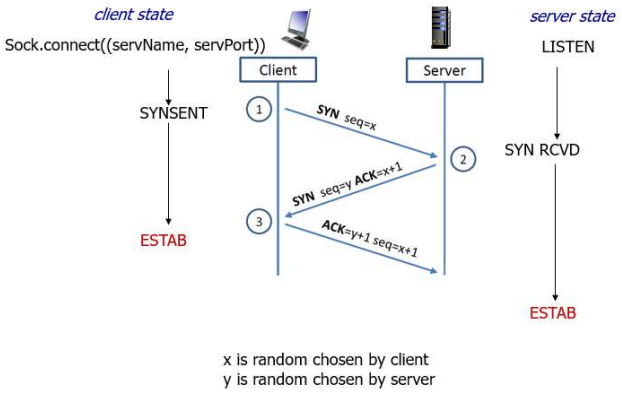
- receiver puts free buffer size `rwnd` in TCP header of receiver-to-sender segments
  - `RcvBuffer=4096bytes` (typical default)
- If sender receives a packet with small rwnd, it reduces the sending speed.
- This guarantees receive buffer will not overflow



receiver-side buffering

If the socket receiving buffer of host A has free space of 3000bytes, then it puts 3000 in the **receive window** field of TCP segment to its connected host B. Then, host B will make sure that the number of bytes on the way to host A should not be more than 3000. To do this, the sender makes sure **sent_but_unacknowled** data size totally no more than **receive_window** value.

TCP 3-way Handshake Protocol



TCP: closing a connection

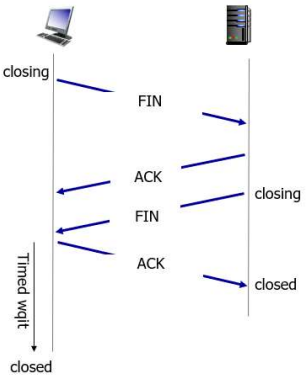
client closes socket:
close(fd);

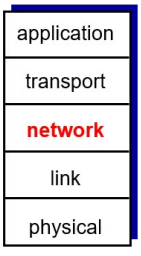
Step 1: client sends FIN segment to server to close C-> S direction

Step 2: server receives FIN, replies with ACK and also FIN to close the S->C direction.

Step 3: client receives FIN, replies with ACK and enters timed-wait state.

Step 4: server receives FIN and enter closed state.





SYN packet means only TCP with header with SYN bit =1 while the other flagbits are all **0**. **SYNACK** packet means **SYN** bit and **ACK** bit are 1 while other flagbits are 0. The three-way handshake is run when the client executes **sock.connect**() function. We will return to this protocol in TCP attack later.	FIN packet means FIN bit = 1. The protocol is run when socket executes **close()** function.

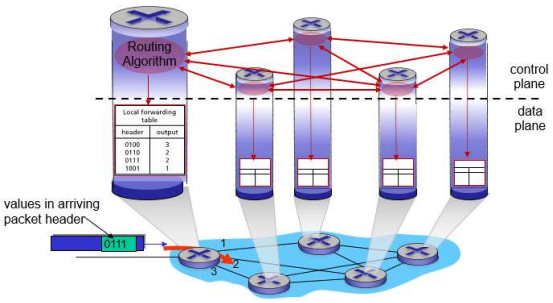
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane

Two network-layer functions

network-layer functions:

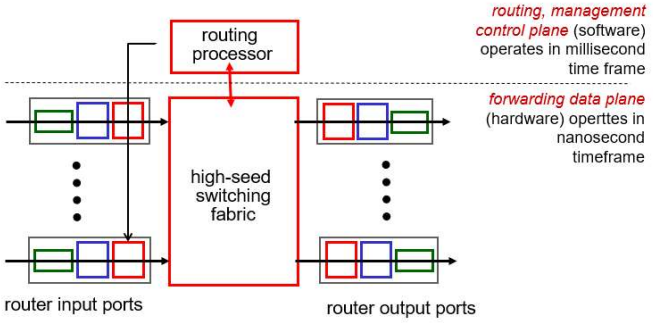
- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*



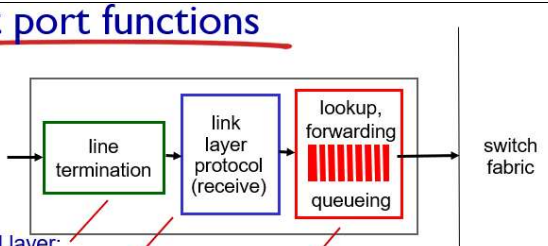
When a packet comes to a router with destination **0111**, the router needs to decide which outgoing link should take. To do this, it prepares a **routing table**: each destination ip is suggested an outgoing link. The routing table is prepared by routing algorithm, jointly run among a set of routers.	This is the architecture of a router, consisting of input ports, output ports and switching fabric controlled by routing processor. Input port takes in the incoming packet and output port sends out the outgoing packet. The switching fabric, under the routing processor control, shifts the packet at input port to the designated output port.	Input port has three parts: **1. Line termination unit**: deal with the physical layer processing, converting the physical signal into bit strings. **2. Link lay protocol**: deal with the link layer processing (to be detailed in the next lecture) **3. Lookup and forwarding:** This looks up the routing table to find out which output link should the packet be moved to and then move to that port. The routing table could be destination-based, or, generalized rule based (which consider other fields such as dest port # in the packet header).

Router architecture overview

- high-level view of generic router architecture:



Input port functions



physical layer: bit-level reception

data link layer: e.g., Ethernet (chapter 5)

decentralized switching:

- *destination-based forwarding:* forward based only on destination IP address (traditional)
- *generalized forwarding:* forward based on any set of header field values
- queuing: if datagrams arrive faster than forwarding rate into switch fabric
  - if queue is full, the arriving packet is dropped.

Destination-based forwarding

forwarding table

Destination IP Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Longest prefix matching

longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination IP Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

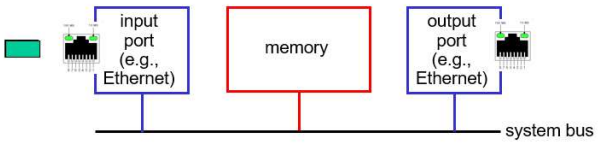
DA: 11001000 00010111 00011000 10101010 which interface?

This is the **destination-based routing table** (also called **forwarding table**). The IP address is in 32-bit format.	The routing table in each category is simplified into a pattern. Note that interface 2 should exclude the pattern 11001000 00010111 00011000 *********, which is for Interface 1. However, all such IP addresses will choose interface 1 (as it has longer match than interface 2).

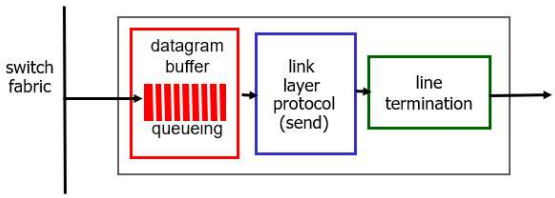
Switching via memory

first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory



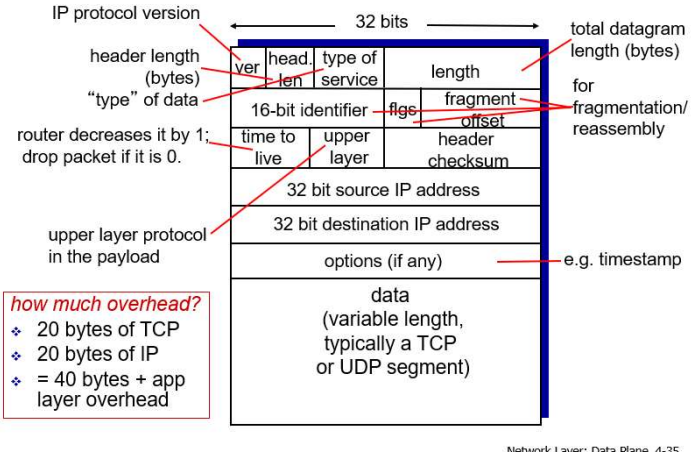
Output ports



- *buffering* (or a queue) required when datagrams via switching is faster than the outgoing transmission
- Packet will be dropped if the queue is full.

This is a special case of switching from **input port to output port**. This is done using **system memory under CPU control**: first **copy to system memory** and then **move to the output port.**	This is IPv4 header. You pay attention to **upper layer, IP addresses and TTL fields.** Upper layer protocol is usually **TCP or UDP or ICMP**. **ID, flags, fragment offset** is for fragmentation purpose, which is introduced in the next slide.

IP datagram format

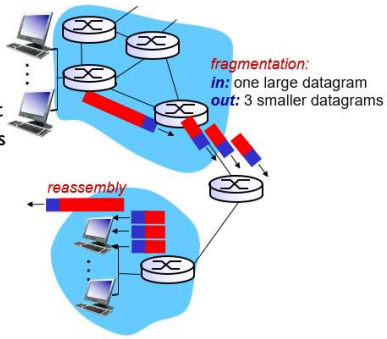


how much overhead?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

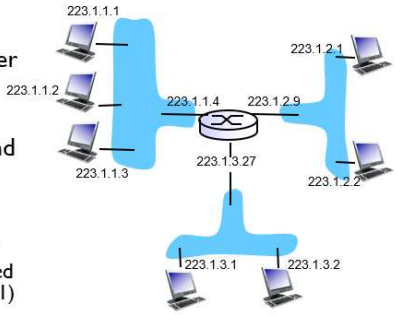
IP fragmentation, reassembly

- network links have MTU (max transfer size)
  - vary on link types
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments



IP addressing: introduction

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

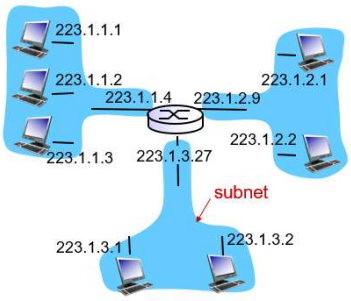


223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*



network consisting of 3 subnets

IP address is defined for **interface** (not for a **device**). IP address has decimal dotted format or 32-bit string format.	223.1.1.0/24 is the subnet that the first 24 bits are common for all devices in this network. Since 223.1.1 has 24 bits, this subnet has IP address of pattern 223.1.1.x. 223.1.1.0 is written into binary format as **11011111 0000000**1 **00000001** 00000000 (bold parts are 24 bits).

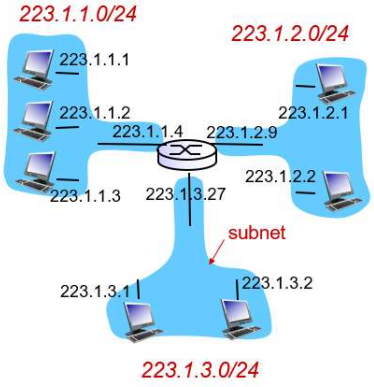
DHCP: Dynamic Host Configuration Protocol

goal: allow host to *dynamically* obtain its IP address from network server when it joins network

Subnets

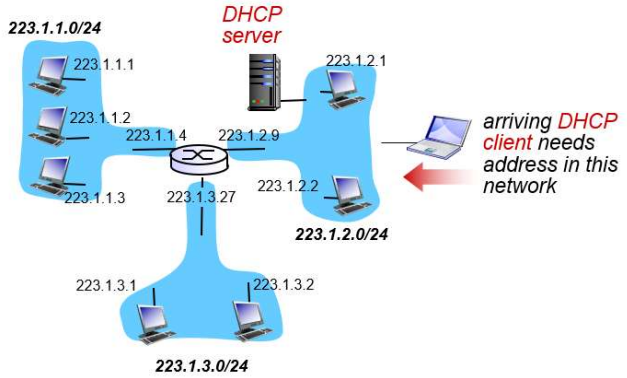
recipe

- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*

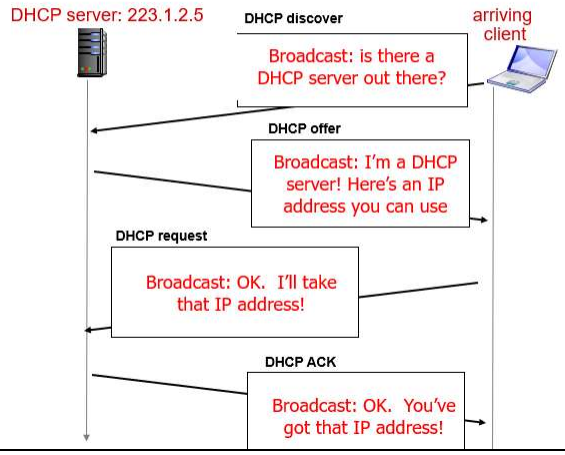


subnet mask: /24

DHCP client-server scenario



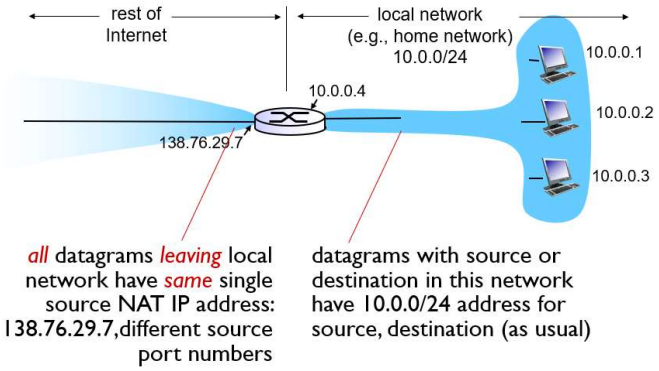
DHCP client-server scenario



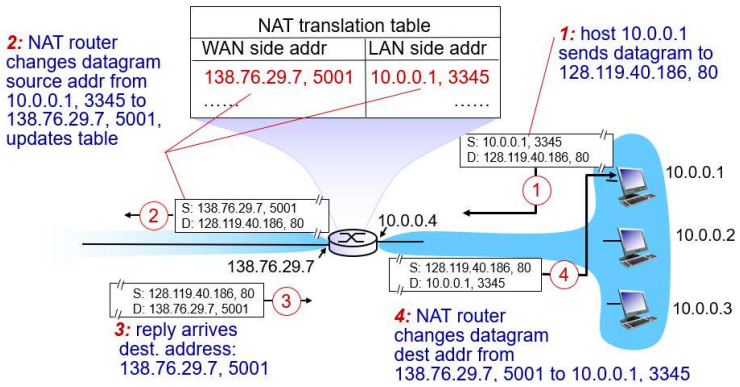
Most importantly, DHCP protocol will provide you not just the ip address. It provides more:

- IP address for you to use
- Local DNS server IP address (recall, whenever you type www.google.com, your browser will first find the IP address of google by issuing a DNS query to a local DNS server).
- Local router IP address (whenever you computer needs to sends a packet out, it first needs to reach the router which then routes your packet to the internet and finally reaches the destination).
- DHCP server IP address

NAT: network address translation



NAT: network address translation



NAT concerns the following problem: in your home network, the IP address has the format of 10.0.0.*. This ip address is not public. So the IP address is not reachable from the Internet. To resolve this issue, all devices in the home network should use the internet IP address of the router when its packets travel outside of the home network. To distinguish each device in the home network, a different port is used.	IP address translation is maintained at the router using a NAT translation table. Our SEEDLAB has the network of 10.0.2.0/24. Your host computer can be regarded as router. Find the out NAT translation table for an application (such DNS).

IPv6: motivation

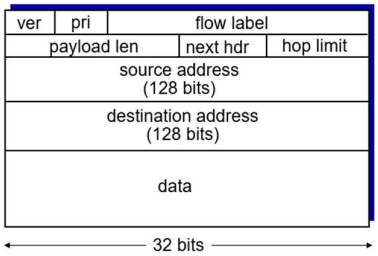
- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

IPv6 datagram format

priority: identify priority among datagrams in flow
flow Label: identify datagrams in same "flow."
(concept of"flow" not well defined).
next header: identify upper layer protocol for data

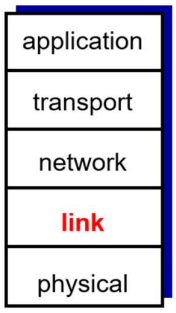


Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

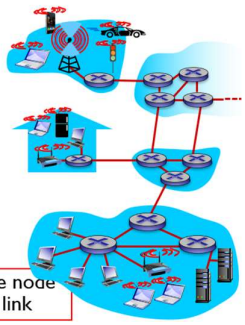
- "good": least "cost", "fastest", "least congested"

Link layer: introduction



terminology:

- hosts and routers: nodes
- Links:
  - wired links
  - wireless links
- layer-2 packet: frame, encapsulates datagram



data-link layer: transfer frame from one node to *physically connected neighbor* over a link

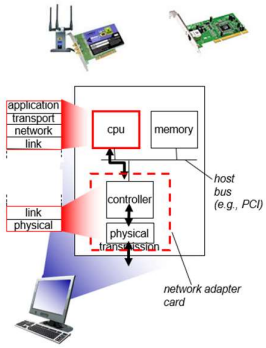
8	Link layer is concerned with the communication between the neighboring link (either wireless or wired link).
11	**Basic task for link layer communication: i.** sender adds error-checking bits et al and **ii.** sends to the receiver (of the neighboring device). The receiver verifies the link layer header and sends the enclosed packet (datagram) to the upper layer (mostly network layer).
12	If the media from the sender to receiver is **point-to-point connection**, then it is simple via **point-to-point communication**. But there are some medias that are shared.
13	In this case, if two users send the packet at the same time, they will interference each other. So **multiple access protocol** is used to determine the order of the channel access.
14	This page shows the difference between IP address and MAC address. **IP address is not portable.**

Link layer services

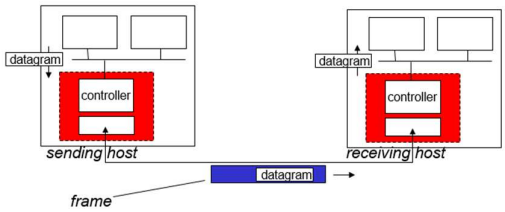
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - MAC addresses
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - Different from chapter 3!
  - solve high error rate and has error correction

Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer



Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side

Multiple access links, protocols

two types of "links":

- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or medium)*
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN

Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: *collision*

multiple access protocol: to avoid collision

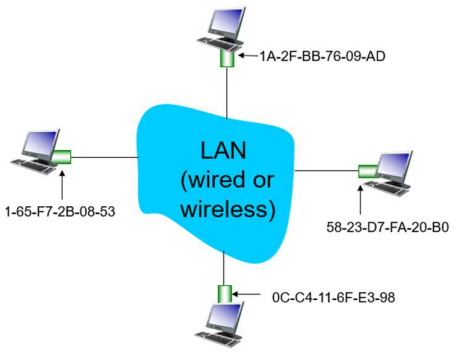
- determine whose turn to use the channel

MAC addresses and ARP

- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
  - function: *used to identify an interface*
  - 48 bit MAC address
  - e.g.: 1A-2F-BB-76-09-AD (hexadecimal (base 16) notation (each "numeral" represents 4 bits))
  - MAC address portable while IP not portable

MAC address

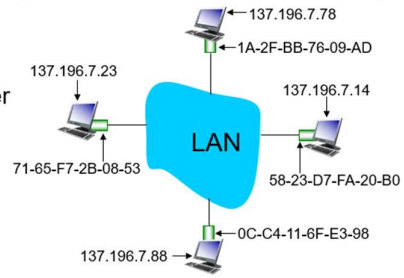
each adapter on LAN has unique *MAC* address



adapter

ARP: address resolution protocol

Question: how to obtain the MAC address of an IP address?
Answer: ARP protocol



each node builds *ARP table*:

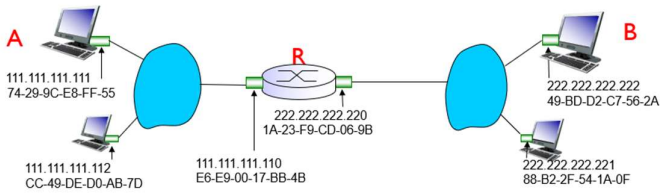
- < IP address; MAC address; TTL>
- TTL (Time To Live): expired record will be deleted (e.g. 20 min)

15	MAC address is attached to an interface.
16	Since link layer communication uses frame (i.e., link layer packet) which contains the MAC address. If A wants to send a frame to B, A has to first obtain the MAC address of B. This uses ARP protocol to obtain this address. It is important that this protocol is running only **within** the LAN (such as your home network), because different subnets are **separated** by router and so you **never** need to send a frame to a device outside your LAN (**recall** that link layer communication only sends a frame to its neighboring device).

ARP protocol: same LAN

- A wants the MAC address of B.
- A broadcasts ARP query packet, containing B's IP address
  - destination MAC address = FF-FF-FF-FF-FF-FF
- Receiving ARP packet, B replies to A with its (B's) MAC address, by unicast

- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*



Addressing: routing to another LAN

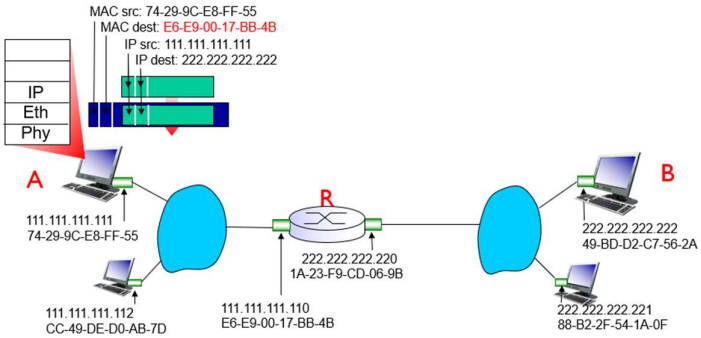
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)

17	This is how the ARP protocol is working. The basic idea is to broadcast the question (identified by IP address) and the IP address owner will reply. **The important point is that there is no configuration is required for this protocol.**
18	This page shows you how hosts from different subnets communicate. That is, how the linker MAC addresses change along the path. You only need to stick to one point: **neighboring communication uses the neighboring devices' MAC addresses.**

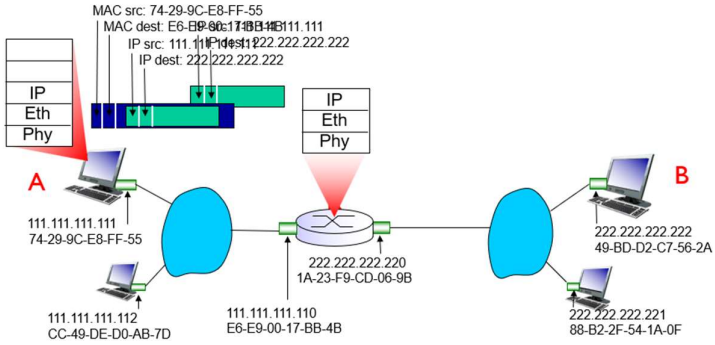
Addressing: routing to another LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



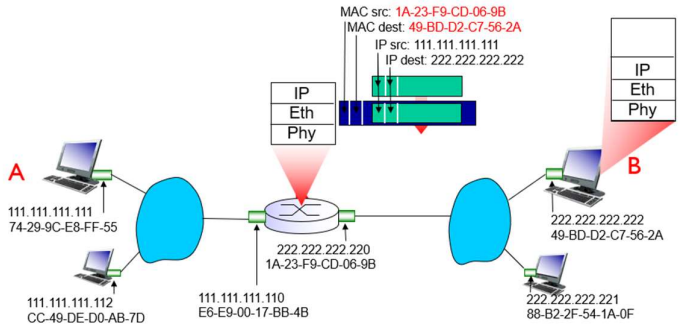
Addressing: routing to another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



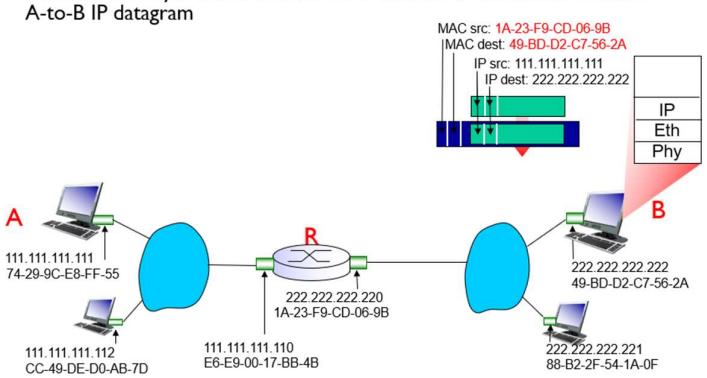
Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



24	**Ethernet protocol** is a very basic and essential **link layer communication protocol**. This paper is concerned with the shared media. This is the link layer header (for the Ethernet communication link). The wireless link will have a different header, which will not be considered in this overview.
26	Previously, we said link layer communication only sends a packet between neighboring device. This is not accurate: a link layer switch can help two indirectly connected devices to have link layer communication, by forwarding their packets back and forth without changing anything. So in view of the communication devices, they are directly connected.
27	A-A' and B-B' communication have no shared source/destion. Switch can deliver the packet simultaneously.

frame structure for Ethernet link

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

Ethernet frame structure (more)

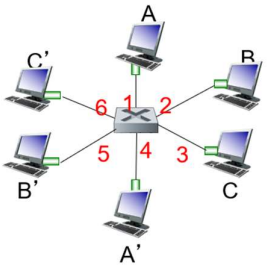
- *addresses:* 6 byte source, destination MAC addresses
  - if adapter sees its MAC address as the destination address, or sees broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- *type:* indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- *CRC:* **C**yclic **R**edundancy **C**heck at receiver
  - error detected: frame is dropped

Ethernet switch

- link-layer device:
  - store, forward Ethernet frames
  - examine and forward the incoming frame to its destination MAC address
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

Switch: *multiple* simultaneous transmissions

- hosts have dedicated, direct connection to switch
- *switching:* A-to-A' and B-to-B' can transmit simultaneously, without collisions

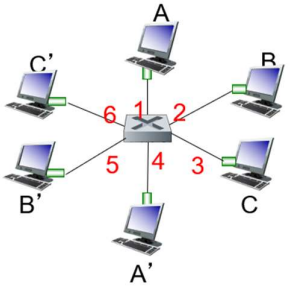


switch with six interfaces (1,2,3,4,5,6)

Switch forwarding table

Q: how does switch know A' reachable via interface 4?

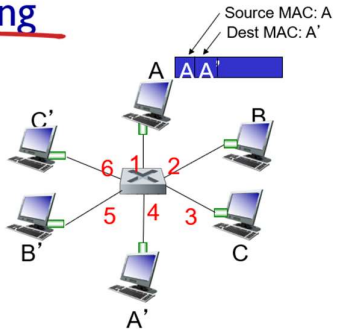
- A: each switch has a switch table, each entry:
  - (MAC addr, interface, TTL)
  - looks like a routing table!



switch with six interfaces (1,2,3,4,5,6)

Switch: self-learning

- switch *learns* which hosts can be reached through
  - an incoming frame from the sending host
  - records (mac address, interface) pair in switch table



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

28	It is better to compare this switch table with arp table and routing table. They are built differently. • **ARP table** is built by **requesting the ip holder to provide the MAC address**. • **Routing table** is built by **running routing algorithm with several routers.** • **Switch table** is built by **receiving the incoming packet and record the (MAC address, link interface) pair**; see the next slides for details.
30	You can first skip this and look at the example on the next slide and then come back to this general algorithm.

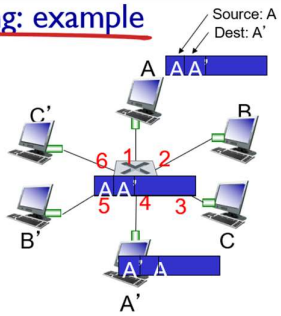
Switch: frame filtering/forwarding

when **frame** received at switch:

1. record (MAC addr, incoming link) of **sender** in switch table
2. if entry for **destination** MAC of **frame** found in the table
   then {
   if destination is on the incoming link
   then drop **frame**
   else forward **frame** on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

Self-learning, forwarding: example

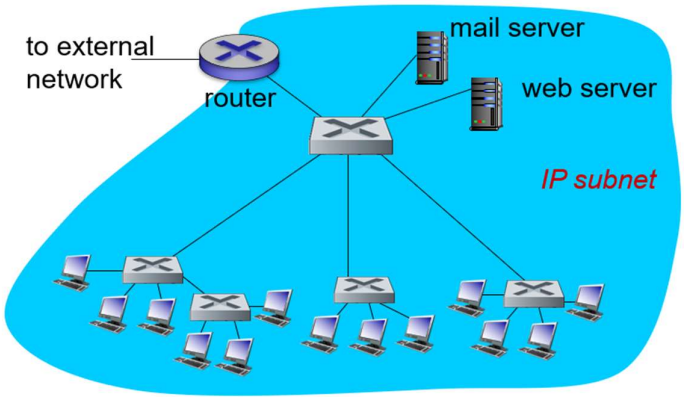
- frame destination, A', location unknown: *flood*
- destination A location known: selectively send on just one link



MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

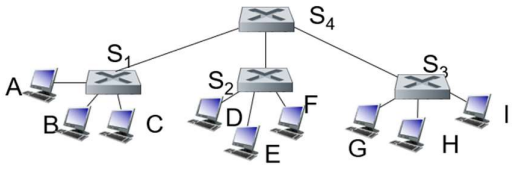
switch table (initially empty)

Institutional network



Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



- Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4