

Introduction to Network _ Layer Architecture

OSI System vs TCP/IP

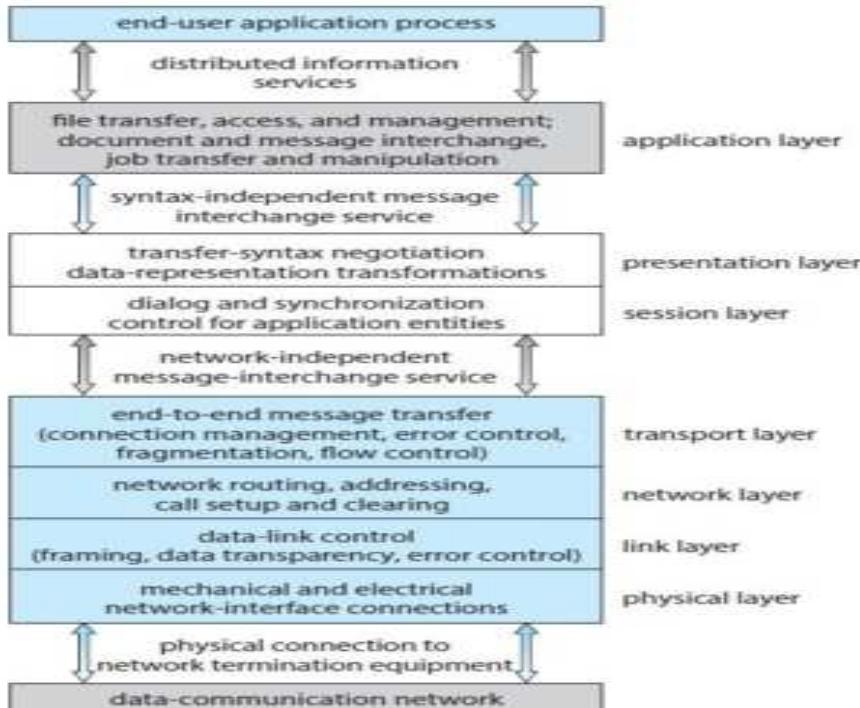
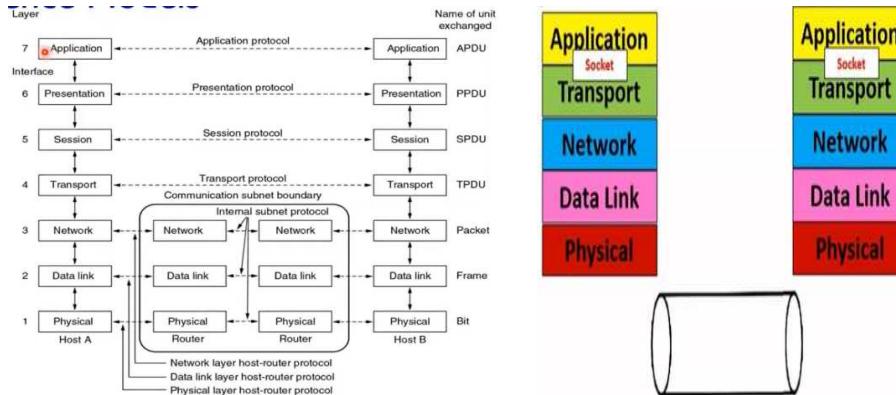


Figure 19.6 The OSI protocol stack.

TCP/IP Layers

- The application layer includes the protocols used by most applications for providing user services or exchanging application data over the network connections established by the lower level protocols
- Service Model is defined by the developer of the application itself, e.g. web client, chatting, online game, video streaming,etc.
- The transport layer establishes process-to-process connectivity, meaning it provides end-to-end services that are independent of the structure of user data and the logistics of exchanging information for any particular specific purpose.
 - Service Model:
 - Segmentation of data into standard size segments (in TCP).
 - Multiplexing and De-Multiplexing of apps running on the same host (**Port Number**)
 - Reliable Communication (in TCP)
- The Network layer has the responsibility of sending packets across potentially multiple networks. Internetworking requires sending data from the source network to the destination network on the best possible path. This process is called routing..
 - Service Model:
 - Host addressing and identification: This is accomplished with a hierarchical **IP addressing** system.
 - Packet routing: This is the basic task of sending packets of data (datagrams) from source to destination by forwarding them to the next network router closer to the final destination.
- The link layer is used to move packets between the Internet layer interfaces of two different hosts on the same link (hop-to-hop). The processes of transmitting and receiving packets on a given link can be controlled both in the software **device driver** for the network card, as well as on **firmware** or specialized chipsets.
 - Service Model:
 - Each end point is addressed using **MAC (Physical Address)**.
 - Frame forwarding and error checking (correction): By adding a check sum (CRC)
 - Synchronization of Communication.
 - Accepting or ignoring frames and passing them to upper layer
- The Physical layer is where actual data communication is carried out through ADC and DAC
- Service Model:
 - Encoding
 - Nodulation/ De-Modulation
 - Signaling
 - Collision Assisting
 - Channel management

Introduction to Network _ Application Layer _ Basic Concept

AppLayer Address

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
- more shortly...

Key Requirements

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

security

- encryption, data integrity, ...

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

AppLayer Protocol

App-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
- 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,

Q: why bother? Why is there a UDP?

Introduction to Network _ Application Layer _ 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



uses **TCP**:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

- protocols that maintain "state" are complex!
- past history (state) must be maintained
 - if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP Connections

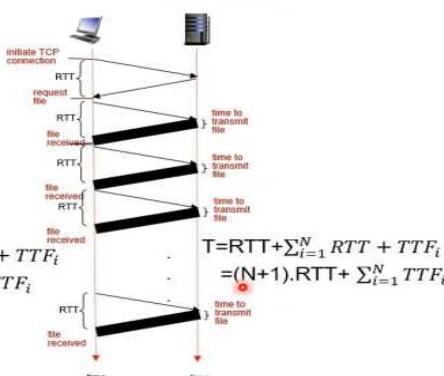
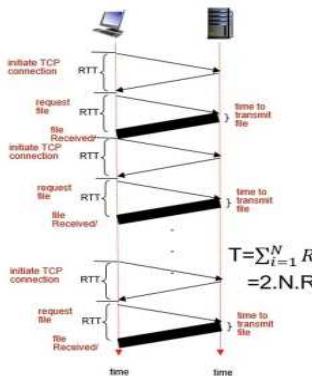
non-persistent HTTP

- at most one object sent over TCP connection
 - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

Persistent vs Non-persistent HTTP



HTTP Method Types

Web and HTTP

First, a review...

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

www.someschool.edu/someDept/pic.gif
host name path name

Method types

HTTP/1.0:

- GET
- POST
- HEAD
- asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
 - uploads file in entity body to path specified in URL field
- DELETE
 - deletes file specified in the URL field

HTTP request message

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

carriage return character
line-feed character
request line (GET, POST, HEAD commands)
header lines
entity body

```
GET /index.html HTTP/1.1\r\n
Host: www.net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.5,*/*;q=0.1\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,rue\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
```

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

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**

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: Tue, 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 ...
```

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

Introduction to Network_ Application Layer _ Cookies and Web Caching

Cookies

User-server state: cookies

many Web sites use cookies
four components:

- 1) cookie header line of HTTP response message
 - 2) cookie header line in next HTTP request message
 - 3) cookie file kept on user's host, managed by user's browser
 - 4) back-end database at Web site
- example:**
- Susan always access Internet from PC
 - visits specific e-commerce site for first time
 - when initial HTTP requests arrives at site, site creates:
 - unique ID
 - entry in backend database for ID

Cookies: keeping "state" (cont.)



what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

Web Caching

goal: satisfy client request without involving origin server

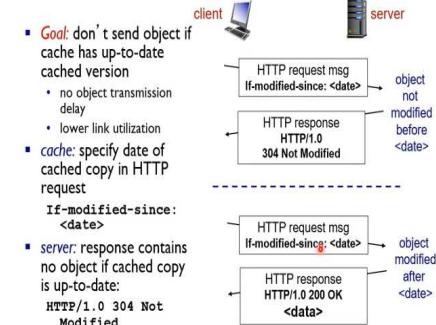
- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
 - object in cache: cache returns object
 - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
 - server for original requesting client
 - client to origin server
 - typically cache is installed by ISP (university, company, residential ISP)
- why Web caching?**
- reduce response time for client request
 - reduce traffic on an institution's access link
 - Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

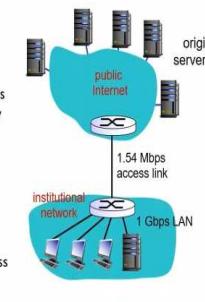
Conditional GET



Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15 Requests/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps



consequences:

- Total Lan Traffic = 1500 k bits/sec
- LAN utilization: 0.15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
 $= 2 \text{ sec} + \text{msec} + \mu\text{sec}$

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
 - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
 - 60% of requests use access link
 - data rate to browsers over access link = $0.6 \cdot 1.50 \text{ Mbps} = 0.9 \text{ Mbps}$
 - utilization = $0.9 / 1.54 = .58$
- total delay
 - $= 0.6 \cdot (\text{delay from origin servers}) + 0.4 \cdot (\text{delay when satisfied at cache})$
 - $= 0.6 \cdot (2.0) + 0.4 \cdot (\sim\text{msec}) = \sim 1.2 \text{ secs}$
 - less than with 154 Mbps link (and cheaper too!)

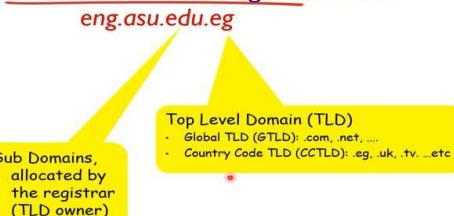
Introduction to Network _ Application Layer _ DNS

DNS : Domain Name System

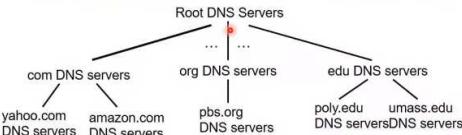
DNS: domain name system

- 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
- Q: how to map between IP address and name, and vice versa ?

DNS: services, naming convention



DNS: a distributed, hierarchical database



- client wants IP for www.amazon.com; 1st approximation:
- client queries root server to find com DNS server
 - client queries .com DNS server to get amazon.com DNS server
 - client queries amazon.com DNS server to get IP address for www.amazon.com

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
 - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
- acts as proxy, forwards query into hierarchy

DNS: services, structure

DNS services

- distributed database implemented in hierarchy of many name servers
- application-layer protocol: hosts, name servers communicate to resolve names (address/name translation)
 - note: core Internet function, implemented as application-layer protocol
 - complexity at network's "edge"

why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: .uk, .fr, .ca, .jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server

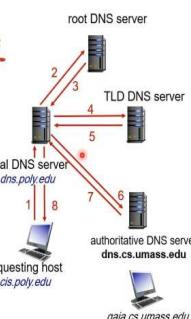


DNS name resolution example

host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

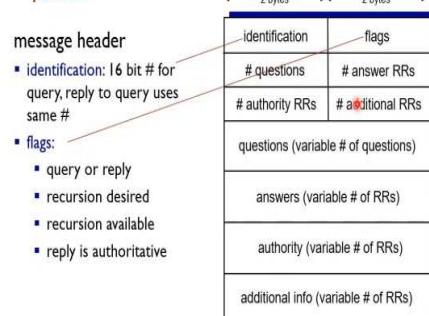
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



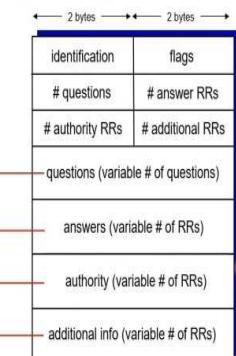
DNS Protocol and Records

DNS protocol, messages

- query and reply messages, both with same message format



DNS protocol, messages



DNS: caching, updating records

- once (any) name server learns mapping, it caches mapping
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
 - thus root name servers not often visited
- cached entries may be out-of-date (best effort name-to-address translation!)
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
 - RFC 2136

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkkutopia.com at DNS registrar (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts two RRs into .com TLD server: (networkkutopia.com, dns1.networkkutopia.com, NS) (dns1.networkkutopia.com, 212.212.212.212, A)
- create authoritative server type A record for www.networkkutopia.com; type MX record for networkkutopia.com

type=A

type=CNAME

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

type=NS

type=MX

- value is name of mailserver associated with name

Attacking DNS

DDoS attacks

- bombard root servers with traffic
 - not successful to date
 - traffic filtering
 - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
 - potentially more dangerous
 - send queries with spoofed source address: target IP requires amplification

Introduction to Network _ Application Layer _ Email

Cookies

Electronic mail

Email Address:

- user@host
- For example ayman.bahaa@mieugypt.edu.eg
- mail server per host, so each host is served by an application running on a computer (mail exchange server , MX)
- Each user has an allocated space on the server to keep his emails
- Each email message is a text only file, all other non text contents are encoded into text using MIME encoding

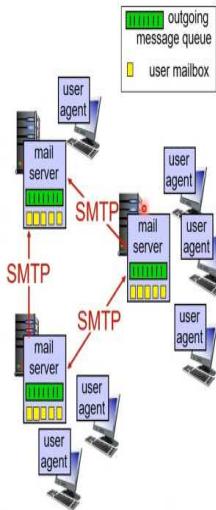
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g. Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



SMTP & POP3

Mail access protocols



- **SMTP:** delivery/storage to receiver's server
- mail access protocol: retrieval from server
 - **POP:** Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP:** Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
- **HTTP:** gmail, Hotmail, Yahoo! Mail, etc.

Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- command/response interaction (like HTTP)
 - commands: ASCII text
 - response: status code and phrase
- messages must be in 7-bit ASCII

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF, CRLF to determine end of message
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

POP3 protocol

authorization phase

client commands:

- user: declare username
 - pass: password
- server responses
- +OK
 - -ERR

transaction phase, client:

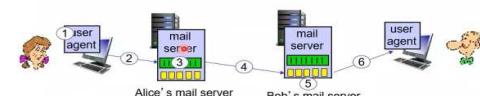
- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```

S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S:
C: retr 1
S: <message 1 contents>
C: dele 1
C: retr 2
S: <message 1 contents>
C: dele 2
C: quit
S: +OK POP3 server signing off
  
```

Scenario: Alice sends message to Bob

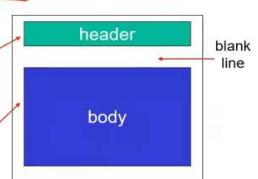
- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



Mail message format

SMTP: protocol for exchanging email messages
RFC 822: standard for text message format:

- header lines, e.g.,
 - To:
 - From:
 - Subject:
- different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
 - ASCII characters only



POP3 (more) and IMAP

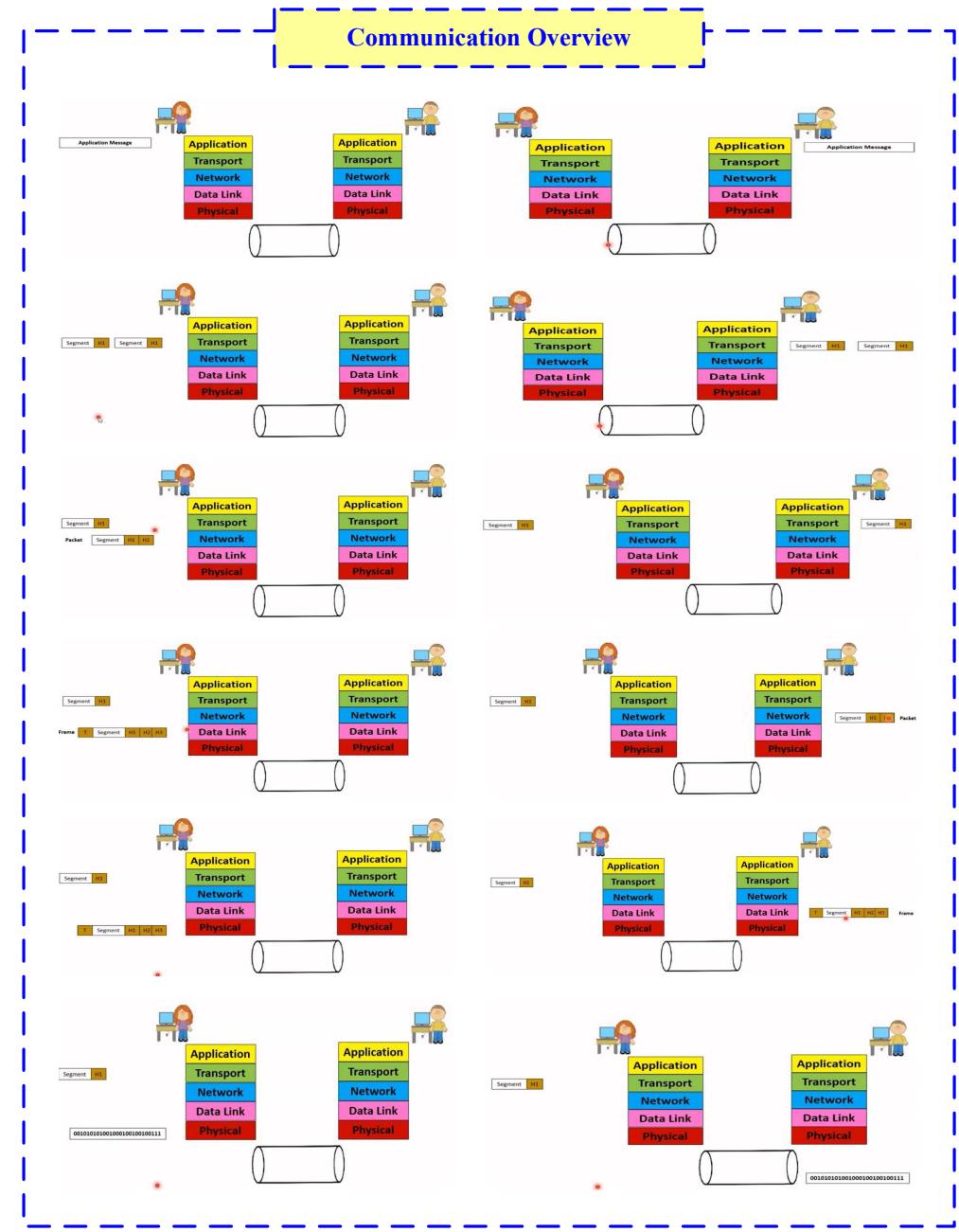
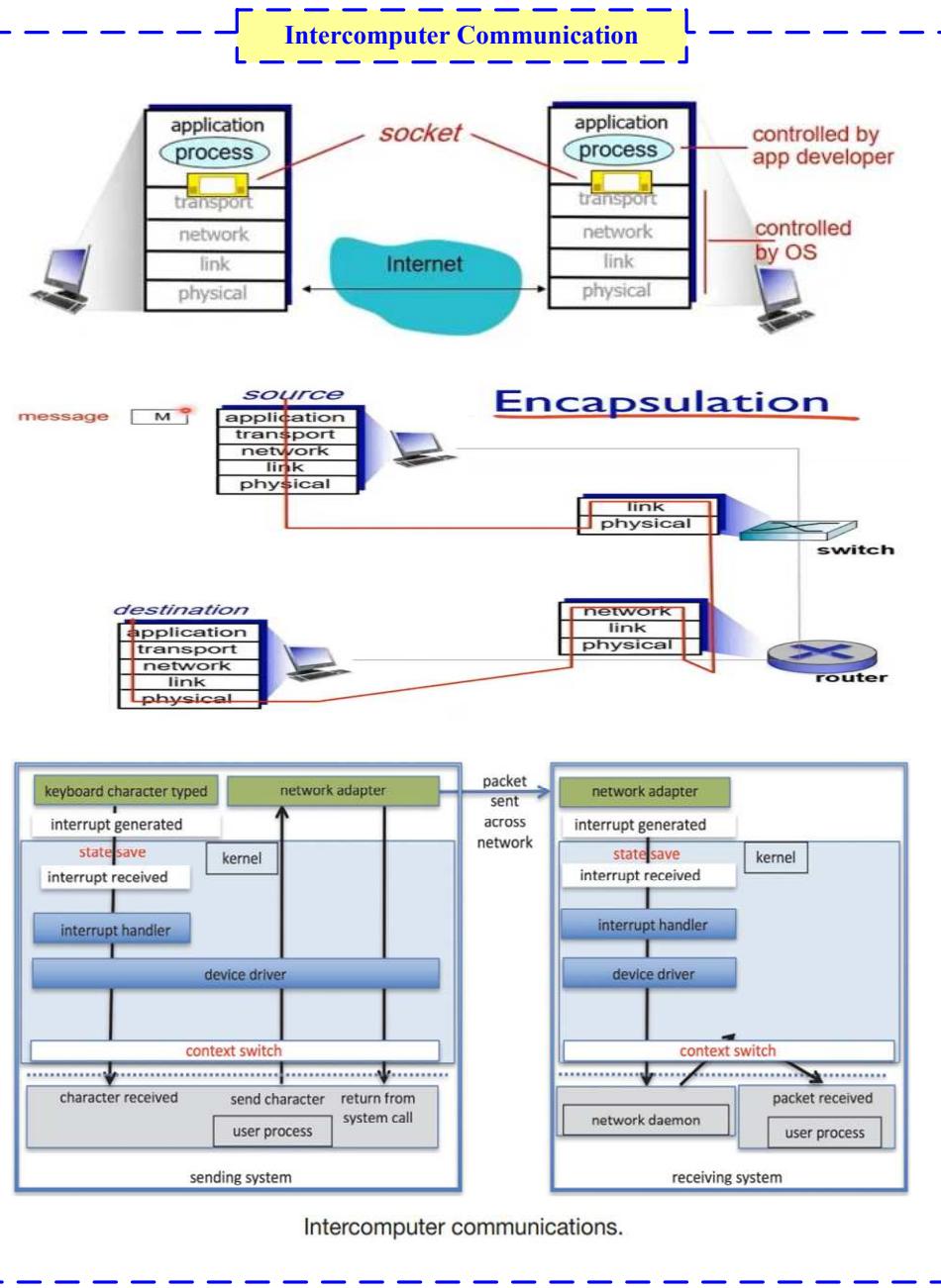
more about POP3

- previous example uses POP3 "download and delete" mode
 - Bob cannot re-read email if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

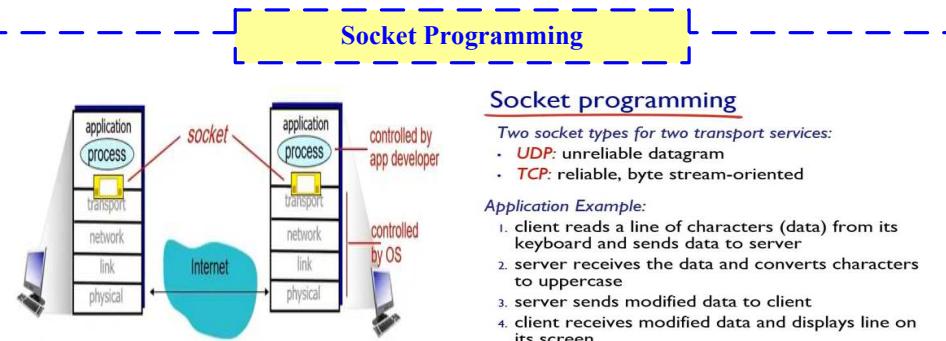
IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
 - names of folders and mappings between message IDs and folder name

Introduction to Network _ AppLayer_IPC and Sockets 1

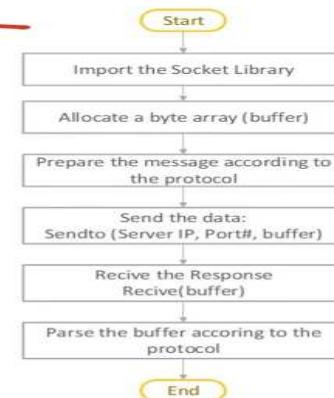


Introduction to Network _ AppLayer_IPC and Sockets 2

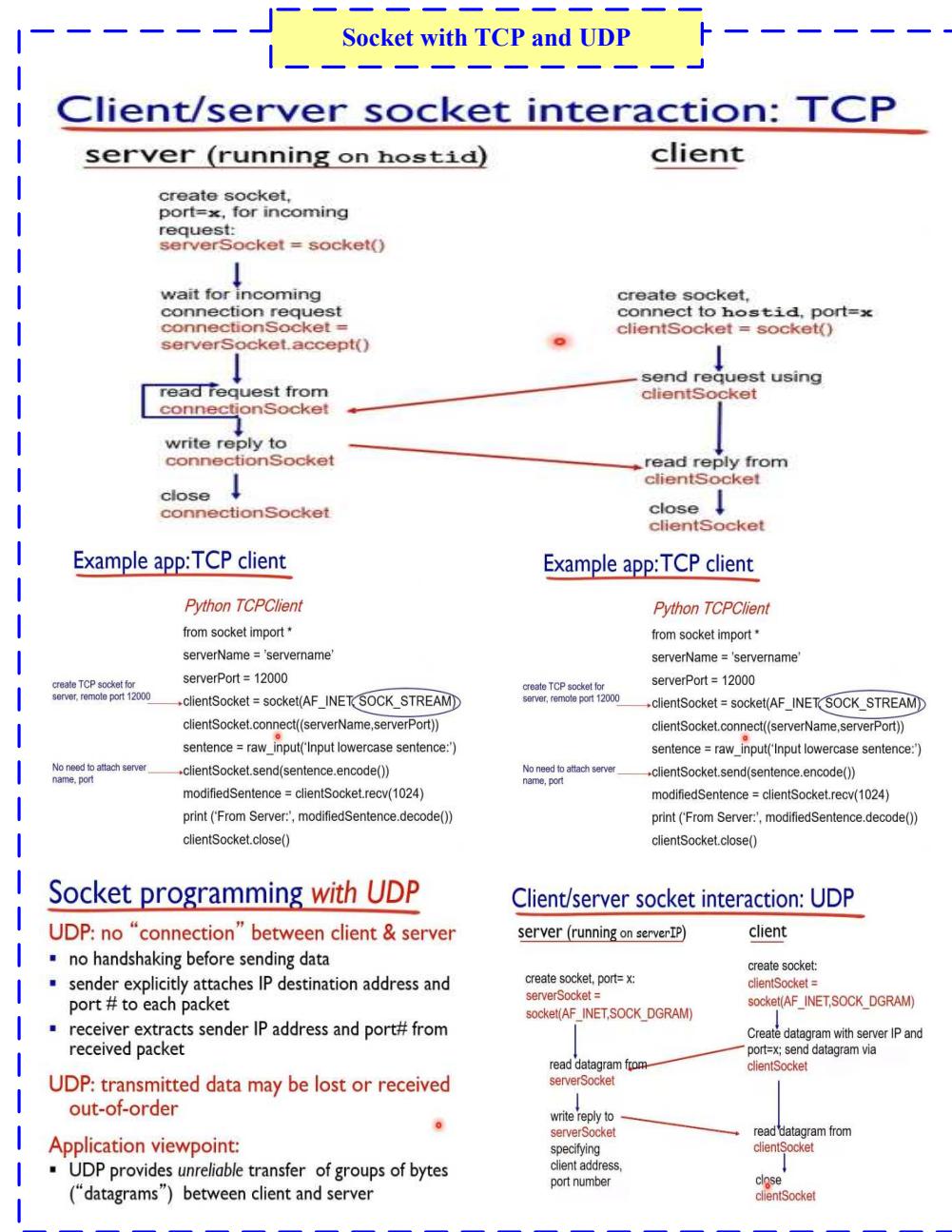
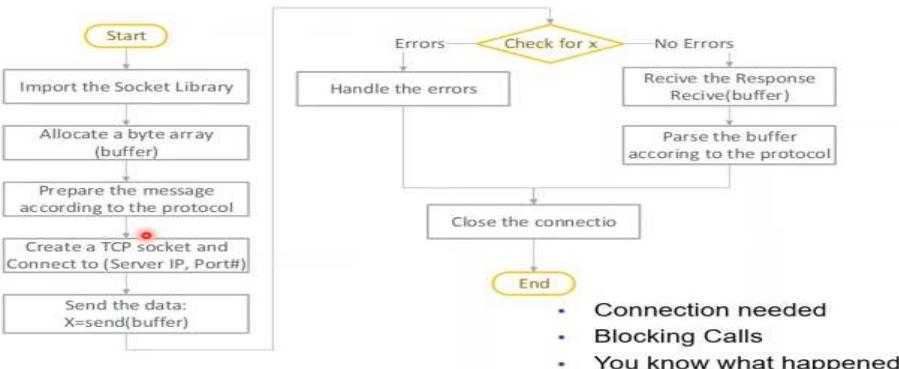


UDP Architecture

- No feed back from the call
- No connection needed
- Non Blocking Calls



TCP Architecture



Introduction to Network _ Transport Layer_ Basic Concept

Transport Protocol

Transport vs. network layer

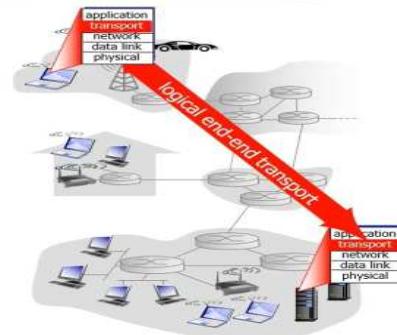
- network layer:** logical communication between hosts
- transport layer:** logical communication between processes
 - relies on, enhances, network layer services

household analogy:

- 12 kids in Ann's house sending letters to 12 kids in Bill's house:
- hosts = houses
 - processes = kids
 - app messages = letters in envelopes
 - transport protocol = Ann and Bill who demux to in-house siblings
 - network-layer protocol = postal service

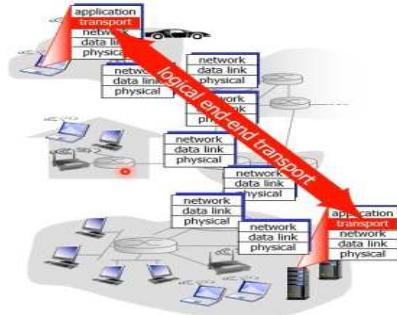
Transport services and protocols

- 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

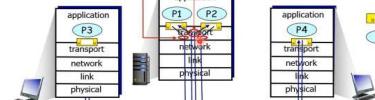
- reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- unreliable, unordered delivery: UDP
 - no-frills extension of "best-effort" IP
- services not available:
 - delay guarantees
 - bandwidth guarantees



Multiplexing and Demultiplexing

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



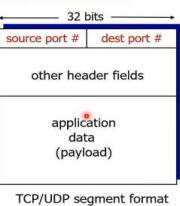
Connectionless demultiplexing

- recall: created socket has host-local port #:
DatagramSocket mySocket1 = new DatagramSocket(1234);
- recall: when creating datagram to send into UDP socket, must specify
 - destination IP address
 - destination port #
- when host receives UDP segment:
 - checks destination port # in segment
 - directs UDP segment to socket with that port #

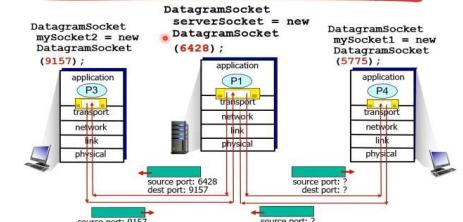
IP datagrams with **same dest_port #**, but different source IP addresses and/or source port numbers will be directed to **same socket** at dest

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

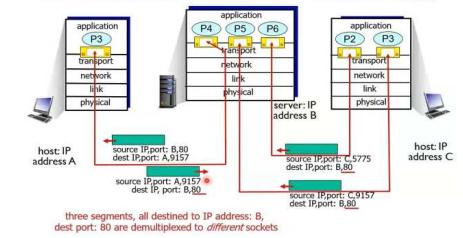


Connectionless demux: example

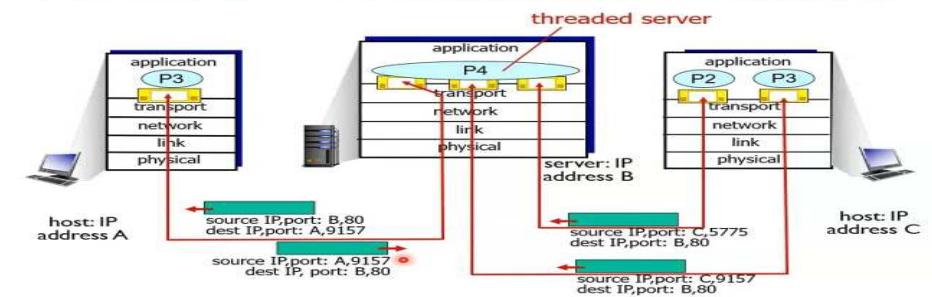


Connection-oriented demux

- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- 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
 - non-persistent HTTP will have different socket for each request



Connection-oriented demux: example



Introduction to Network _ Transport Layer_ TCP 1

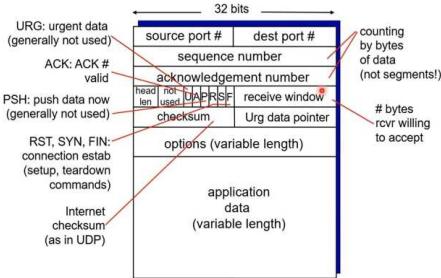
TCP Segment Structure

TCP: Overview

- point-to-point:**
 - one sender, one receiver
- reliable, in-order byte stream:**
 - 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) init's sender, receiver state before data exchange
- flow controlled:**
 - sender will not overwhelm receiver

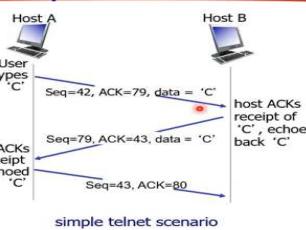
TCP segment structure



TCP seq. numbers, ACKs

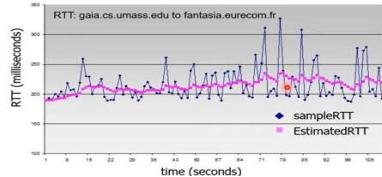
- sequence numbers:**
- byte stream “number” of first byte in segment’s data
- acknowledgements:**
- seq # of next byte expected from other side
 - cumulative ACK
- Q:** how receiver handles out-of-order segments
- A: TCP spec doesn’t say, - up to implementor
- Diagram illustrating sequence numbers and acknowledgements. It shows an outgoing segment from sender with fields: source port #, dest port #, sequence number, acknowledgement number, header length, options, checksum, and urg pointer. The sequence number is highlighted. The receiver’s sequence number space is shown with a grid of bits. An incoming segment from the receiver is shown with fields: source port #, dest port #, sequence number, acknowledgement number, header length, options, checksum, and urg pointer. The acknowledgement number is highlighted.

TCP seq. numbers, ACKs



TCP round trip time, timeout

- $\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$
- exponential weighted moving average
 - influence of past sample decreases exponentially fast
 - typical value: $\alpha = 0.125$



TCP round trip time, timeout

- Q:** how to set TCP timeout value?
- longer than RTT
 - but RTT varies
 - too short: premature timeout, unnecessary retransmissions
 - too long: slow reaction to segment loss
- Q:** how to estimate RTT?
- SampleRTT:** measured time from segment transmission until ACK receipt
 - ignore retransmissions
 - SampleRTT will vary, want estimated RTT “smoother”**
 - average several recent measurements, not just current SampleRTT

TCP round trip time, timeout

- timeout interval:** EstimatedRTT plus “safety margin”
- large variation in EstimatedRTT \rightarrow larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:**
- $$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$
- (typically, $\beta = 0.25$)
- TimeoutInterval** = EstimatedRTT + 4 * DevRTT
- estimated RTT “safety margin”

TCP Reliable Data Transfer

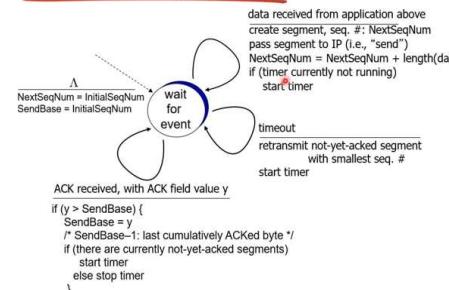
TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
- pipelined segments
- cumulative acks
- single retransmission timer
- retransmissions triggered by:
 - timeout events
 - duplicate acks

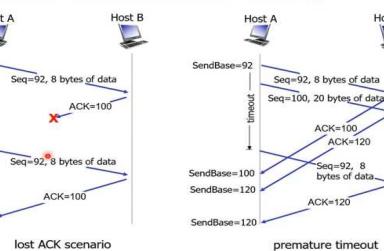
let’s initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

TCP sender (simplified)



TCP: retransmission scenarios



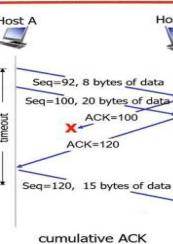
TCP fast retransmit

- time-out period often relatively long:**
- long delay before resending lost packet
- detect lost segments via duplicate ACKs:**
- sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs
- TCP fast retransmit**
- if sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq #
- likely that unacked segment lost, so don’t wait for timeout

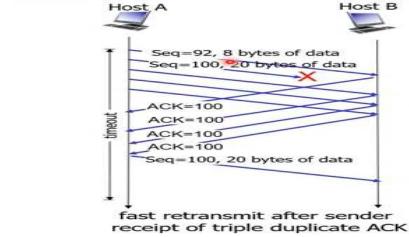
TCP ACK generation [RFC 1122, RFC 2581]

event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq #. Gap detected	immediately send duplicate ACK , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP: retransmission scenarios

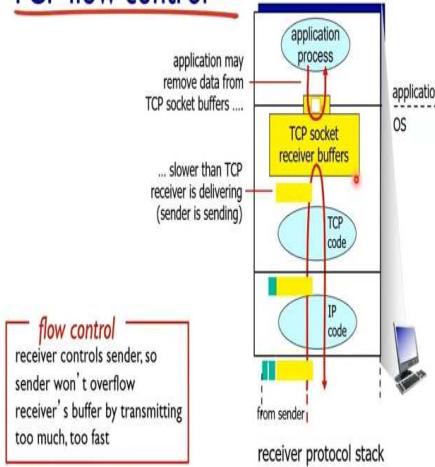


TCP fast retransmit



Introduction to Network _ Transport Layer_ TCP 2

TCP flow control



TCP flow control

TCP flow control

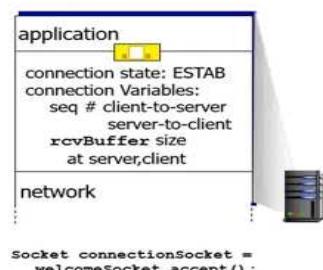
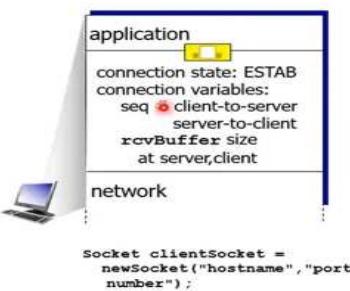
- receiver "advertises" free buffer space by including rwnd value in TCP header of receiver-to-sender segments
 - RcvBuffer size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust RcvBuffer
- sender limits amount of unacked ("in-flight") data to receiver's rwnd value
- guarantees receive buffer will not overflow

to application process
RcvBuffer
free buffer space
TCP segment payloads
receiver-side buffering

Connection Management

Connection Management

- before exchanging data, sender/receiver "handshake":
- agree to establish connection (each knowing the other willing to establish connection)
 - agree on connection parameters



TCP Reliable Data Transfer

Agreeing to establish a connection

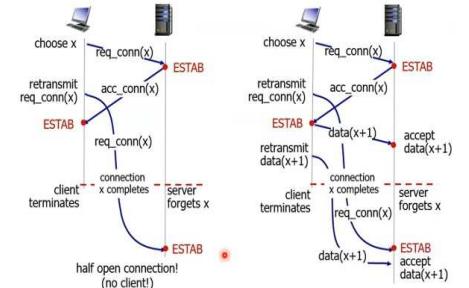
2-way handshake:



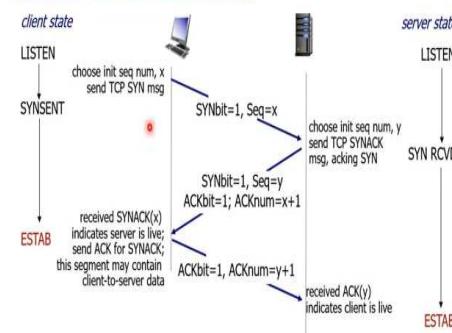
- Q: will 2-way handshake always work in network?
 - variable delays
 - retransmitted messages (e.g. req_conn(x)) due to message loss
 - message reordering
 - can't "see" other side

Agreeing to establish a connection

2-way handshake failure scenarios:

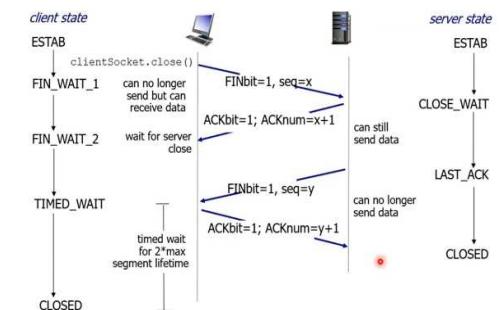


TCP 3-way handshake: FSM



TCP: closing a connection

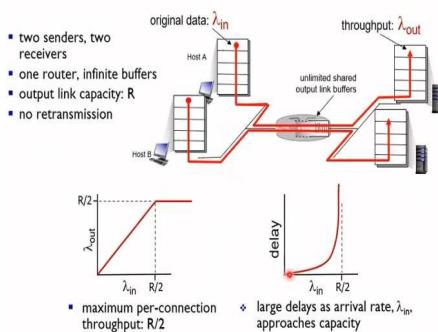
- client, server each close their side of connection
 - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
 - on receiving FIN, ACK can be combined with own FIN
 - simultaneous FIN exchanges can be handled



Introduction to Network _ Transport Layer_ TCP 3

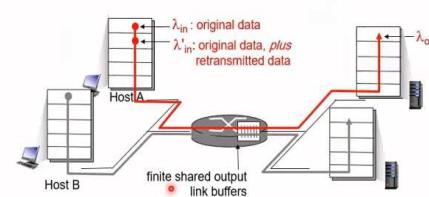
Principle of Congestion

Causes/costs of congestion: scenario 1



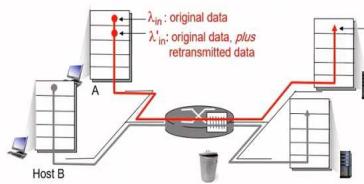
Causes/costs of congestion: scenario 2

- one router, **finite** buffers
- sender retransmission of timed-out packet
 - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
 - transport-layer input includes retransmissions: $\lambda_{in}' \geq \lambda_{in}$



Causes/costs of congestion: scenario 2

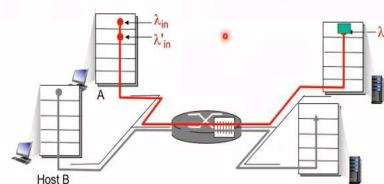
- Idealization:** known loss
packets can be lost, dropped at router due to full buffers
- sender only resends if packet known to be lost



Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending two copies, both of which are delivered

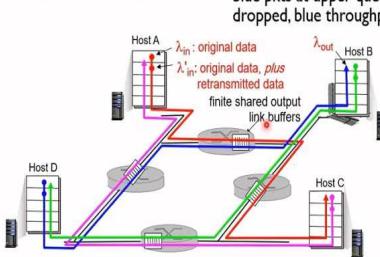


Causes/costs of congestion: scenario 3

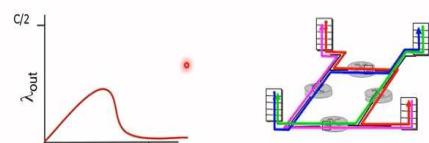
- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ_{in}' increase?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput $\rightarrow 0$



Causes/costs of congestion: scenario 3



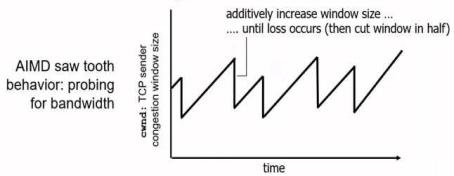
another “cost” of congestion:

- when packet dropped, any “upstream” transmission capacity used for that packet was wasted!

TCP Congestion Control

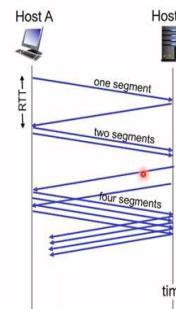
TCP congestion control: additive increase multiplicative decrease

- **approach:** sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
- **additive increase:** increase **cwnd** by 1 MSS every RTT until loss detected
 - **multiplicative decrease:** cut **cwnd** in half after loss



TCP Slow Start

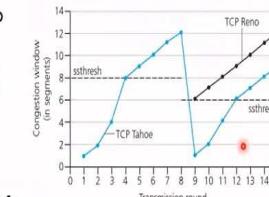
- when connection begins, increase rate exponentially until first loss event:
 - initially **cwnd** = 1 MSS
 - double **cwnd** every RTT
 - done by incrementing **cwnd** for every ACK received
- **summary:** initial rate is slow but ramps up exponentially fast



TCP: switching from slow start to CA

- Q: when should the exponential increase switch to linear?

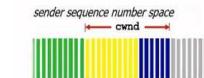
- A: when **cwnd** gets to 1/2 of its value before timeout.



Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

TCP Congestion Control: details



- TCP sending rate:**
- **roughly:** send **cwnd** bytes, wait RTT for ACKS, then send more bytes
- rate $\approx \frac{\text{cwnd}}{\text{RTT}}$ bytes/sec

- sender limits transmission:

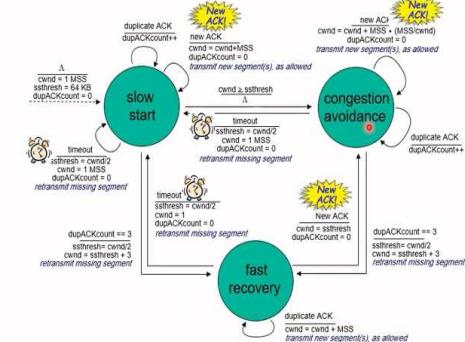
$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

TCP: detecting, reacting to loss

- loss indicated by timeout:
 - **cwnd** set to 1 MSS;
 - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
 - dup ACKs indicate network capable of delivering some segments
 - **cwnd** is cut in half window then grows linearly
- TCP Tahoe always sets **cwnd** to 1 (timeout or 3 duplicate acks)

Summary: TCP Congestion Control



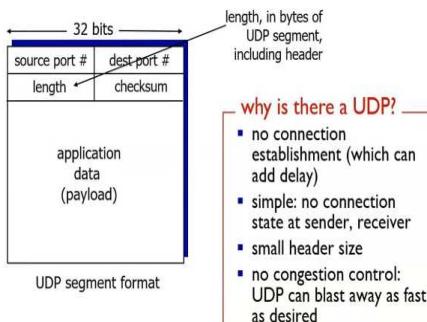
Introduction to Network _ Transport Layer_ UDP

UDP Protocol

UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet 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: segment header



Internet checksum: example

example: add two 16-bit integers

1	1	1	0	0	1	1	0	0	1	1	0	0	1	0
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
<hr/>														
wraparound	1	0	1	1	1	0	1	1	1	0	1	1	1	0
sum	1	0	1	1	1	0	1	1	0	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

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 nonetheless? More later

On the receiver

add the three 16-bit integers (including checksum)

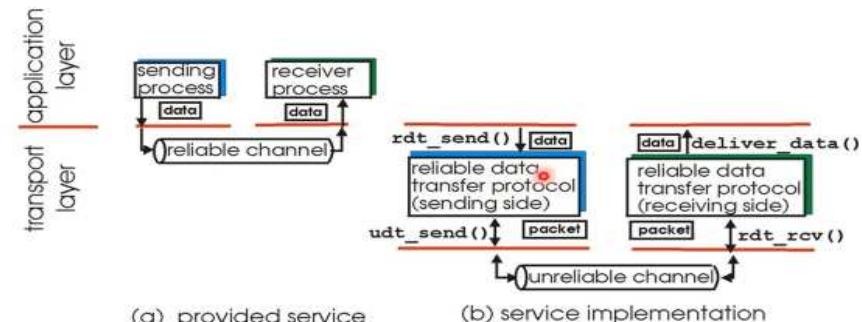
1	1	1	0	0	1	1	0	0	1	1	0	0	1	0
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
<hr/>														
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	1
①	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Note: The final result must be all ones or an error occurred

UDP Reliable Data Transfer

Principles of reliable data transfer

- important in application, transport, link layers
 - top-10 list of important networking topics!

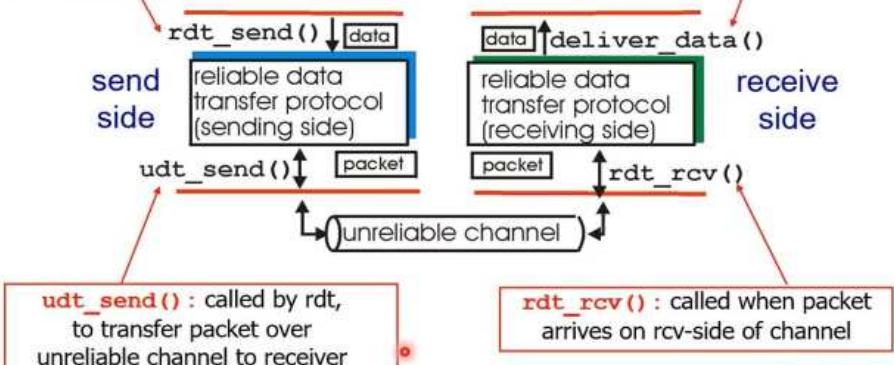


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer

rdt_send() : called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data() : called by rdt to deliver data to upper



Introduction to Network _ Network Layer_ Basic Concept

Network Layer

Transport vs. network layer

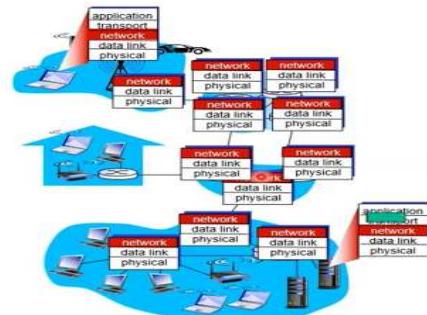
- **network layer:** logical communication between hosts
- **transport layer:** logical communication between processes
 - relies on, enhances, network layer services

household analogy:

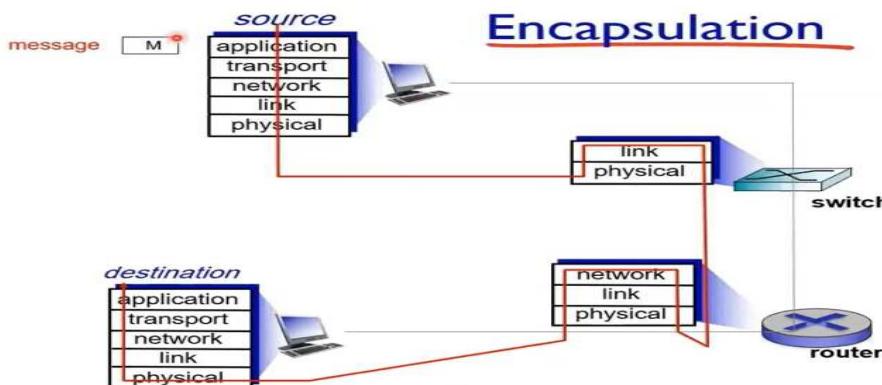
- 12 kids in Ann's house sending letters to 12 kids in Bill's house:
 - hosts = houses
 - processes = kids
 - app messages = letters in envelopes
 - transport protocol = Ann and Bill who demux to in-house siblings
 - network-layer protocol = postal service

Network layer

- ❖ transport segment from sending to receiving host
- ❖ on sending side encapsulates segments into datagrams
- ❖ on receiving side, delivers segments to transport layer
- ❖ network layer protocols in **every** host, router
- ❖ router examines header fields in all IP datagrams passing through it



Encapsulation



Routing and Forwarding

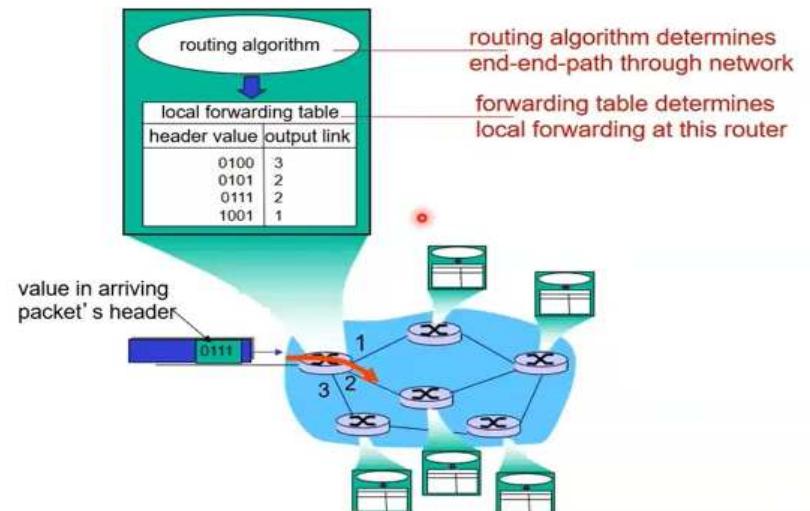
Two key network-layer functions

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

analogy:

- ❖ **routing:** process of planning trip from source to dest
- ❖ **forwarding:** process of getting through single interchange

Interplay between routing and forwarding



Introduction to Network _ Network Layer_ Diagram and VC

Network Layer

Datagram or VC network: why?

- | | |
|--|---|
| Internet (datagram) | ATM (VC) |
| <ul style="list-style-type: none"> data exchange among computers <ul style="list-style-type: none"> "elastic" service, no strict timing req. many link types <ul style="list-style-type: none"> different characteristics uniform service difficult "smart" end systems (computers) <ul style="list-style-type: none"> can adapt, perform control, error recovery simple inside network, complexity at "edge" | <ul style="list-style-type: none"> evolved from telephony human conversation: <ul style="list-style-type: none"> strict timing, reliability requirements need for guaranteed service "dumb" end systems <ul style="list-style-type: none"> telephones complexity inside network |

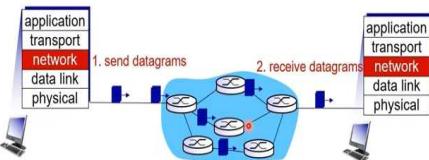
Connection-less connection service

- datagram network provides network-layer connectionless service
- virtual-circuit network provides network-layer connection service
- analogous to TCP/UDP connection-oriented / connectionless transport-layer services, but:
 - service: host-to-host
 - no choice: network provides one or the other
 - implementation: in network core

Diagram Network

Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
 - no network-level concept of "connection"
- packets forwarded using destination host address

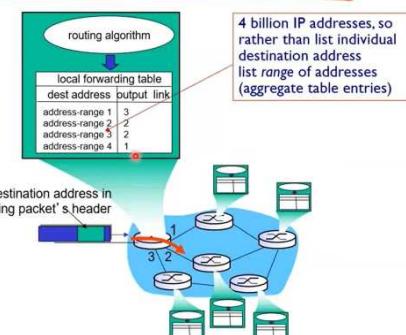


Datagram forwarding table

Destination 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

Q: but what happens if ranges don't divide up so nicely?

Datagram forwarding table



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 Address Range	Link interface
11001000 00010111 00010111 ****	0
11001000 00010111 00011000 ****	1
11001000 00010111 00011001 ****	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001 which interface?
DA: 11001000 00010111 00010100 10101010 which interface?

Virtual Circuit

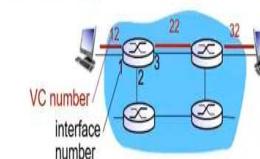
Virtual circuits

"source-to-dest path behaves much like telephone circuit"

- performance-wise
- network actions along source-to-dest path

- call setup, teardown for each call before data can flow
- each packet carries VC identifier (not destination host address)
- every router on source-dest path maintains "state" for each passing connection
- link, router resources (bandwidth, buffers) may be allocated to VC (dedicated resources = predictable service)

VC forwarding table



forwarding table in northwest router:

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
...

VC routers maintain connection state information!

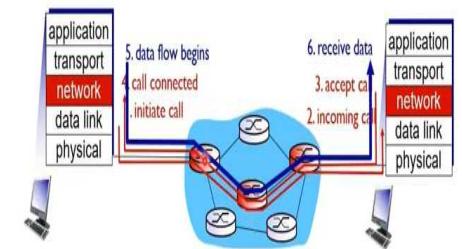
VC implementation

a VC consists of:

- path from source to destination
- VC numbers, one number for each link along path
- entries in forwarding tables in routers along path
- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
 - new VC number comes from forwarding table

Virtual circuits: signaling protocols

- used to setup, maintain, teardown VC
- used in ATM, frame-relay, X.25
- not used in today's Internet



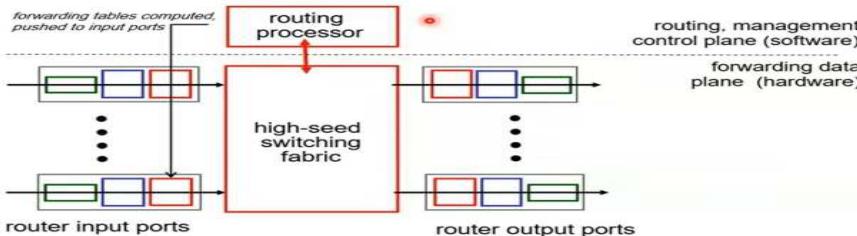
Introduction to Network _ Network Layer_ Routing and Forwarding

Inside Routing

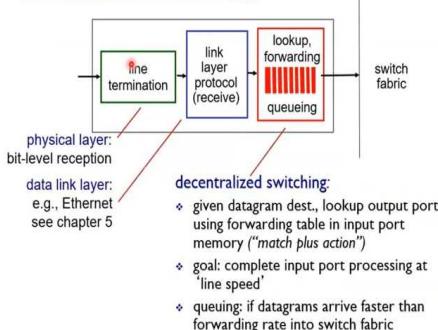
Router architecture overview

two key router functions:

- ❖ run routing algorithms/protocol (RIP, OSPF, BGP)
- ❖ forwarding datagrams from incoming to outgoing link

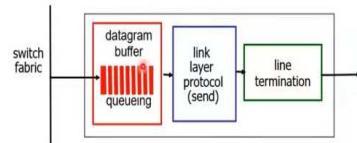


Input port functions



Output ports

This slide is HUGELY important!



- ❖ **buffering** required when datagrams arrive from fabric faster than the transmission rate
Datagram (packets) can be lost due to congestion, lack of buffers

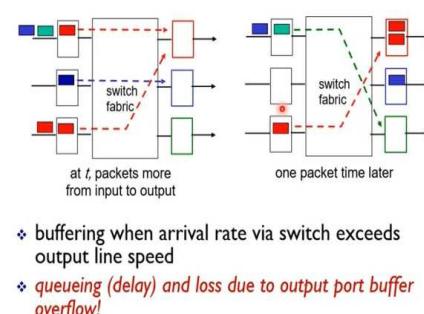
- ❖ **scheduling discipline** chooses among queued datagrams for transmission
Priority scheduling – who gets best performance, network neutrality

Input port queuing

- ❖ fabric slower than input ports combined -> queuing may occur at input queues
 - **queueing delay and loss due to input buffer overflow!**
- ❖ **Head-of-the-Line (HOL) blocking:** queued datagram at front of queue prevents others in queue from moving forward

The diagram shows two stages of a queue. In the first stage, multiple packets are in the queue. In the second stage, one packet has been processed, but the red packet at the head of the queue is still there, preventing the green packet from moving forward. This is labeled as 'one packet time later: green packet experiences HOL blocking'.

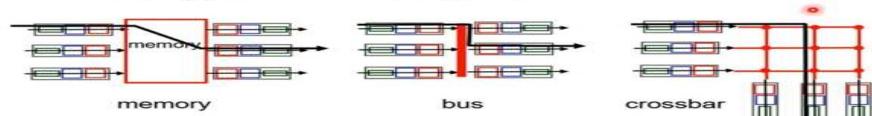
Output port queuing



Switching Fabrics

Switching fabrics

- ❖ transfer packet from input buffer to appropriate output buffer
- ❖ **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable
- ❖ **three types of switching fabrics**



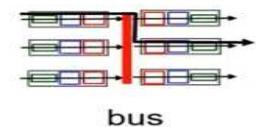
Switching via memory

- first generation routers:**
- ❖ traditional computers with switching under direct control of CPU
 - ❖ packet copied to system's memory
 - ❖ speed limited by memory bandwidth (2 bus crossings per datagram)



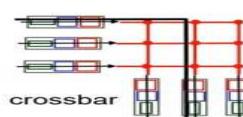
Switching via a bus

- ❖ datagram from input port memory to output port memory via a shared bus
- ❖ **bus contention:** switching speed limited by bus bandwidth
- ❖ 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



Switching via interconnection network

- ❖ overcome bus bandwidth limitations
- ❖ banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- ❖ advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- ❖ Cisco 12000: switches 60 Gbps through the interconnection network

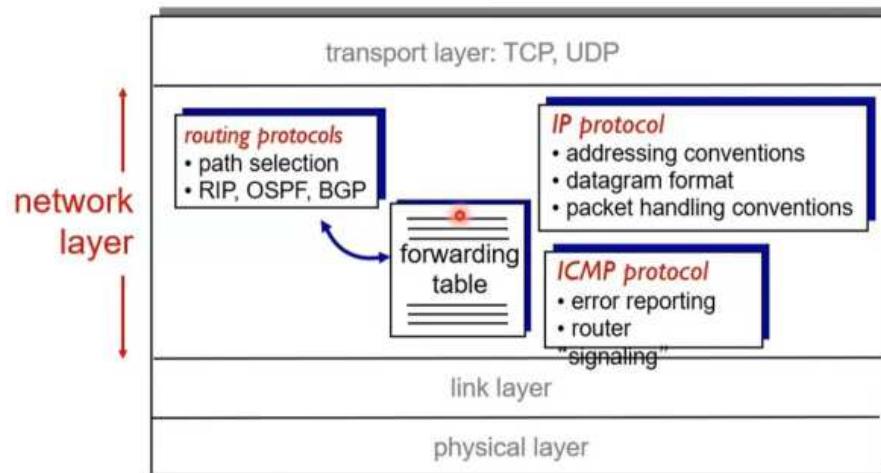


Introduction to Network _ Network Layer_ Internet Protocol Basic

IP Diagram Format

The Internet network layer

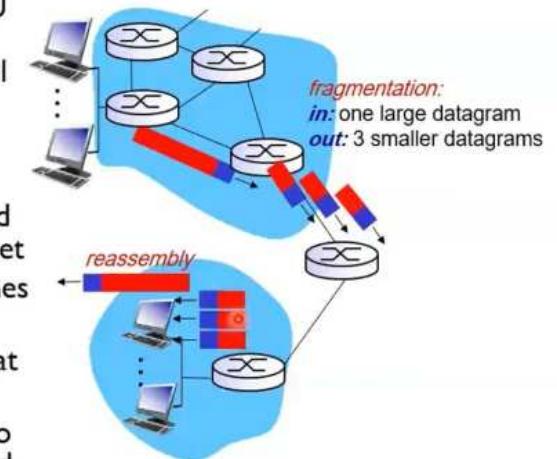
host, router network layer functions:



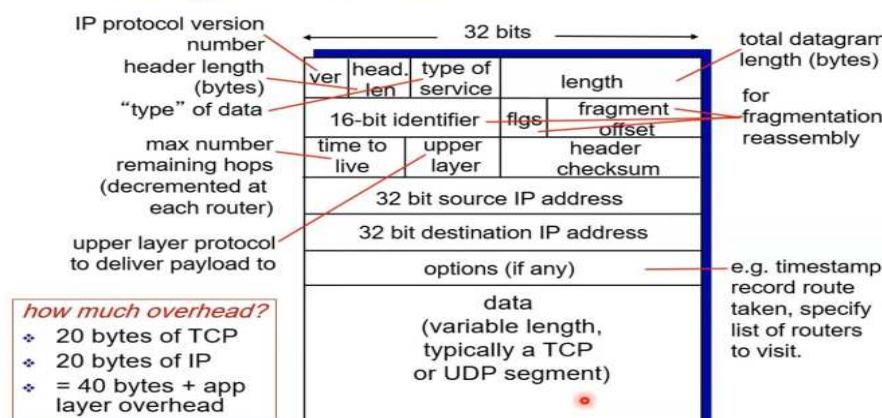
IP Fragment

IP fragmentation, reassembly

- ❖ network links have MTU (max.transfer size) - largest possible link-level frame
 - different link types, different MTUs
- ❖ 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 datagram format



IP fragmentation, reassembly

example:

- ❖ 4000 byte datagram
- ❖ MTU = 1500 bytes

length = 4000	ID = x	fragflag = 0	offset = 0
---------------	--------	--------------	------------

one large datagram becomes several smaller datagrams

1480 bytes in data field

$$\text{offset} = \frac{1480}{8}$$

length = 1500	ID = x	fragflag = 1	offset = 0
length = 1500	ID = x	fragflag = 1	offset = 185
length = 1040	ID = x	fragflag = 0	offset = 370

Introduction to Network _ Network Layer_ IP_DHCP

IP Diagram Format

DHCP: Dynamic Host Configuration Protocol

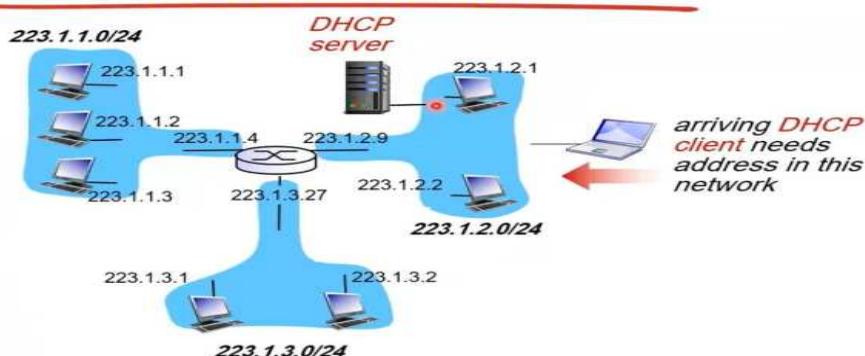
goal: allow host to *dynamically* obtain its IP address from network server when it joins *network*

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/“on”)
- support for mobile users who want to join network (more shortly)

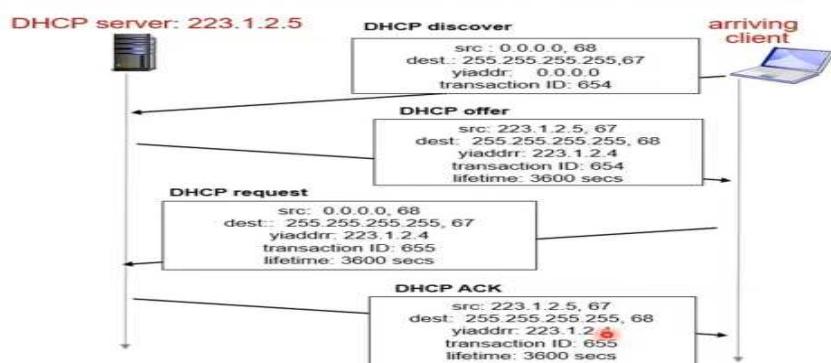
DHCP overview:

- host broadcasts “DHCP discover” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg

DHCP client-server scenario

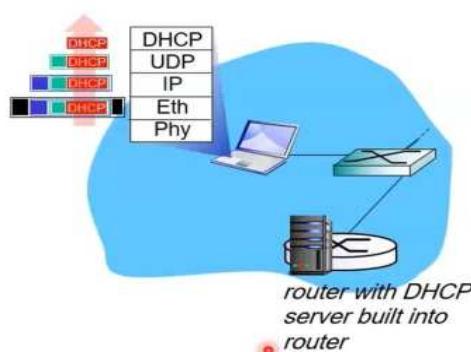
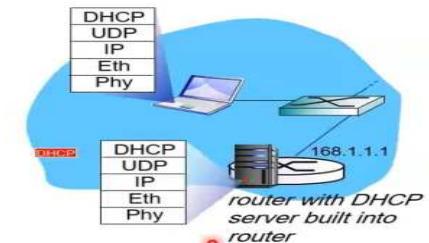


DHCP client-server scenario



DHCP

- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
- DCP server formulates DHCP ACK containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DSN server, IP address of its first-hop router



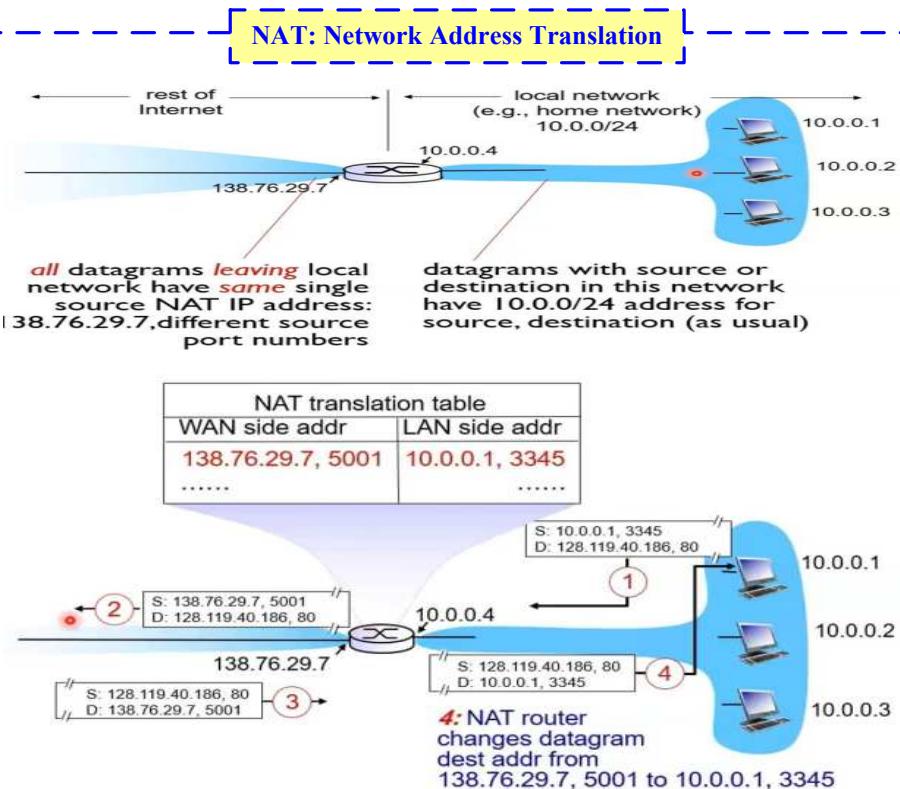
DHCP: Wireshark output (home LAN)

```

Message type: Boot Request (1)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Boot flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
New IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:68 (00:16:d3:23:68:68)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (l=53,l=1) DHCP Message Type = DHCP Request
Option: (l=61,C=1) Client Identifier = 00:16:d3:23:68:68
Option: (l=54,l=4) Server Identifier = 192.168.1.1
Option: (l=1,l=4) Subnet Mask = 255.255.255.0
Option: (l=1,l=1) Lease Time = 3600
Option: (l=6) Domain Name Server
Length: 12; Value: 445747E2445749F244574092;
IP Address: 68.87.71.226;
IP Address: 68.87.71.226;
IP Address: 68.87.64.146;
Option: (l=15,l=20) Domain Name = "hsd1.ma.comcast.net."

```

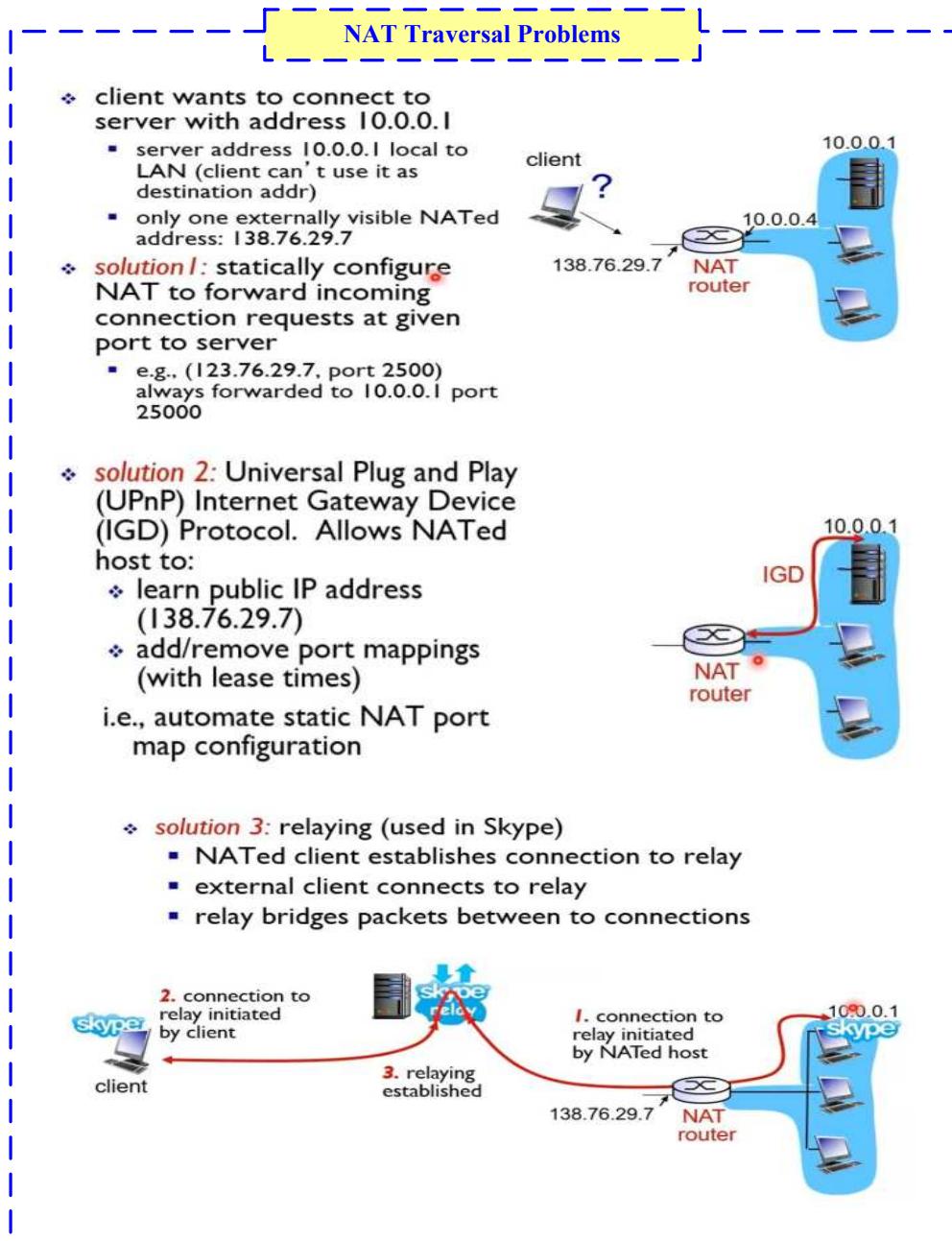
Introduction to Network _ Network Layer_ IP_NAT



- motivation:** local network uses just one IP address as far as outside world is concerned:
- range of addresses not needed from ISP: just one IP address for all devices
 - can change addresses of devices in local network without notifying outside world
 - can change ISP without changing addresses of devices in local network
 - devices inside local net not explicitly addressable, visible by outside world (a security plus)

implementation: NAT router must:

- **outgoing datograms:** replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
... remote clients/servers will respond using (NAT IP address, new port #) as destination addr
- remember (in **NAT translation table**) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- **incoming datograms:** replace (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table



Introduction to Network _ Network Layer_ IP_ICMP and IPv6

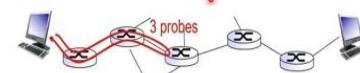
NAT: Network Address Translation

ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (used by ping)
 - network-layer "above" IP: ICMP msgs carried in IP datagrams
 - ICMP message: type, code plus first 8 bytes of IP datagram causing error
- | Type | Code | Description |
|------|------|---|
| 0 | 0 | echo reply (ping) |
| 3 | 0 | dest. network unreachable |
| 3 | 1 | dest host unreachable |
| 3 | 2 | dest protocol unreachable |
| 3 | 3 | dest port unreachable |
| 3 | 6 | dest network unknown |
| 3 | 7 | dest host unknown |
| 4 | 0 | source quench (congestion control - not used) |
| 8 | 0 | echo request (ping) |
| 9 | 0 | route advertisement |
| 10 | 0 | router discovery |
| 11 | 0 | TTL expired |
| 12 | 0 | bad IP header |

Traceroute and ICMP

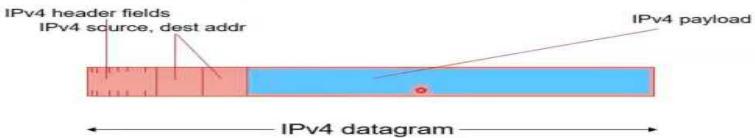
- source sends series of UDP segments to dest
 - first set has TTL=1
 - second set has TTL=2, etc.
 - unlikely port number
- when nth set of datagrams arrives to nth router:
 - router discards datagrams
 - and sends source ICMP messages (type 11, code 0)
 - ICMP messages includes name of router & IP address
- when ICMP messages arrives, source records RTTs
- stopping criteria:
 - UDP segment eventually arrives at destination host
 - destination returns ICMP "port unreachable" message (type 3, code 3)
 - source stops



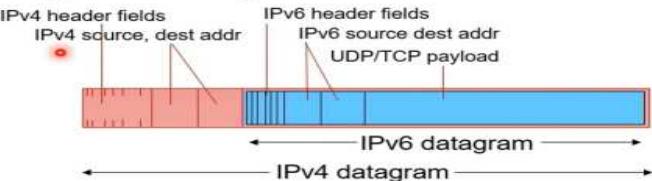
From IPv4 to IPv6

Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
 - no "flag days"
 - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling:** IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers



- not all routers can be upgraded simultaneously
 - no "flag days"
 - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling:** IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers



IPv6: motivation

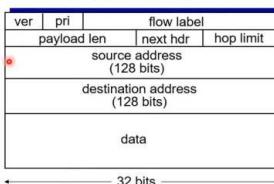
- initial motivation:** 32-bit address space soon to be completely allocated.
- additional motivation:**
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS

IPv6 datagram format:

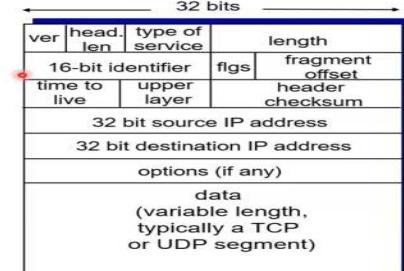
- fixed-length 40 byte header
- no fragmentation allowed

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



IPv4 datagram format

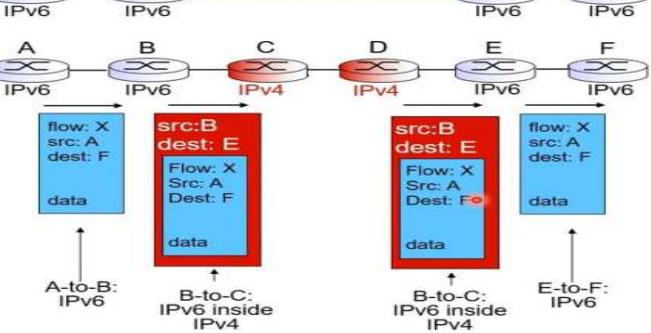


Tunneling

logical view:



physical view:



Introduction to Network _ DataLink Layer_Basic Concept

DataLink Layer

Link layer: introduction

terminology:

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



Link layer: context

- ❖ datagram transferred by different link protocols over different links:
 - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- ❖ each link protocol provides different services
 - e.g., may or may not provide rdt 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 used in frame headers to identify source, dest
 - different from IP address!
- ❖ **reliable delivery between adjacent nodes**
 - we learned how to do this already (chapter 3)!
 - seldom used on low bit-error link (fiber, some twisted pair)
 - wireless links: high error rates
 - Q: why both link-level and end-end reliability?

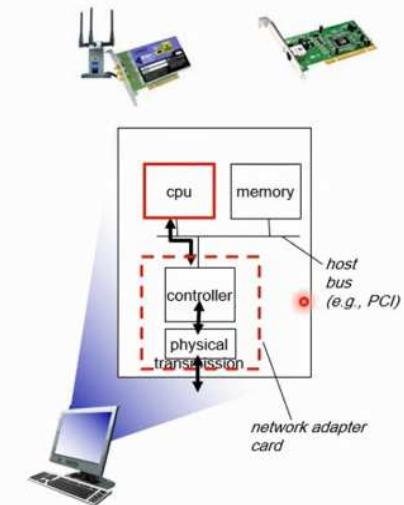
Link layer services (more)

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

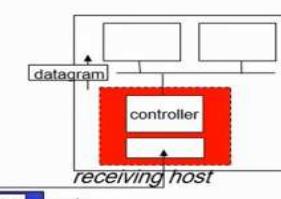
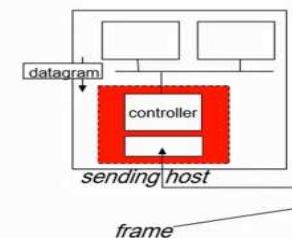
DataLink Implementation

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
- ❖ attaches into host’s system buses
- ❖ combination of hardware, software, firmware



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

Introduction to Network _ DataLink Layer_MAC and ARP

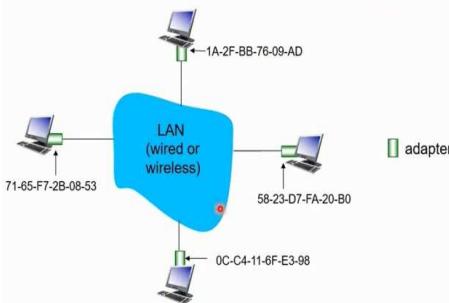
MAC Address and ARP

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 "locally" to get frame from one interface to another physically-connected interface (same network, in IP addressing sense)
 - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
 - e.g.: 1A-2F-BB-76-09-AD
- hexadecimal (base 16) notation
(each "number" represents 4 bits)

LAN addresses and ARP

each adapter on LAN has unique LAN address



ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?

ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:
- <IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol: same LAN

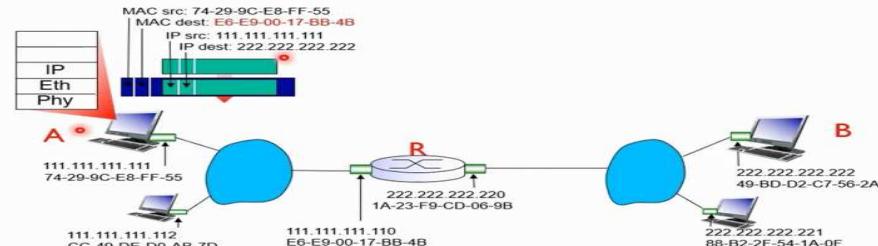
- A wants to send datagram to B
 - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
 - dest MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
 - frame sent to A's MAC address (unicast)

- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
 - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
 - nodes create their ARP tables without intervention from net administrator

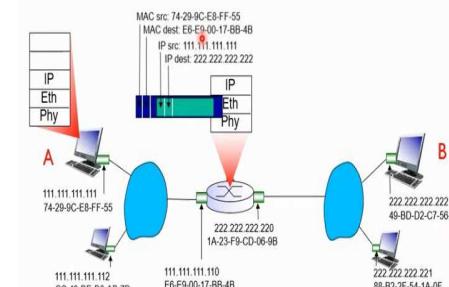
ARP to another LAN

Addressing: routing to another LAN

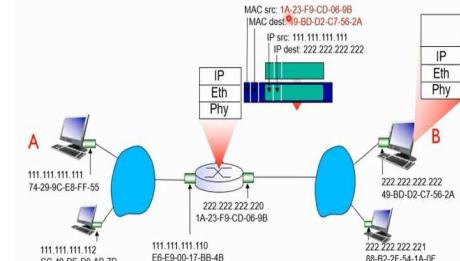
- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as dest, frame contains A-to-B IP datagram



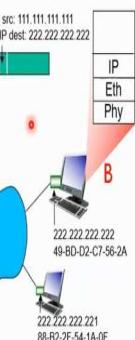
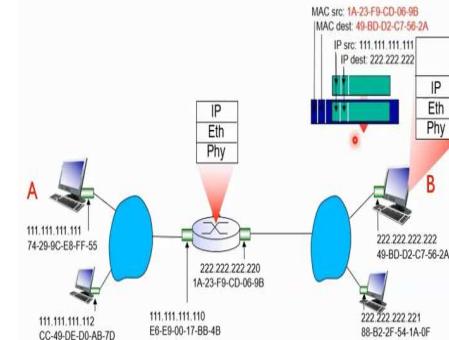
- frame sent from A to R



- 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



- 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



Introduction to Network _ DataLink Layer_Ethernet and Switch

Ethernet Frame Structure

Ethernet frame structure

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 receives frame with matching destination address, or with 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:** cyclic redundancy check at receiver
 - error detected: frame is dropped



Ethernet: unreliable, connectionless

- connectionless:** no handshaking between sending and receiving NICs
- unreliable:** receiving NIC doesn't send acks or nacks to sending NIC
 - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted CSMA/CD wth **binary backoff**

Ethernet switch

- link-layer device: takes an active role**
 - store, forward Ethernet frames
 - examine incoming frame's MAC address, **selectively forward** frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- transparent**
 - hosts are unaware of presence of switches
- plug-and-play, self-learning**
 - switches do not need to be configured

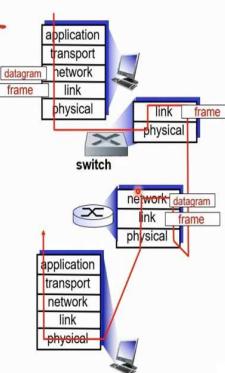
Switches vs. routers

both are store-and-forward:

- routers:** network-layer devices (examine network-layer headers)
- switches:** link-layer devices (examine link-layer headers)

both have forwarding tables:

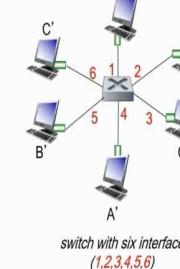
- routers:** compute tables using routing algorithms, IP addresses
- switches:** learn forwarding table using flooding, learning, MAC addresses



Switch

Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on each incoming link, but no collisions; full duplex
 - each link is its own collision domain
- switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions

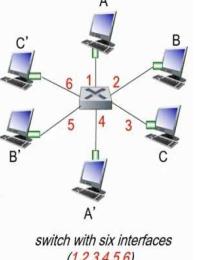


Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- A:** each switch has a **switch table**, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!

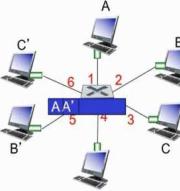


Q: how are entries created, maintained in switch table?

- something like a routing protocol?

Switch: self-learning

- switch **learns** which hosts can be reached through which interfaces
- when frame received, switch "learns" location of sender: incoming LAN segment
- records sender/location pair in switch table



MAC addr	interface	TTL
A	1	60

Switch table (initially empty)

Switch: frame filtering/forwarding

when frame received at switch:

- record incoming link, MAC address of sending host
- index switch table using MAC destination address
- if entry found for destination
 - then {
 - if destination on segment from which frame arrived then drop frame
 - else forward frame on interface indicated by entry
- else flood /* forward on all interfaces except arriving interface */

Self-learning, forwarding: example

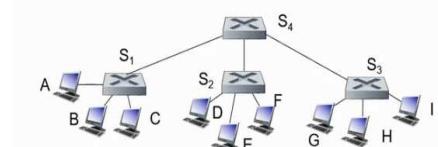
- frame destination, A', location unknown: **flood**

- destination A location known: **selectively send on just one link**

MAC addr	interface	TTL
A	1	60
A	4	60

switch table (initially empty)

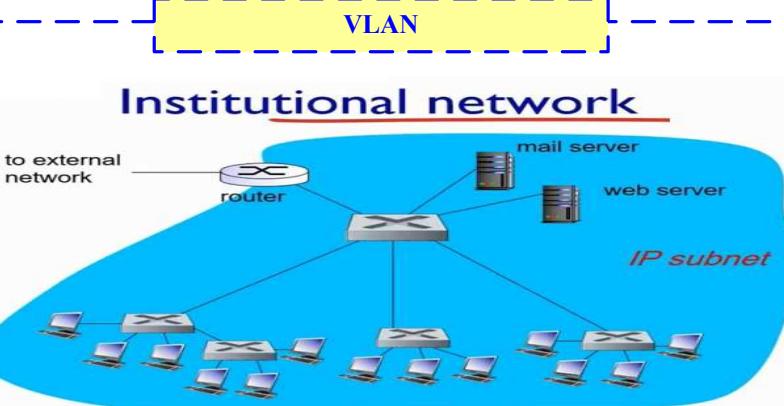
- switches can be connected together



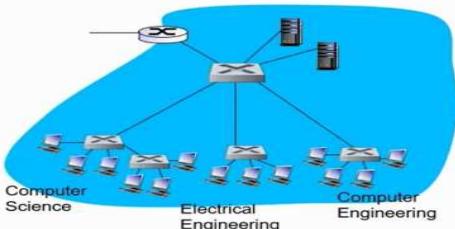
Q: sending from A to G - how does S₁ know to forward frame destined to F via S₄ and S₃?

- A:** self learning! (works exactly the same as in single-switch case!)

Introduction to Network _ DataLink Layer_VLAN



VLANs: motivation



consider:

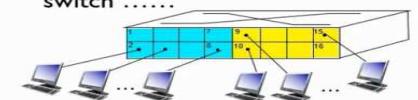
- ❖ CS user moves office to EE, but wants connect to CS switch?
- ❖ single broadcast domain:
 - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
 - security/privacy, efficiency issues

VLANs

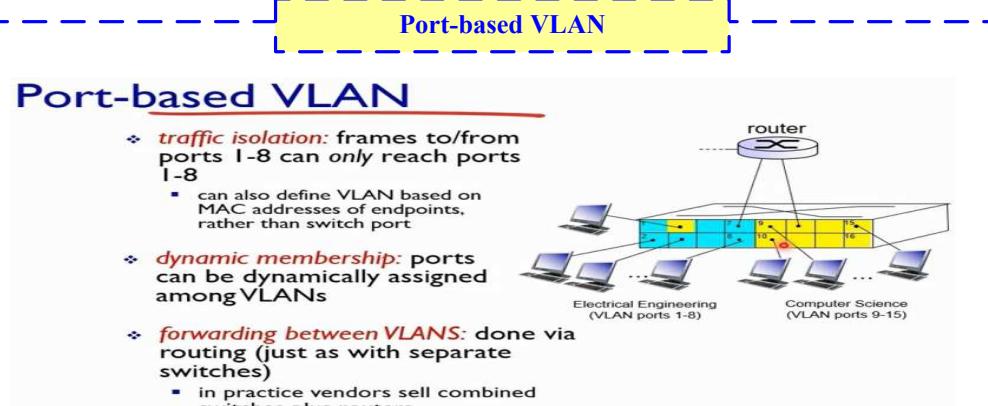
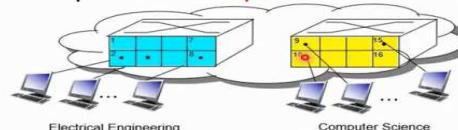
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANs over single physical LAN infrastructure.

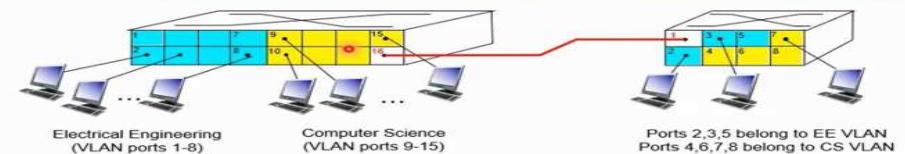
port-based VLAN: switch ports grouped (by switch management software) so that **single** physical switch



... operates as **multiple** virtual switches

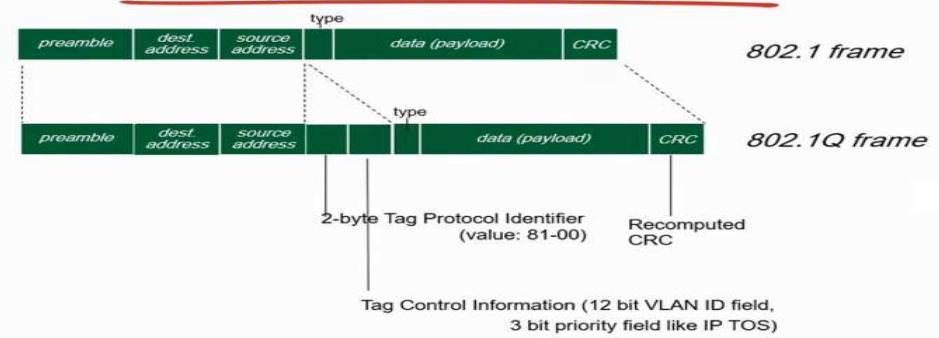


VLANs spanning multiple switches



- ❖ **trunk port:** carries frames between VLANs defined over multiple physical switches
 - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
 - 802.1Q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN frame format



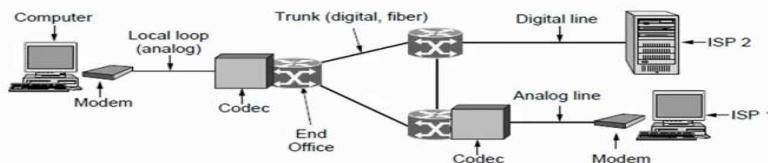
Introduction to Network _ Physical Layer_Basic Concept

Local Loop

Local loop (1): modems

Telephone modems send digital data over an 3.3 KHz analog voice channel interface to the POTS

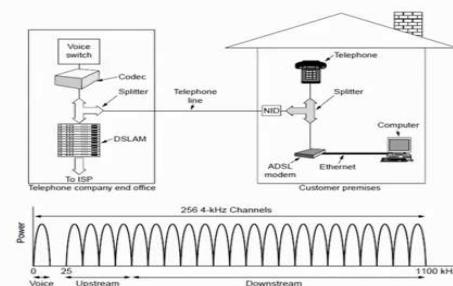
- Rates <56 kbps; early way to connect to the Internet



Local loop (2): Digital Subscriber Lines

DSL broadband sends data over the local loop to the local office using frequencies that are not used for POTS

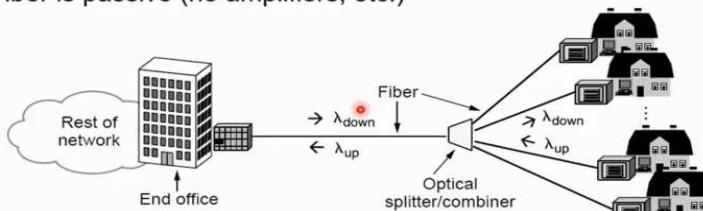
- Telephone/computers attach to the same old phone line
- Rates vary with line
 - ADSL2 up to 12 Mbps
- OFDM is used up to 1.1 MHz for ADSL2
 - Most bandwidth down



Local loop (3): Fiber To The Home

FTTH broadband relies on deployment of fiber optic cables to provide high data rates customers

- One wavelength can be shared among many houses
- Fiber is passive (no amplifiers, etc.)

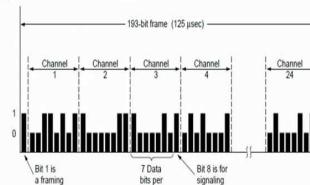


Trunk and Multiplexing and Switching

Trunks and Multiplexing (1)

Calls are carried digitally on PSTN trunks using TDM

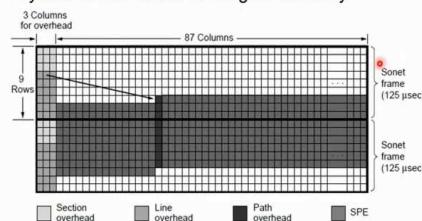
- A call is an 8-bit PCM sample each 125 µs (64 kbps)
- Traditional T1 carrier has 24 call channels each 125 µs (1.544 Mbps) with symbols based on AMI



Trunks and Multiplexing (2)

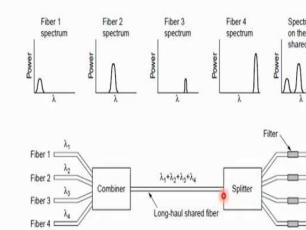
SONET (Synchronous Optical NETwork) is the worldwide standard for carrying digital signals on optical trunks

- Keeps 125 µs frame; base frame is 810 bytes (52Mbps)
- Payload "floats" within framing for flexibility



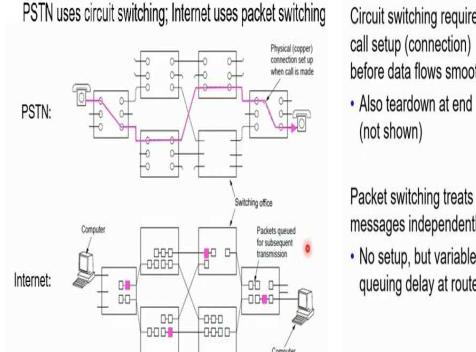
Trunks and Multiplexing (4)

WDM (Wavelength Division Multiplexing), another name for FDM, is used to carry many signals on one fiber:



Switching (1)

PSTN uses circuit switching; Internet uses packet switching



Switching (2)

Circuit switching requires call setup (connection) before data flows smoothly

- Also teardown at end (not shown)

Packet switching treats messages independently

- No setup, but variable queuing delay at routers

Switching (3)

Comparison of circuit- and packet-switched networks

Item	Circuit switched	Packet switched
Call setup	Required	Not needed
Dedicated physical path	Yes	No
Each packet follows the same route	Yes	No
Packets arrive in order	Yes	No
Is a switch crash fatal	Yes	No
Bandwidth available	Fixed	Dynamic
Time of possible congestion	At setup time	On every packet
Potentially wasted bandwidth	Yes	No
Store-and-forward transmission	No	Yes
Charging	Per minute	Per packet