

CS2105

An Awesome Introduction to Computer Networks

Lecture 1: Introduction



© CS2105

Lecture 1 - 4

Lecture 1 - 2

© CS2105

Course Instructors



Dr. Zhou Lifeng
Senior Lecturer
Office: COM2 #02-56
Email: zhoulifeng@nus.edu.sg



Dr. Leong Wai Kay
Lecturer
Office: COM2 #02-11
Email: waikay@comp.nus.edu.sg

© CS2105

What is CS2105 About?

- ❖ Discussion of fundamental **concepts** and **principles** behind computer networking
 - Using the [Internet](#) as a case study
- ❖ Introduction to networking tools and networked application programming
 - Programming with [Python](#), [Java](#), or [C++](#)

Lecture 1 - 4

© CS2105

What you will NOT learn in CS2105

- ❖ How to configure hardware, e.g. router
 - This is covered in [CS3103 Computer Networks Practice](#) - perform hands-on experiments in subnetting, DHCP, DNS, RIP, OSPF, TCP handshaking and congestion mechanism
- ❖ Mobile and wireless networks
 - This is covered in [CS4222 Wireless Networking](#)

Lecture 1 - 5

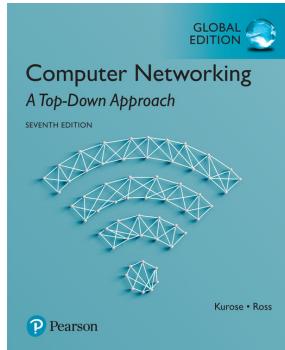
© CS2105

Textbook

Computer Networking: A Top-Down Approach: Global Edition, 7/E

Authors : Kurose
Ross
Publisher : Pearson
ISBN : 9781292153599

Acknowledgement:
Most of the lecture slides are adopted from slides of this textbook.



Available at NUS Campus bookstore

Lecture 1 - 6

© CS2105

Contact Hours

- ❖ Lectures
 - Every [Monday](#) 2 - 4pm @ ICube Auditorium
 - 2 hours per session
 - [week 8 lecture reserved for midterm test](#)
- ❖ Tutorials
 - Start from week 3.
 - 1 hour per session
 - [No tutorial in week 8](#) 😊
- ❖ Consultation
 - Email me or Wai Kay to make arrangement

Lecture 1 - 7

© CS2105

Assessments

- ❖ **CA (50%)**
 - Tutorial attendance/participation - 5%
 - Individual programming assignments - 25%
 - Mid-term test (week 8 lecture time: [Mon, 7 Oct 2019, 2:10 - 3:30pm](#)) - 20%
- ❖ **Final Exam (50%)**
 - [Mon, 2 Dec 2019, 5 - 7pm](#)

Lecture 1 - 8



© CS2105

Notes and Tips

- ❖ Why CS2105 can be **easy**
 - You use and interact with the Internet constantly
 - Many of the concepts are intuitive and based on very practical design considerations
 - There are very few equations!
- ❖ Why CS2105 can be **tough**
 - Many concepts are covered
 - Programming!

Lecture 1 - 9

Lecture 1: Introduction

After this class, you are expected to:

- ❖ understand the basic terms, including host, packet, protocol, throughput, store-and-forward, and autonomous system.
- ❖ know about the **logical** (five protocol layers) and **physical** (a network of ASes) architecture of the Internet.
- ❖ understand the different components of end-to-end delay and their relations to bandwidth, packet size, distance, propagation speed, and queue size.

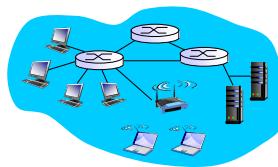
Lecture 1: Roadmap

- 1.1 What is the Internet?
- 1.2 Network Edge
- 1.3 Network Core
- 1.4 Delay, Loss and Throughput in Networks
- 1.5 Protocol Layers and Service Models

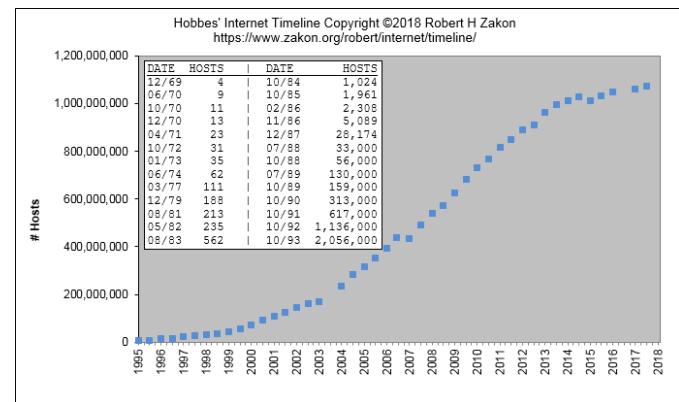
Kurose Textbook, Chapter 1
(Some slides are taken from the book)

Internet: “nuts and bolts” View

- ❖ The Internet is a network of connected computing devices (e.g. PC, server, laptop, smartphone)
 - such devices are known as **hosts** or **end systems**.
 - they run network applications (e.g. WhatsApp, browser).
 - they communicate over links.



Growth of Internet Hosts



“Fun” Internet-connected Devices

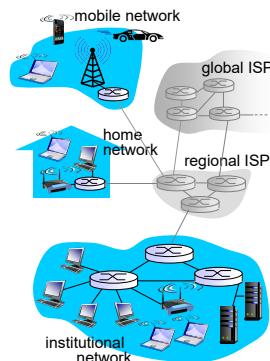


Lecture 1: Roadmap

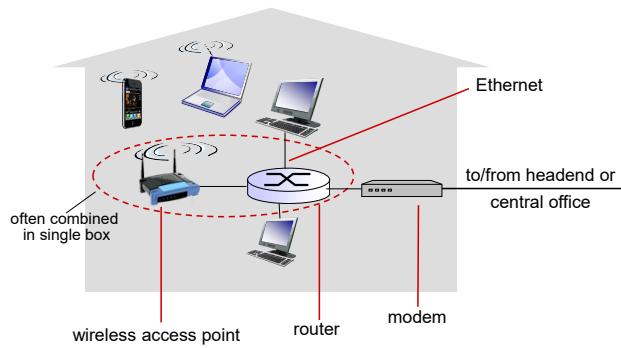
- 1.1 What is the Internet?
- 1.2 Network Edge
 - hosts, access networks, links
- 1.3 Network Core
 - packet switching, circuit switching, network structure
- 1.4 Delay, Loss and Throughput in Networks
- 1.5 Protocol Layers and Service Models

Access Networks

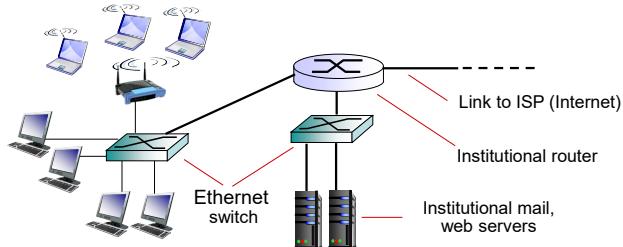
- ❖ Hosts access the Internet through **access network**.
 - Residential access networks
 - Institutional access networks (school, company)
 - Mobile access networks



Home Networks



Enterprise Access Networks (Ethernet)



- Typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- Today, hosts typically connect to Ethernet switch

Physical Media

- Hosts connect to the access network over different physical media.

- Guided media:**

- signals propagate in solid media



Twisted pair cable



Coaxial cable



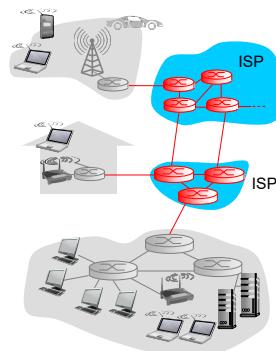
Fiber optic cable

- Unguided media:**

- signals propagate freely, e.g., radio

The Network Core

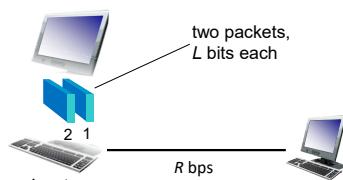
- A mesh of interconnected routers
- The fundamental question: **how is data transmitted through net?**
 - Circuit switching:** dedicated circuit per call
 - Packet switching:** data sent thru net in discrete "chunks"



Packet Switching

Host sending function:

- breaks application message into smaller chunks, known as **packets**, of length L bits
- transmits packets onto the link at **transmission rate R**
 - link transmission rate is aka **link capacity** or **link bandwidth**



$$\text{packet transmission delay} = \frac{\text{time needed to transmit } L\text{-bit packet into link}}{R \text{ (bits/sec)}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Wireless Access Networks

- Wireless access network connects hosts to router
 - via base station aka "access point"

Wireless LANs:

- within building (100 ft)
- 802.11 b/g/n/ac (Wi-Fi)



Wide-area wireless access

- 3G, 4G
- provided by telco (cellular) operator, 10's km



Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

- packet switching, circuit switching, network structure

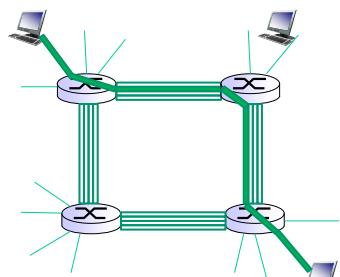
1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

Circuit Switching

End-end resources allocated to and reserved for "call" between source & dest:

- call setup required
- circuit-like (guaranteed) performance
- circuit segment idle if not used by call (*no sharing*)
- commonly used in traditional telephone networks
- divide link bandwidth into "pieces"
 - frequency division
 - time division

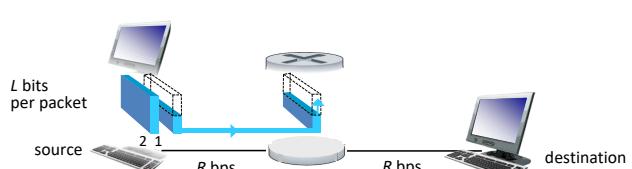


In above diagram, each link has four circuits. A "call" gets 2nd circuit in top link and 1st circuit in right link.

Packet-switching: store-and-forward

- Packets are passed from one **router** to the next, across links on path from source to destination.

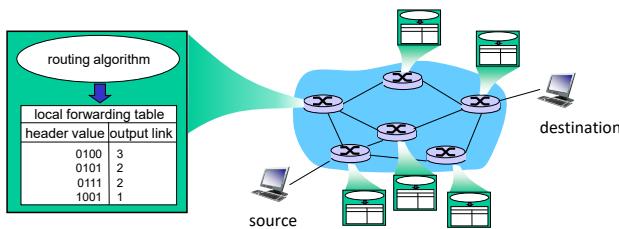
- Store and forward:** entire packet must arrive at a router before it can be transmitted on the next link.



End-to-end delay = $2*L/R$ (assuming no other delay)

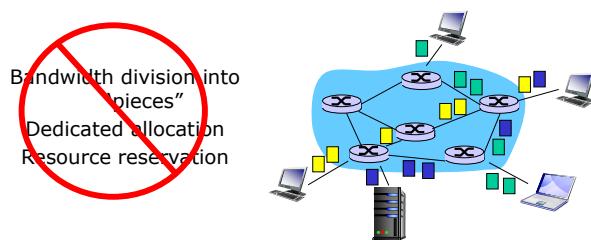
Routing and Addressing

- ❖ Routers determine source-destination route taken by packets.
 - Routing algorithms
- ❖ **Addressing:** each packet needs to carry source and destination information



Packet Switching

- ❖ The Internet is a packet switching network
- ❖ User A, B ... 's packets *share* network resources
- ❖ Resources are used on demand
- ❖ Excessive congestion is possible



Internet Structure: Network of Networks

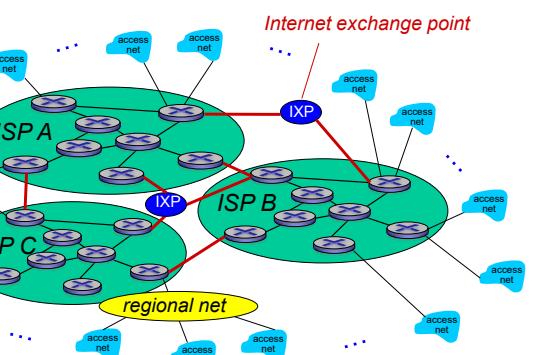
- ❖ Hosts connect to Internet via access **ISPs** (Internet Service Providers)
 - Residential, company and university ISPs
- ❖ Access ISPs in turn must be interconnected.
- ❖ Resulting network of networks is very complex
 - Evolution was driven by **economics** and **national policies**
- ❖ Therefore, the Internet is a “network-of-networks”, organized into autonomous systems (AS), each is owned by an organization.

Internet Structure: Network of Networks

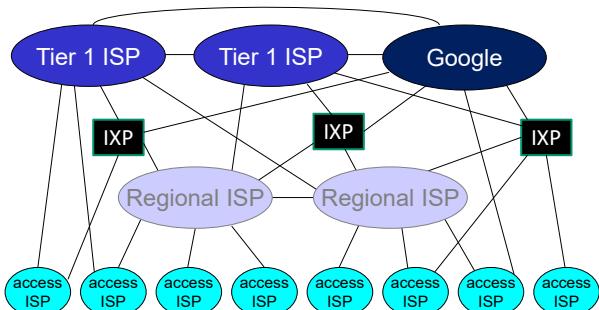
Question: given *millions* of access nets, how to connect them together?



Internet Structure: Network of Networks



Internet Structure: Network of Networks



Who Runs the Internet?

- ❖ IP address & Internet Naming administered by Network Information Centre (NIC)
 - Refer to: www.sgnic.net.sg; www.apnic.org
- ❖ The Internet Society (ISOC) - Provides leadership in Internet related standards, education, and policy around the world.
- ❖ The Internet Architecture Board (IAB) - Authority to issue and update technical standards regarding Internet protocols.
- ❖ Internet Engineering Task Force (IETF) - Protocol engineering, development and standardization arm of the IAB.
 - Internet standards are published as RFCs (Request For Comments)
 - Refer to: www.ietf.org; for RFCs: <http://www.ietf.org/rfc.html>

Lecture 1: Roadmap

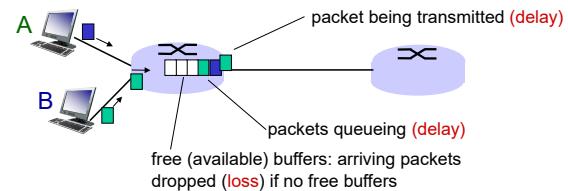
- 1.1 What is the Internet?
- 1.2 Network Edge
 - hosts, access networks, links
- 1.3 Network Core
 - packet switching, circuit switching, network structure
- 1.4 Delay, Loss and Throughput in Networks
- 1.5 Protocol Layers and Service Models

Recall: Packet Switching Network

- To send a packet in a packet switching network,
 - Sender transmit a packet onto the link as a sequence of bits.
 - Bits are propagated to the next node (e.g. a router) on the link.
 - Router stores, processes and forwards the packet to the next link.
 - Steps 2 & 3 repeat till the packet arrives at the receiver.

How do Delay and Loss Occur?

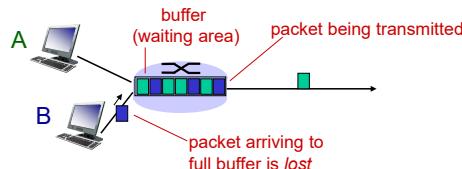
- Packets **queue** in router buffers
 - wait for turn to be sent out one by one



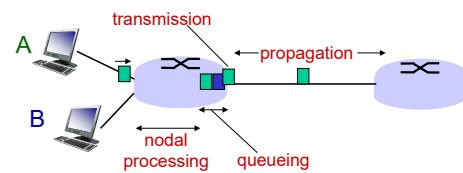
Q: What if packet arrival rate exceeds departure rate?

Packet Loss

- Queue (aka **buffer**) of a router has finite capacity.
- Packet arriving to full queue will be dropped (aka lost).
- Lost packet may be retransmitted by previous node, by source host, or not at all.



Four Sources of Packet Delay



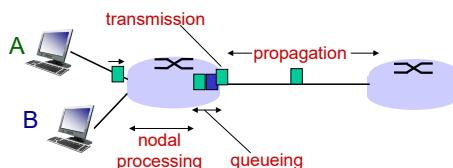
d_{proc} : nodal processing

- check bit errors
- determine output link
- typically < msec

d_{queue} : queueing delay

- time waiting in the queue for transmission
- depends on congestion level of router

Four Sources of Packet Delay



d_{trans} : transmission delay

- L: packet length (bits)
- R: link bandwidth (bps)
- $d_{trans} = L/R$

d_{prop} : propagation delay

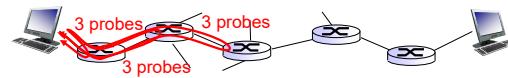
- d: length of physical link
- s: propagation speed in medium ($\sim 2 \times 10^8$ m/sec)
- $d_{prop} = d/s$

End-to-end Packet Delay

- End-to-end packet delay is the time taken for a packet to travel from source to destination. It consists of:

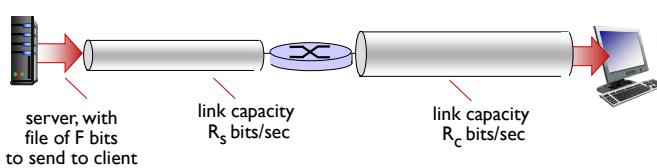
- transmission delay
- propagation delay
- processing delay
- queueing delay

traceroute program displays the route (path) from source to destination and measures the delay from source to each router along the end-end Internet path.



Throughput

- Throughput: how many bits can be transmitted per unit time.
 - Throughput is measured for end-to-end communication.
 - Link capacity (bandwidth) is meant for a specific link.



Metric Units

- 1 byte = 8 bits

| Exp. | Explicit | Prefix | Exp. | Explicit | Prefix |
|------------|------------------------|--------|-----------|-----------------------------------|--------|
| 10^{-3} | 0.001 | milli | 10^3 | 1,000 | Kilo |
| 10^{-6} | 0.000001 | micro | 10^6 | 1,000,000 | Mega |
| 10^{-9} | 0.00000001 | nano | 10^9 | 1,000,000,000 | Giga |
| 10^{-12} | 0.000000000001 | pico | 10^{12} | 1,000,000,000,000 | Tera |
| 10^{-15} | 0.00000000000001 | femto | 10^{15} | 1,000,000,000,000,000 | Peta |
| 10^{-18} | 0.0000000000000001 | atto | 10^{18} | 1,000,000,000,000,000,000 | Exa |
| 10^{-21} | 0.000000000000000001 | zepto | 10^{21} | 1,000,000,000,000,000,000,000 | Zetta |
| 10^{-24} | 0.00000000000000000001 | yocto | 10^{24} | 1,000,000,000,000,000,000,000,000 | Yotta |

The principal metric prefixes

Lecture 1: Roadmap

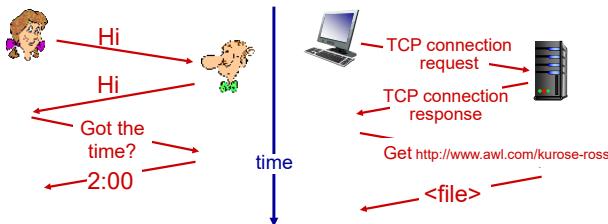
- 1.1 What is the Internet?
- 1.2 Network Edge
 - hosts, access networks, links
- 1.3 Network Core
 - packet switching, circuit switching, network structure
- 1.4 Delay, Loss and Throughput in Networks
- 1.5 Protocol Layers and Service Models**

Internet: a Service View

- ❖ The Internet supports various kinds of network applications:
 - Web, VoIP, email, games, e-commerce, social nets, ...
- ❖ Network applications exchange messages and communicate among peers according to **protocols**.

What's a Protocol?

a human protocol and a computer network protocol:



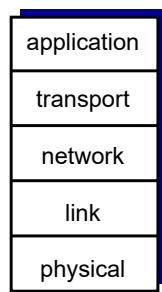
Protocols define **format** and **order** of messages exchanged and the **actions** taken after messages are sent or received.

Protocol “Layers”

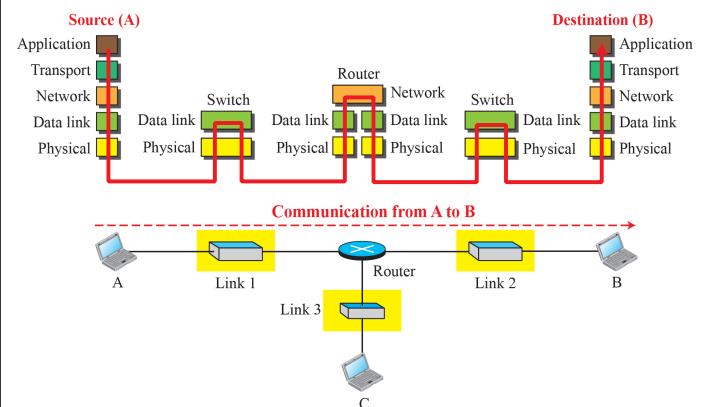
- ❖ Protocols in the Internet are logically organized into “layers” according to their purposes.
 - Each layer provides a service
 - Simple interfaces between layers
 - Hide details from each other
- ❖ Layering is a common CS trick to deal with large and complex systems.
 - Explicit structure allows identification, relationship of complex system’s pieces
 - Modularization eases maintenance, updating of system
 - E.g. change of implementation of one layer’s service is transparent to rest of system

Internet Protocol Stack

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



Example



ISO/OSI reference model (FYI)

- ❖ Theoretical model – not in use
- ❖ Two additional layers not present in Internet Protocol Stack
 - **presentation:** allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
 - **session:** synchronization, checkpointing, recovery of data exchange



Lecture 1: Summary

- covered a “ton” of material!**
- ❖ Internet overview
 - ❖ Network edge, core, access network
 - packet-switching versus circuit-switching
 - Internet structure
 - ❖ Performance: loss, delay, throughput
 - ❖ What’s a protocol?
 - ❖ Layering, service models
- you now have:**
- ❖ Context, overview, “feel” of networking
 - ❖ More depth, detail to follow!

CS2105

An **Awesome** Introduction to Computer Networks

Lectures 2&3: The Application Layer



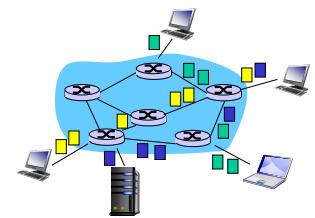
© CS2105

PREVIOUS LECTURE

Packet Switching

- The Internet is a **packet switching** network
 - Hosts share and contend network resources.
 - Application message** is broken into a bunch of packets and sent onto the link one by one.
 - A router stores and forwards packets.
 - Receiver assembles all the packets to restore the **application message**.

Bandwidth division into "pieces"
Dedicated allocation
Resource reservation

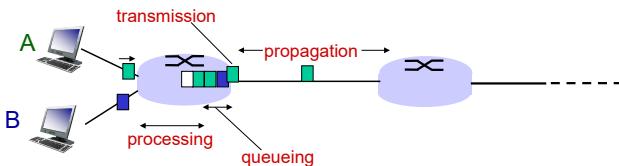


© CS2105

PREVIOUS LECTURE

Packet Delay

- End-to-end delay is the time taken for a packet to travel from source to destination. It consists of:
 - processing delay
 - queueing delay
 - transmission delay
 - propagation delay



© CS2105

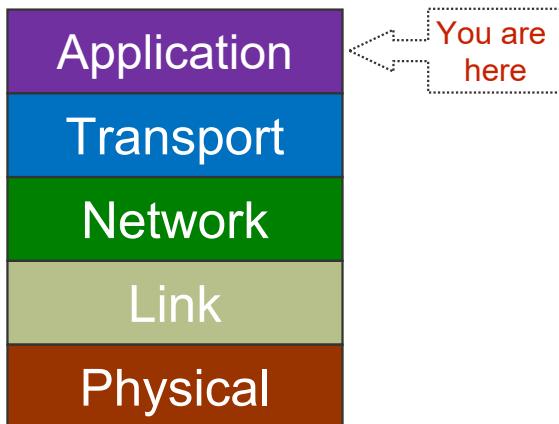
PREVIOUS LECTURE

Network Protocols

- Networks are complex. There are many issues to consider, to support different applications running on large number of hosts through different access technology and physical media.
- Protocols** regulate communication activities in a network.
 - Define the **format** and **order** of messages exchanged between hosts for a specific purpose.

© CS2105

Lectures 2&3 - 5



© CS2105

Lectures 2&3 - 6

Lectures 2&3: Application Layer

After this class, you are expected to:

- understand the basic HTTP interactions between the client and the server, the concepts of persistent and non-persistent connections.
- understand the services provided by DNS and how a DNS query is resolved.
- understand the concept of socket.
- be able to write simple client/server programs through socket programming.

© CS2105

Lectures 2&3 - 7

Lectures 2&3: Roadmap

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- 2.4 DNS
- 2.7 Socket programming

To discuss next week

© CS2105

Lectures 2&3 - 8

Evolution of Network Applications

- Early days of Internet
 - Remote access (e.g. telnet, now ssh)
- 70s – 80s
 - Email, FTP
- 90s
 - Web
- 2000s – now
 - P2P file sharing
 - Online games
 - Instant Messaging, Skype
 - YouTube, Facebook

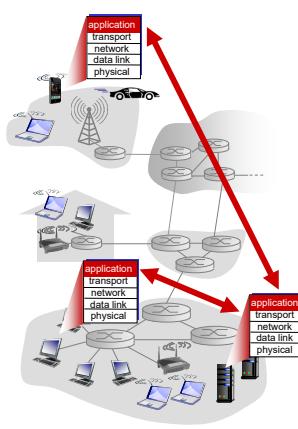
Creating Network Applications

write programs that:

- ❖ run on (different) hosts
- ❖ communicate over network
- ❖ e.g., web server software ⇔ browser software

classic structure of network applications:

- ❖ Client-server
- ❖ Peer-to-peer (P2P)



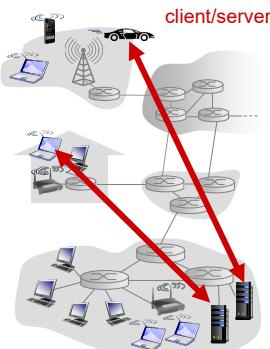
Client-Server Architecture

Server:

- ❖ Waits for incoming requests
- ❖ Provides requested service to client
- ❖ data centers for scaling

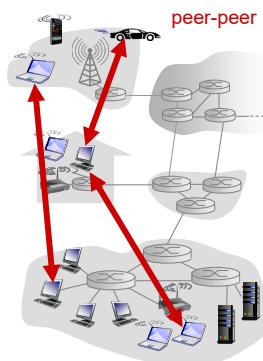
Client:

- ❖ Initiates contact with server ("speaks first")
- ❖ Typically requests service from server
- ❖ For Web, client is usually implemented in browser



P2P Architecture

- ❖ No always-on server
- ❖ Arbitrary end systems directly communicate.
- ❖ Peers request service from other peers, provide service in return to other peers
 - **self scalability** – new peers bring new service capacity, as well as new service demands
- ❖ Peers are intermittently connected and change IP addresses
 - complex management



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 streaming) can tolerate some data loss

Timing

- ❖ some apps (e.g., online interactive games) require low delay to be "effective"

Throughput

- ❖ some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- ❖ other apps (e.g., file transfer) make use of whatever throughput available

Security

- ❖ encryption, data integrity, authentication ...

Requirements of Example Apps

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

App-layer Protocols Define...

- ❖ **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 applications send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

Internet Transport Protocols

TCP service:

- ❖ **reliable transport** between sending and receiving process
- ❖ **flow control**: sender won't overwhelm receiver
- ❖ **congestion control**: throttle sender when network is overloaded
- ❖ **does not provide**: timing, minimum throughput guarantee, security

UDP service:

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

Example App/Transport Protocols

| Application | Application Layer Protocol | Underlying Transport Protocol |
|------------------------|--|-------------------------------|
| Electronic mail | SMTP [RFC 5321] | TCP |
| Remote terminal access | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| File transfer | FTP [RFC 959] | TCP |
| Streaming multimedia | HTTP (e.g., YouTube) | TCP |
| Internet telephony | SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype) | TCP or UDP |

Lectures 2&3: Roadmap

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- 2.4 DNS
- 2.7 Socket programming

The Web: Some Jargon

- ❖ A Web page typically consists of:
 - *base HTML file*, and
 - *several referenced objects*.
- ❖ An object can be HTML file, JPEG image, Java applet, audio file, ...
- ❖ Each object is addressable by a *URL*, e.g.,

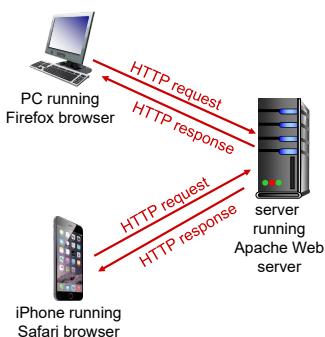
`www.comp.nus.edu.sg/~cs2105/img/doge.jpg`

host name path name

HTTP Overview

HTTP: Hypertext transfer protocol

- ❖ Web's application layer protocol
- ❖ Client/server model
 - *client*: usually is browser that requests, receives and displays Web objects
 - *server*: Web server sends objects in response to requests
- ❖ http 1.0: RFC 1945
- ❖ http 1.1: RFC 2616



HTTP Over TCP

HTTP uses TCP as transport service

- ❖ Client initiates TCP connection to server.
- ❖ Server accepts TCP connection request from client.
- ❖ HTTP messages are exchanged between browser (HTTP client) and Web server (HTTP server) over TCP connection.
- ❖ TCP connection closed.

Two Types of HTTP Connections

Non-persistent HTTP

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

Persistent HTTP

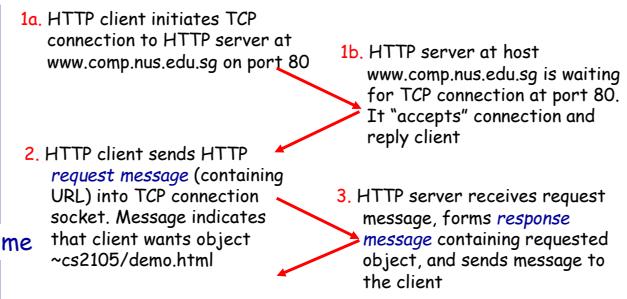
- ❖ multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP Example

suppose user enters URL:

`www.comp.nus.edu.sg/~cs2105/demo.html`

(contains text, reference to a jpeg image)



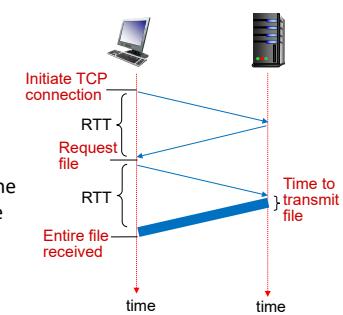
Non-persistent HTTP Example

Non-persistent HTTP: Response Time

RTT: time for a packet to travel from client to server and go back

HTTP response time:

- ❖ one RTT to establish TCP connection
- ❖ one RTT for HTTP request and the first few bytes of HTTP response to return
- ❖ file transmission time
- ❖ non-persistent HTTP response time = $2 * \text{RTT} + \text{file transmission time}$



time

- 5. HTTP client receives response message containing html file, displays html. Parsing html file, client notices the referenced jpeg object
- 6. Steps 1-5 repeated for the jpeg object
- This is an example of *non-persistent connection* (http 1.0).
 - One object per connection
- HTTP 1.1 allows *persistent connection* (to discuss).
 - Multiple objects per connection

Persistent HTTP

non-persistent HTTP issues:

- ❖ requires 2 RTTs per object
- ❖ OS overhead for *each* TCP connection
- ❖ browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- ❖ server leaves connection open after sending response
- ❖ subsequent HTTP messages between same client/server sent over the same TCP connection
- ❖ moreover, client may send requests as soon as it encounters a referenced object (**persistent with pipelining**)
- ❖ as little as one RTT for all the referenced objects

Example HTTP Request Message

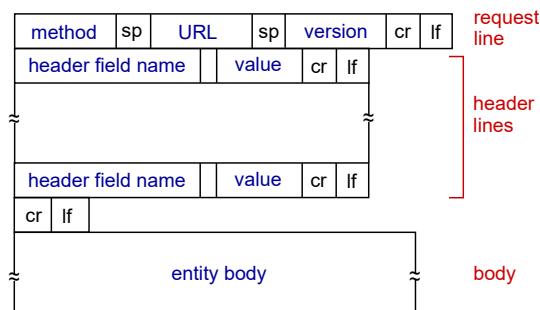
- ❖ Two types of HTTP messages: *request, response*

HTTP request message:

```

request line (GET method)
GET /index.html HTTP/1.1\r\n
header lines   Host: www.example.org\r\n
               Connection: keep-alive\r\n
               ...
               \r\n
Extra "blank" line indicates the end of header lines
  
```

HTTP Request Message: General Format



HTTP Request Method Types

HTTP/1.0:

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

Example HTTP Response Message

```

status line (protocol status code)
HTTP/1.1 200 OK\r\n
Date: Wed, 23 Jan 2019 13:11:15 GMT\r\n
Content-Length: 606\r\n
Content-Type: text/html\r\n
...
\r\n
data data data data data ...   data, e.g. requested HTML file
  
```

For a full list of request/response header fields, check
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

HTTP Response Status Codes

- ❖ Status code appears in 1st line in server-to-client response message.

Some sample codes:

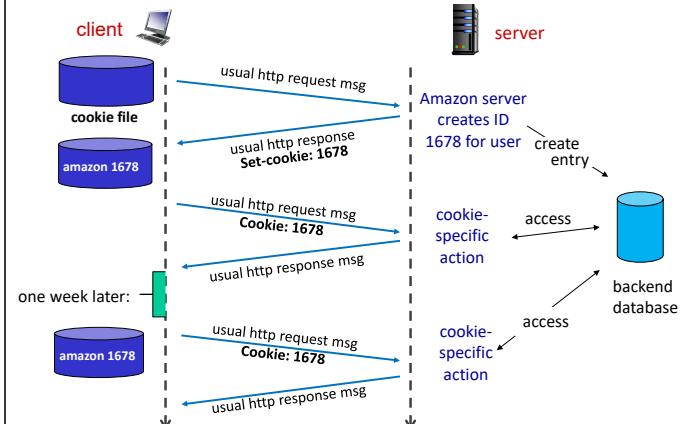
- | | |
|------------------------------|--|
| 200 OK | ▪ request succeeded, requested object later in this msg |
| 301 Moved Permanently | ▪ requested object moved, new location specified later in this msg (Location:) |
| 403 Forbidden | ▪ server declines to show the requested webpage |
| 404 Not Found | ▪ requested document not found on this server |

For a full list of status codes, check
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Cookies

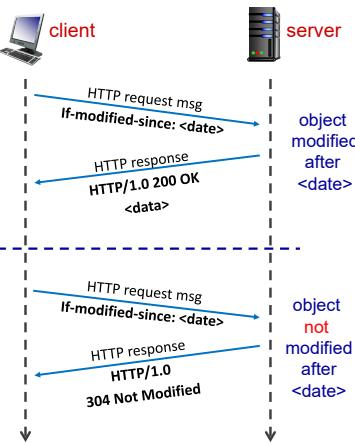
- ❖ HTTP is designed to be “stateless”.
 - Server maintains no information about past client requests.
- ❖ Sometimes it’s good to maintain states (history) at server/client over multiple transactions.
 - E.g. shopping carts
- ❖ Cookie: http messages carry “state”
 - 1) cookie header field of HTTP *request / response* messages
 - 2) cookie file kept on user’s host, managed by user’s browser
 - 3) back-end database at Web site

Keep User States with Cookie



Conditional GET

- ❖ **Goal:** don't send object if (client) cache has up-to-date cached version
- ❖ **cache:** specify date of cached copy in HTTP request
If-modified-since:
 `<date>`
- ❖ **server:** response contains no object if cached copy is up-to-date:
HTTP/1.0 304 Not Modified



Lectures 2&3: Roadmap

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- 2.4 DNS
- 2.7 Socket programming

Domain Name System [RFC 1034, 1035]

- ❖ Two ways to identify a host:
 - **Hostname**, e.g., www.example.org
 - **IP address**, e.g., 93.184.216.34
- ❖ **DNS (Domain Name System)** translates between the two.
 - A client must carry out a DNS query to determine the IP address corresponding to the server name (e.g., www.example.org) prior to the connection.

DNS: Resource Records (RR)

- ❖ Mapping between host names and IP addresses (and others) are stored as resource records (RR).

RR format: (name, value, type, ttl)

type = A

- **name** is hostname
- **value** is IP address

type = CNAME

- **name** is alias name for some “canonical” (the real) name

- **value** is canonical name
- e.g. `www.nus.edu.sg` is really `mgnzsqc.x.incapdns.net`

type = NS

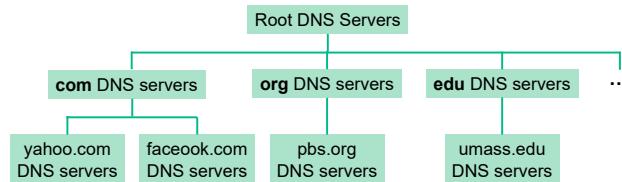
- **name** is domain (e.g., `nus.edu.sg`)
- **value** is hostname of authoritative name server for this domain

type = MX

- **value** is name of mail server associated with **name**

Distributed, Hierarchical Database

- ❖ DNS stores RR in distributed databases implemented in hierarchy of many name servers.

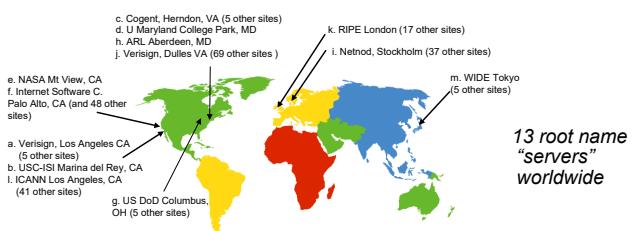


A client wants IP address for `www.facebook.com`:

- ❖ client queries root server to find .com DNS server
- ❖ client queries .com DNS server to get facebook.com DNS server
- ❖ client queries facebook.com DNS server to get IP address for `www.facebook.com`

Root Servers

- ❖ Answers requests for records in the root zone by returning a list of the authoritative name servers for the appropriate top-level domain (TLD).



TLD and Authoritative Servers

Top-level domain (TLD) servers:

- ❖ responsible for com, org, net, edu, ... and all top-level country domains, e.g., uk, sg, jp

Authoritative servers:

- ❖ Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts (e.g. Web, mail)
- ❖ can be maintained by organization or service provider

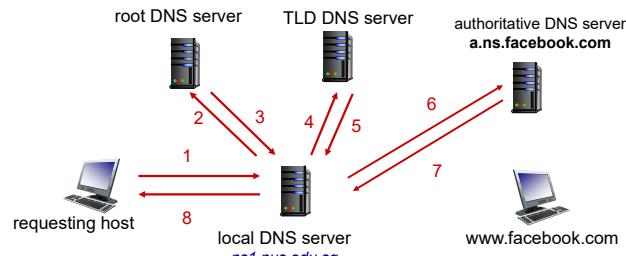
Local DNS Server

- ❖ Does not strictly belong to hierarchy
- ❖ Each ISP (residential ISP, company, university) has one local DNS server.
 - also called “default name server”
- ❖ When host makes a DNS query, query is sent to its local DNS server
 - Retrieve name-to-address translation from local cache
 - Local DNS server acts as proxy and forwards query into hierarchy if answer is not found locally

DNS Caching

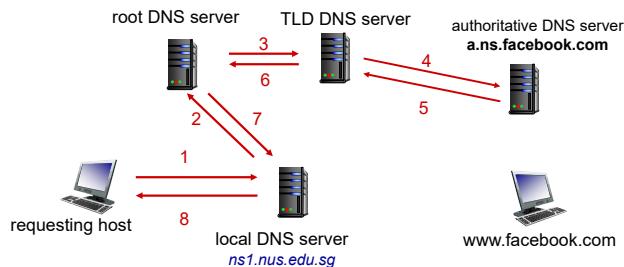
- ❖ Once a name server learns mapping, it *caches* mapping.
 - cached entries may be out-of-date (best effort name-to-address translation!)
 - cached entries expire after some time (TTL).
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- ❖ Update/notify mechanisms proposed IETF standard
 - RFC 2136
- ❖ DNS runs over **UDP**.

DNS Name Resolution



- ❖ This is known as *iterative query*.

DNS Name Resolution



- ❖ This is known as *recursive query*.
 - rarely happens in practice

Lectures 2&3: Roadmap

- 2.1 Principles of Network Applications**
- 2.2 Web and HTTP**
- 2.4 DNS**
- 2.7 Socket programming**

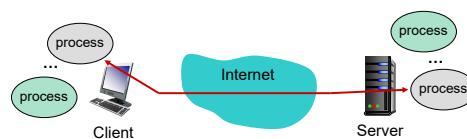
Processes

- ❖ Process: program running within a host.
 - Within the same host, two processes communicate using **inter-process communication** (defined by OS).
 - Processes in different hosts communicate by exchanging **messages** (according to protocols).

In C/S model
server process waits to be contacted
client process initiates the communication

Addressing Processes

- ❖ **IP address** is used to identify a host device
 - A 32-bit integer (e.g. 137.132.21.27)
 - ❖ **Question:** is IP address of a host suffice to identify a process running inside that host?
- A: no, many processes may run concurrently in a host.



Addressing Processes

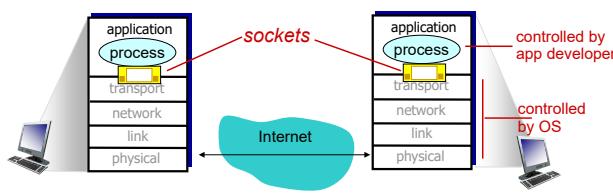
- ❖ A process is identified by (**IP address, port number**).
 - Port number is 16-bit integer (1-1023 are reserved for standard use).
- ❖ Example port numbers
 - HTTP server: 80
 - SMTP server: 25
- ❖ IANA coordinates the assignment of port number:
 - <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>

Analogy

- | | |
|--|--|
| Postal service: | Protocol service: |
| ❖ <i>deliver letter to the doorstep</i> : home address | ❖ <i>deliver packet to the right host</i> : IP address of the host |
| ❖ <i>dispatch letter to the right person in the house</i> : name of the receiver as stated on the letter | ❖ <i>dispatch packet to the right process in the host</i> : port number of the process |

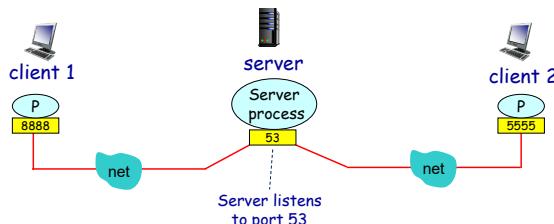
Sockets

- Socket is the software interface between app processes and transport layer protocols.
 - Process sends/receives messages to/from its socket.
 - Programming-wise: a set of APIs



Socket Programming with UDP

- UDP: no “connection” between client and server**
- Sender (client) explicitly attaches destination IP address and port number to each packet.
 - Receiver (server) extracts sender IP address and port number from the received packet.



Example: UDP Echo Server

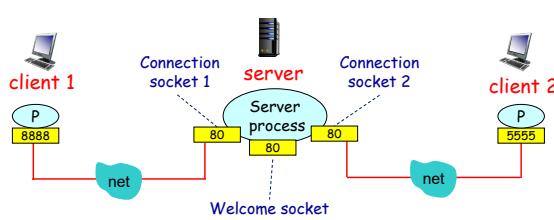
```
from socket import * ← include Python's socket library

serverPort = 2105
# create a socket
serverSocket = socket(AF_INET, SOCK_DGRAM)
# bind socket to local port number 2105
serverSocket.bind(('', serverPort))
print('Server is ready to receive message')
# extract client address from received packet
message, clientAddress = serverSocket.recvfrom(2048)
serverSocket.sendto(message, clientAddress)
serverSocket.close()
```

Annotations: 'IPv4' and 'UDP socket' point to the AF_INET and SOCK_DGRAM parameters respectively. 'receive datagram buffer size: 2048B' points to the 2048 parameter in the recvfrom call.

Socket Programming with TCP

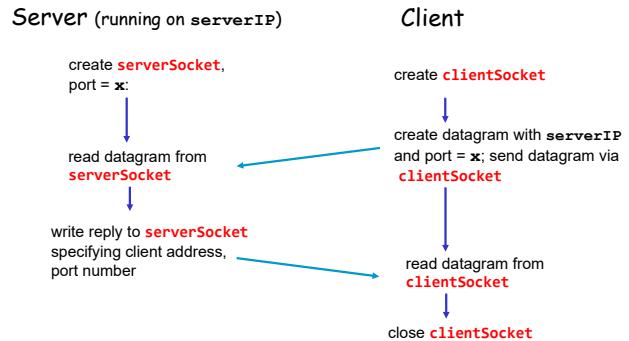
- When client creates socket, client TCP establishes a connection to server TCP.
- When contacted by client, server TCP creates a new socket for server process to communicate with that client.
 - allows server to talk with multiple clients individually.



Socket Programming

- Applications (or processes) treat the Internet as a black box, sending and receiving messages through sockets.
- Two types of sockets
 - TCP: reliable, byte stream-oriented socket
 - UDP: unreliable datagram socket
- Now let's write a simple client/server application that **client sends a line of text to server, and server echoes it.**
 - We will demo both **TCP socket version** and **UDP socket version**.

UDP: Client/server Socket Interaction



Example: UDP Echo Client

```
from socket import *

serverName = 'localhost' # client, server on the same host
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_DGRAM)

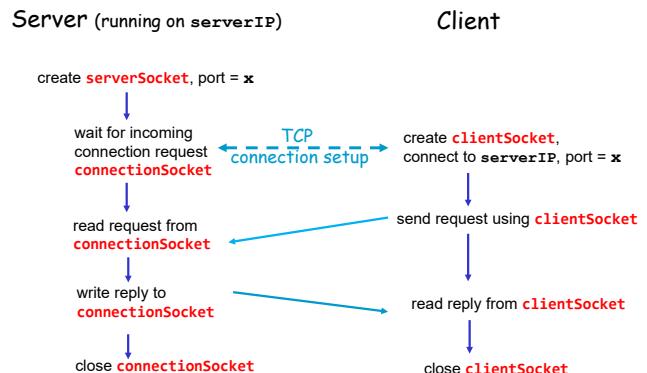
message = input('Enter a message: ')
# send msg to server address
clientSocket.sendto(message.encode(), (serverName, serverPort))

receivedMsg, serverAddress = clientSocket.recvfrom(2048)

print('from server:', receivedMsg.decode())
clientSocket.close()
```

Annotations: 'convert message from string to byte and send it' points to the encode() call. 'convert from byte to string' points to the decode() call.

TCP: Client/server Socket Interaction



Example: TCP Echo Server

```
from socket import *

serverPort = 2105
TCP socket

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', serverPort))
listens for incoming TCP request
(not available in UDP socket)
serverSocket.listen(1) ←

print('Server is ready to receive message')

connectionSocket, clientAddr = serverSocket.accept()
message = connectionSocket.recv(2048)
connectionSocket.send(message)
connectionSocket.close()

connectionSocket.send(message)
returns a new socket
to communicate with
client socket
```

Example: TCP Echo Client

```
from socket import *

serverName = 'localhost'
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort)) ← establish a
connection

message = input('Enter a message: ')
clientSocket.send(message.encode()) ← no need to attach
server name, port

receivedMsg = clientSocket.recv(2048)

print('from server:', receivedMsg.decode())

clientSocket.close()
```

TCP Socket vs. UDP Socket

- ❖ In TCP, two processes communicate as if there is a pipe between them. The pipe remains in place until one of the two processes closes it.
 - When one of the processes wants to send more bytes to the other process, it simply writes data to that pipe.
 - The sending process doesn't need to attach a destination IP address and port number to the bytes in each sending attempt as the logical pipe has been established (which is also reliable).
- ❖ In UDP, programmers need to form UDP datagram packets explicitly and attach destination IP address / port number to every packet.

Lectures 2&3: Summary

- ❖ Application architectures
 - Client-server
 - P2P
- ❖ Application service requirements:
 - reliability, throughput, delay, security
- ❖ Specific protocols:
 - HTTP
 - DNS
- ❖ Internet transport service model
 - connection-oriented, reliable: TCP
 - Connection-less, unreliable: UDP

Lectures 2&3: Summary

- ❖ Socket programming
 - **TCP socket**
 - When contacted by client, server TCP creates new socket.
 - Server uses **(client IP + port #)** to distinguish clients.
 - When client creates its socket, client TCP establishes connection to server TCP.
 - **UDP socket**
 - Server uses **one socket** to serve all clients.
 - No connection is established before sending data.
 - Sender explicitly attaches **destination IP address** and **port #** to **each packet**.
 - Transmitted data may be lost or received out-of-order.

CS2105

An Awesome Introduction to Computer Networks

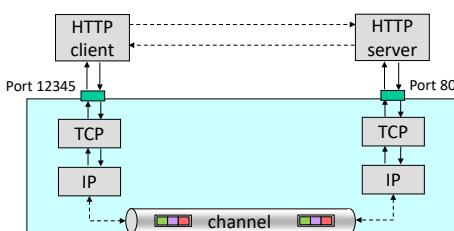
Lectures 4&5: The Transport Layer



Web and HTTP

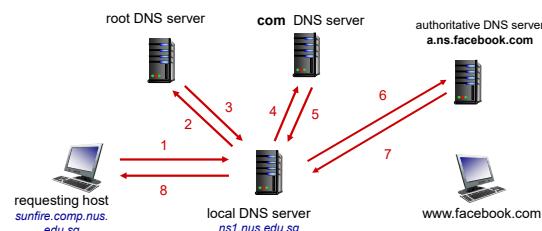
PREVIOUS LECTURE

- ❖ A Web page consists of a **base HTML file** and **some other objects** referenced by the HTML file.
- ❖ HTTP uses **TCP** as transport service.
 - TCP, in turn, uses service provided by **IP**!



Domain Name System

- ❖ DNS is the Internet's primary directory service.
 - It translates **host names**, which can be easily memorized by humans, to **numerical IP addresses** used by hosts for the purpose of communication.



Socket

- ❖ Applications (processes) send messages over the network through sockets.
 - Conceptually, socket = IP address + port number
 - Programming wise, socket = a set of APIs
- ❖ UDP socket
 - Server uses **one socket** to serve all clients.
 - **No connection** is established before sending data.
 - Sender explicitly attaches **destination IP address + port #**.
- ❖ TCP socket
 - Server creates **a new socket** for each client.
 - Client establishes **connection** to server.
 - Server uses **connection** to identify client.

Lectures 4&5: The Transport Layer

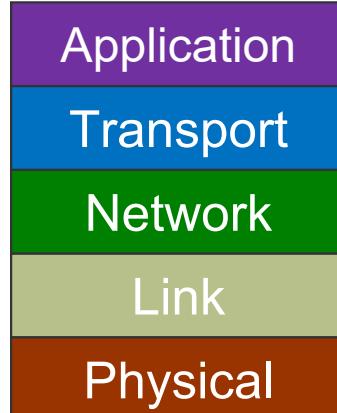
After this class, you are expected to:

- ❖ appreciate the simplicity of UDP and the service it provides.
- ❖ know how to calculate the checksum of a packet.
- ❖ be able to design your own reliable protocols with *ACK, NAK, sequence number, timeout and retransmission*.
- ❖ understand the working of *Go-Back-N* and *Selective Repeat* protocols.
- ❖ understand the operations of TCP.

Lectures 4&5: Roadmap

- 3.1 Transport-layer Services
- 3.3 Connectionless Transport: UDP
- 3.4 Principles of Reliable Data Transfer
- 3.5 Connection-oriented Transport: TCP

Kurose Textbook, Chapter 3
(Some slides are taken from the book)

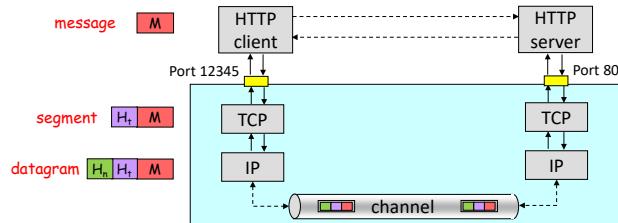


Transport Layer Services

- ❖ Deliver messages between application processes running on different hosts
 - Two popular protocols: **TCP** and **UDP**
- ❖ Transport layer protocols run in hosts.
 - **Sender side:** breaks app message into **segments** (as needed), passes them to network layer (aka IP layer).
 - **Receiver side:** reassembles segments into message, passes it to app layer.
 - **Packet switches (routers) in between:** only check destination IP address to decide routing.

Transport / Network Layers

- ❖ Each IP datagram contains **source and dest IP addresses**.
 - Receiving host is identified by **dest IP address**.
 - Each IP datagram carries one transport-layer segment.
 - Each segment contains **source and dest port numbers**.



Lectures 4&5: Roadmap

- 3.1 Transport-layer Services
- 3.3 Connectionless Transport: UDP
- 3.4 Principles of Reliable Data Transfer
- 3.5 Connection-oriented Transport: TCP

UDP: User Datagram Protocol [RFC 768]

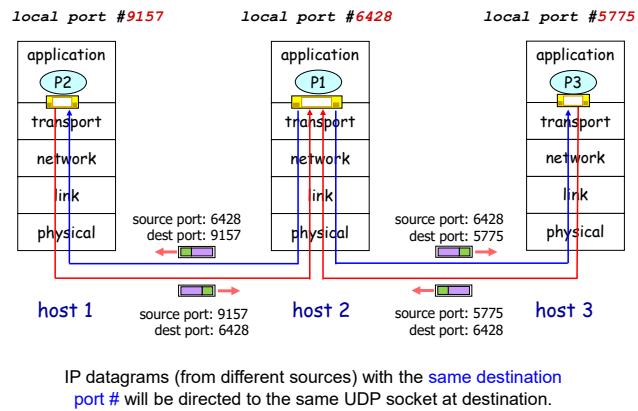
- ❖ UDP adds very little service on top of IP:
 - Connectionless multiplexing / de-multiplexing
 - Checksum
- ❖ UDP transmission is **unreliable**
 - Often used by streaming multimedia apps (loss tolerant & rate sensitive)
- ❖ To achieve reliable transmission over UDP
 - Application implements error detection and recovery mechanisms!

Connectionless De-multiplexing

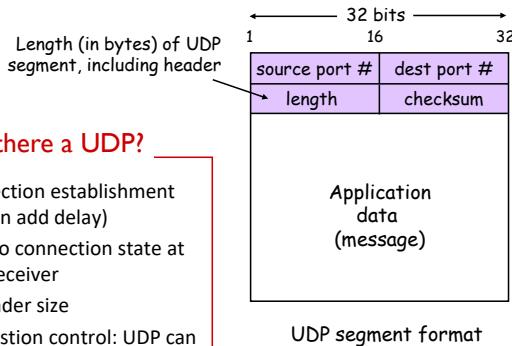
- ❖ **UDP sender:**
 - Creates a socket with **local port #**.
 - When creating a datagram to send to UDP socket, sender must specify **dest. IP address** and **port #**.

- ❖ When **UDP receiver** receives a UDP segment:
 - Checks **destination port #** in segment.
 - Directs UDP segment to the socket with that port #.
 - IP datagrams (from different sources) with the **same destination port #** will be directed to the same UDP socket at destination.

Connectionless De-multiplexing



UDP Header



UDP Checksum

Goal: to detect “errors” (i.e., flipped bits) in transmitted segment.

Sender:

- ❖ compute checksum value (next page)
- ❖ put 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 really no error?)

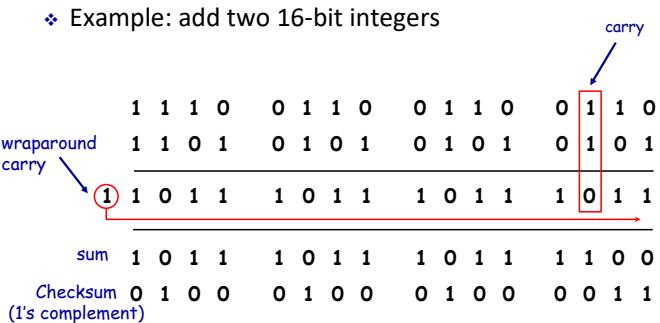
Checksum Computation

- ❖ How is UDP checksum computed?
 1. Treat UDP segment as a sequence of 16-bit integers.
 2. Apply binary addition on every 16-bit integer (checksum field is currently 0).
 3. Carry (if any) from the most significant bit will be added to the result.
 4. Compute 1's complement to get UDP checksum.

| x | y | $x \oplus y$ | carry |
|---|---|--------------|-------|
| 0 | 0 | 0 | - |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | 1 |

Checksum Example

- ❖ Example: add two 16-bit integers



Lectures 4&5: Roadmap

- 3.1 Transport-layer Services
- 3.3 Connectionless Transport: UDP
- 3.4 Principles of Reliable Data Transfer
- 3.5 Connection-oriented transport: TCP

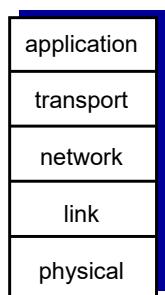


“Sending Data Reliably Over the Internet is Much Harder Than You Think. The Intricacy Involved in Ensuring Reliability Will Make Your Head Explode.”

Transport vs. Network Layer

- Transport layer resides on end hosts and provides process-to-process communication.
- Network layer provides host-to-host, best-effort and unreliable communication.

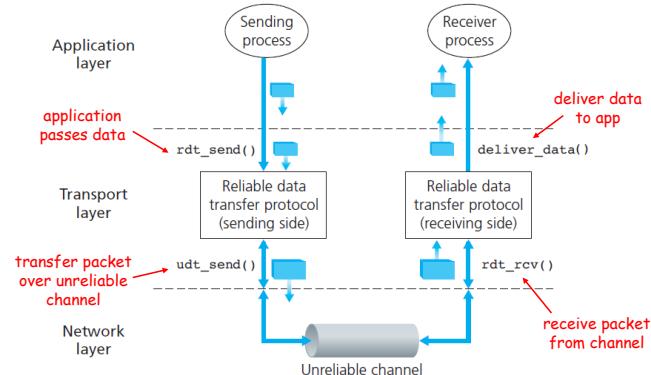
Question: How to build a reliable transport layer protocol on top of unreliable communication?



Reliable Transfer over Unreliable Channel

- Underlying network may
 - corrupt packets
 - drop packets
 - re-order packets (not considered in this lecture)
 - deliver packets after an arbitrarily long delay
- End-to-end reliable transport service should
 - guarantee packets delivery and correctness
 - deliver packets (to application) in the same order they are sent

Reliable Data Transfer: Service Model

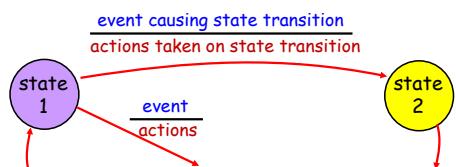


Reliable Data Transfer Protocols

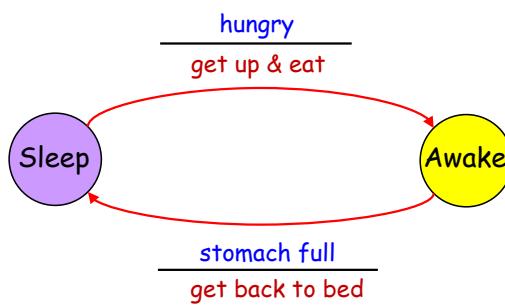
- Characteristics of unreliable channel will determine the complexity of reliable data transfer protocols (**rdt**).
- We will incrementally develop sender & receiver sides of **rdt** protocols, considering increasingly complex models of unreliable channel.
- We consider only unidirectional data transfer
 - but control info may flow in reverse direction!

Finite State Machine (FSM)

- We will use finite state machines (FSM) to describe sender and receiver of a protocol.
 - We will learn a protocol by examples, but FSM provides you the complete picture to refer to as necessary.

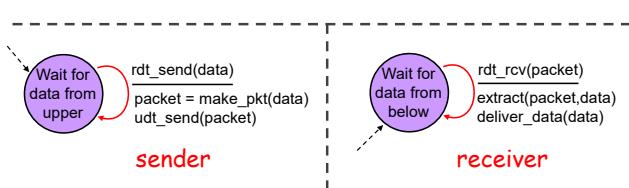


Example FSM



rdt 1.0: Perfectly Reliable Channel

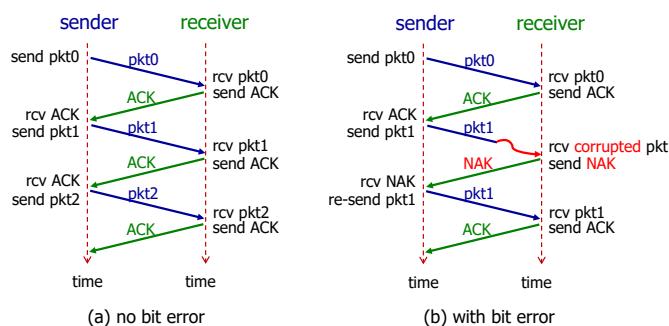
- Assume underlying channel is perfectly reliable.
- Separate FSMs for sender, receiver:
 - Sender sends data into underlying (perfect) channel
 - Receiver reads data from underlying (perfect) channel



rdt 2.0: Channel with Bit Errors

- Assumption:
 - underlying channel may flip bits in packets
 - other than that, the channel is perfect
- Receiver may use checksum to detect bit errors.
- Question: how to recover from bit errors?
 - Acknowledgements (ACKs):** receiver explicitly tells sender that packet received is OK.
 - Negative acknowledgements (NAKs):** receiver explicitly tells sender that packet has errors.
 - Sender retransmits packet on receipt of NAK.

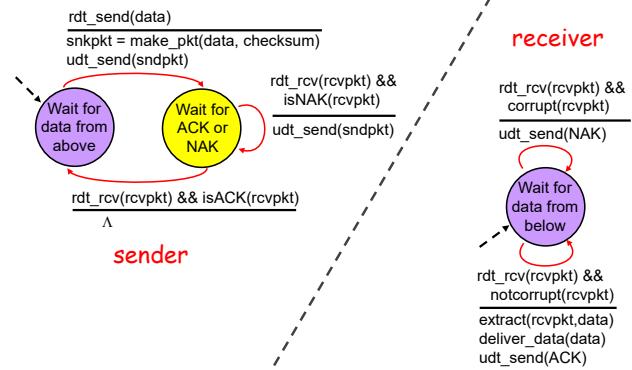
rdt 2.0 In Action



stop and wait protocol

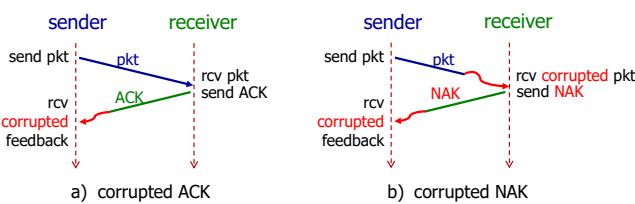
Sender sends one packet at a time, then waits for receiver response

rdt 2.0: FSM



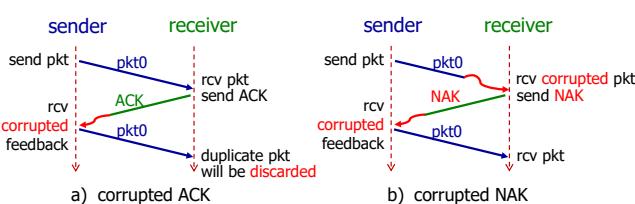
rdt 2.0 has a Fatal Flaw!

- ❖ What happens if ACK/NAK is corrupted?
 - Sender doesn't know what happened at receiver!
- ❖ So what should the sender do?
 - Sender just retransmits when receives garbled ACK or NAK.
 - **Questions:** does this work?

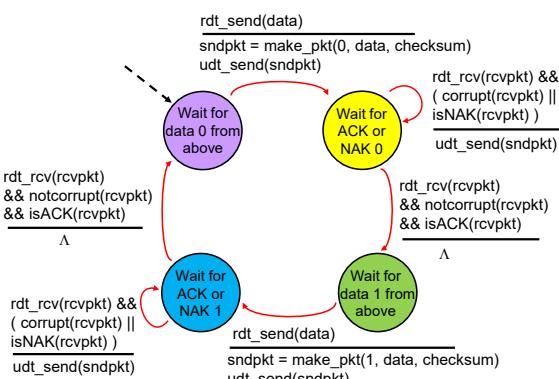


rdt 2.1: rdt 2.0 + Packet Seq.

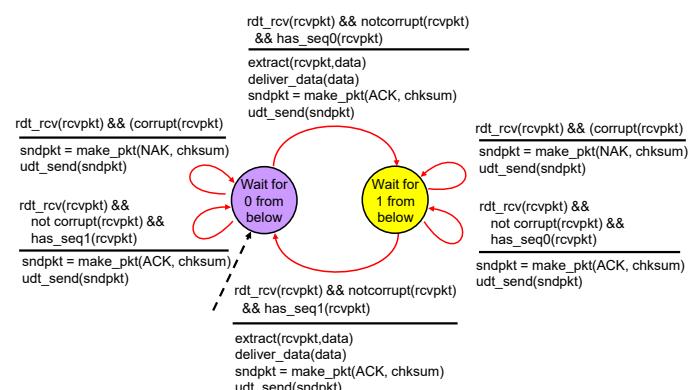
- ❖ To handle duplicates:
 - Sender retransmits current packet if ACK/NAK is garbled.
 - Sender adds **sequence number** to each packet.
 - Receiver discards (doesn't deliver up) duplicate packet.
- ❖ This gives rise to protocol rdt 2.1.



rdt 2.1 Sender FSM



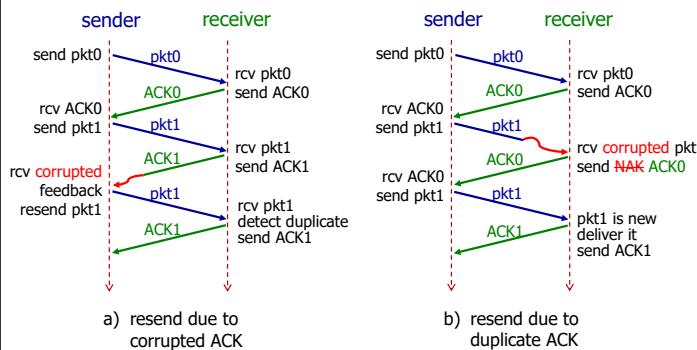
rdt 2.1 Receiver FSM



rdt 2.2: a NAK-free Protocol

- Same assumption and functionality as rdt 2.1, but use ACKs only.
- Instead of sending NAK, receiver sends ACK for the last packet received OK.
 - Now receiver must *explicitly* include seq. # of the packet being ACKed.
- Duplicate ACKs at sender results in same action as NAK: *retransmit current pkt.*

rdt 2.2 In Action



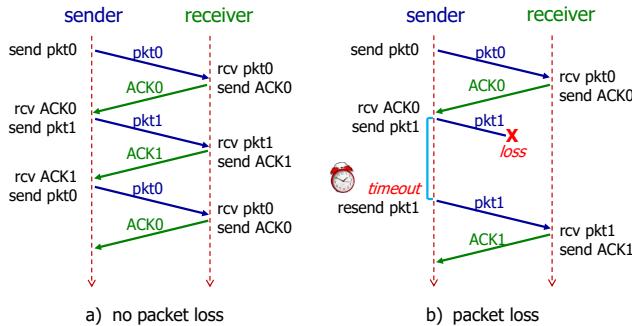
rdt 3.0: Channel with Errors and Loss

- Assumption: underlying channel
 - may flip bits in packets
 - may lose packets
 - may incur arbitrarily long packet delay
 - but won't re-order packets
- Question: how to detect packet loss?
 - checksum, ACKs, seq. #, retransmissions will be of help... but not enough

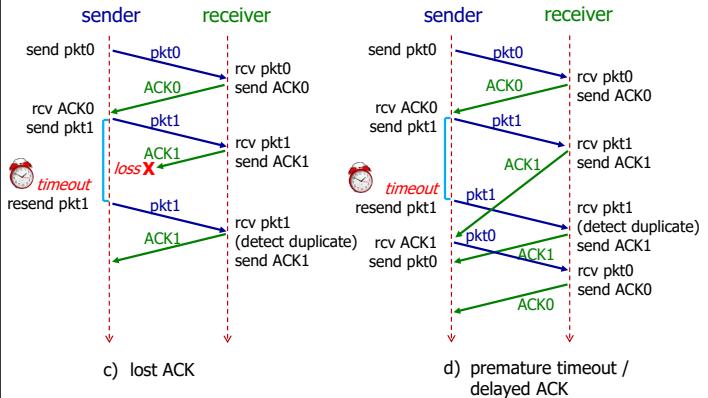
rdt 3.0: Channel with Errors and Loss

- To handle packet loss:
 - Sender waits "reasonable" amount of time for ACK.
 - Sender retransmits if no ACK is received till *timeout*.
- Question: what if packet (or ACK) is just delayed, but not lost?
 - Timeout will trigger retransmission.
 - Retransmission will generate duplicates in this case, but receiver may use seq. # to detect it.
 - Receiver must specify seq. # of the packet being ACKed (check scenario (d) two pages later).

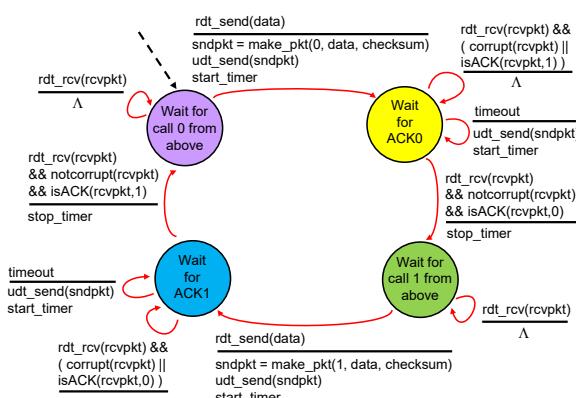
rdt 3.0 In Action



rdt 3.0 In Action



rdt 3.0 Sender FSM



Performance of rdt 3.0

- rdt 3.0 works, but performance stinks.
- Example: packet size = 8000 bits, link rate = 1 Gbps:

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 0.008 \text{ msec}$$

- If RTT = 30 msec, sender sends 8000 bits every 30.008 msec.

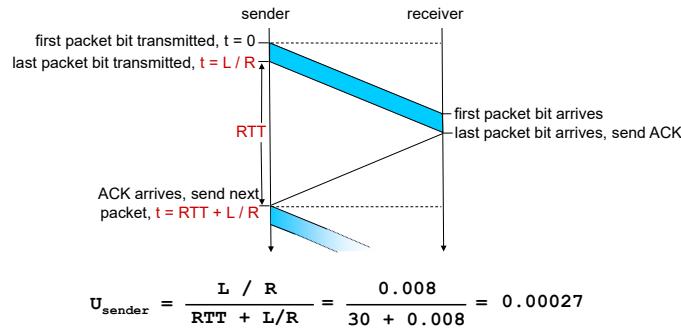
$$\text{throughput} = \frac{L}{RTT + d_{trans}} = \frac{8000}{30 + 0.008} = 267 \text{ kbps}$$

- U_{sender} : *utilization* – fraction of time sender is busy sending

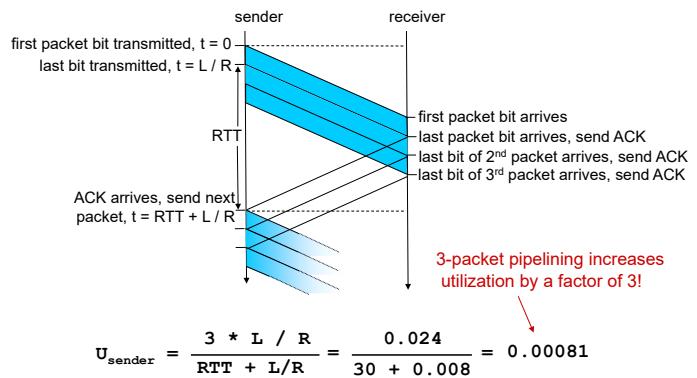
$$U_{\text{sender}} = \frac{d_{trans}}{RTT + d_{trans}} = \frac{0.008}{30 + 0.008} = 0.00027$$

rdt 3.0: Stop-and-wait Operation

- ❖ Network protocol limits use of physical resources!



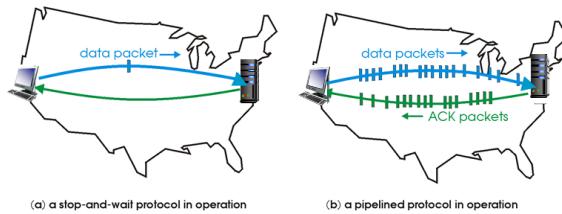
Pipelining: Increased Utilization



Pipelined Protocols

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets.

- ❖ range of sequence numbers must be increased
- ❖ buffering at sender and/or receiver



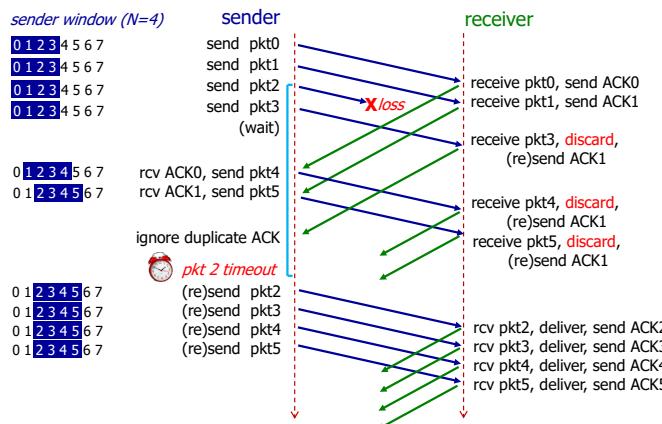
Benchmark Pipelined Protocols

- ❖ Two generic forms of pipelined protocols:
 - *Go-Back-N*
 - *Selective repeat*

- ❖ Assumption (same as rdt 3.0): underlying channel

- may flip bits in packets
- may lose packets
- may incur arbitrarily long packet delay
- but won't re-order packets

Go-back-N In Action



Go-back-N: Key Features

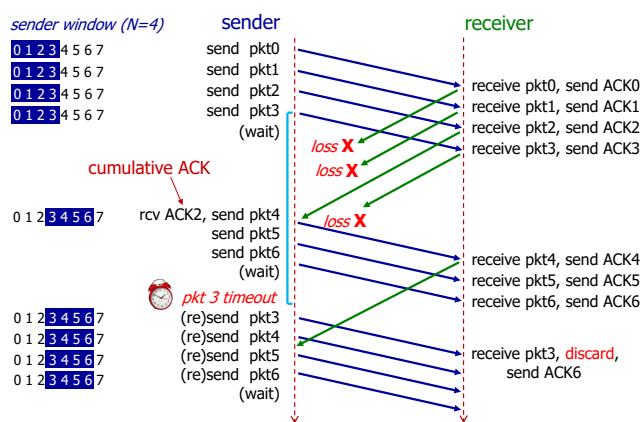
❖ GBN Sender

- can have up to N unACKed packets in pipeline.
- insert k-bits sequence number in packet header.
- use a “sliding window” to keep track of unACKed packets.
- keep a timer for the oldest unACKed packet.
- $\text{timeout}(n)$: retransmit packet n and all subsequent packets in the window.

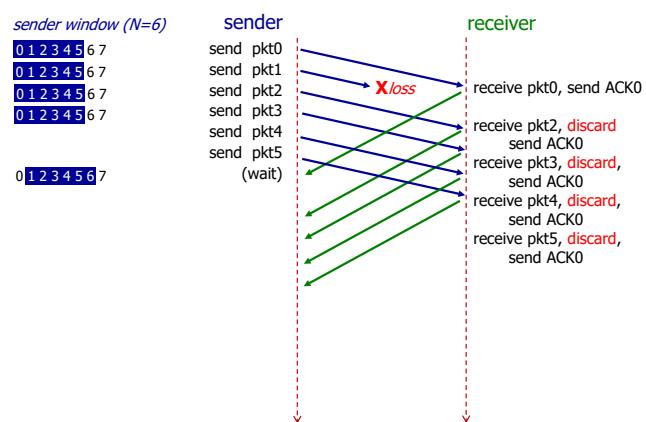
❖ GBN Receiver

- only ACK packets that arrive in order.
 - simple receiver: need only remember `expectedSeqNum`
- discard out-of-order packets and ACK the last in-order seq. #.
- *Cumulative ACK*: “ACK m ” means all packets up to m are received.

Go-back-N In Action



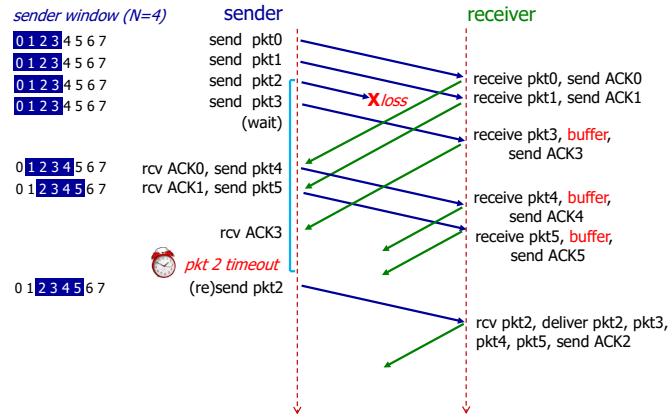
Go-back-N In Action



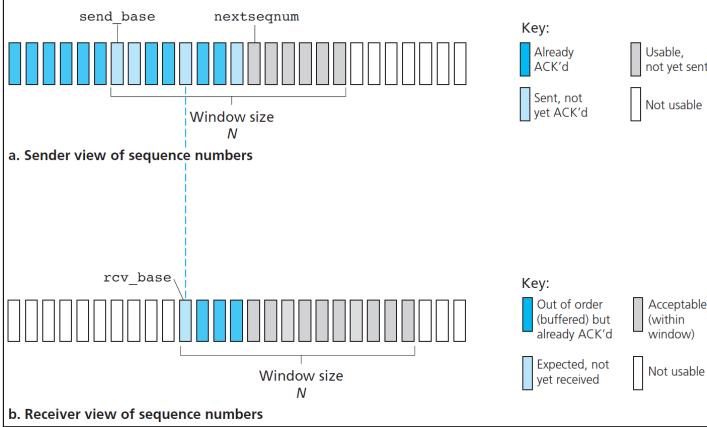
Selective Repeat: Key Features

- Receiver **individually acknowledges** all correctly received packets.
 - Buffers out-of-order packets, as needed, for eventual in-order delivery to upper layer.
- Sender maintains timer for **each unACKed packet**.
 - When timer expires, retransmit only that unACKed packet.

Selective Repeat In Action



SR Sender and Receiver Windows



Selective Repeat: Behaviors

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N]

- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq. #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ACK(n)

otherwise:

- ignore

Lectures 4&5: Roadmap

- 3.1 Transport-layer Services
- 3.2 Multiplexing and De-multiplexing
- 3.3 Connectionless Transport: UDP
- 3.4 Principles of Reliable Data Transfer
- 3.5 Connection-oriented transport: TCP

TCP: Transport Control Protocol

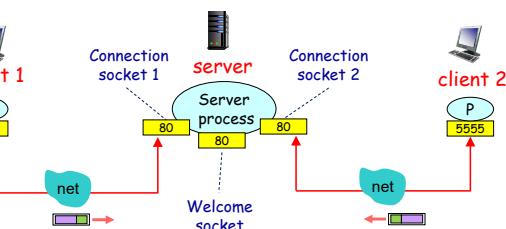
- In contrast to UDP, TCP is complex and is described in tens of RFCs, with new mechanisms or tweaks introduced throughout the years, resulting in many variants of TCP.
- We will only scratch the surface of TCP in CS2105.
 - More will be covered in CS3103.

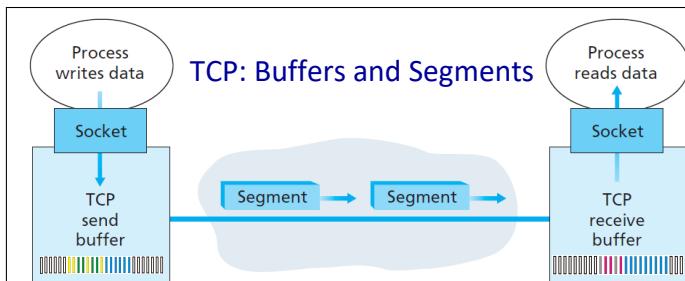
TCP Overview

- Connection-oriented:**
 - handshaking (exchange of control messages) before sending app data.
- Full duplex service:**
 - bi-directional data flow in the same connection
- Reliable, in-order byte stream:**
 - use sequence numbers to label bytes
- Flow control and congestion control**
 - not in syllabus

Connection-oriented De-mux

- A TCP connection (socket) is identified by 4-tuple:
 - (srcIPAddr, srcPort, destIPAddr, destPort)
 - Receiver uses all four values to direct a segment to the appropriate socket.

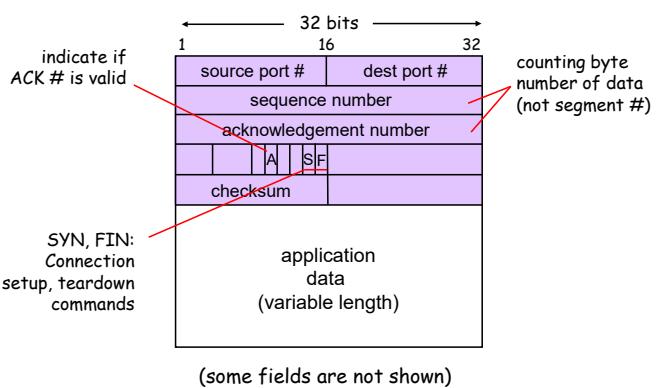




- ❖ TCP send and receive buffers
 - two buffers created after handshaking at any side.
- ❖ How much app-layer data a TCP segment can carry?
 - maximum segment size (**MSS**), typically 1,460 bytes
 - app passes data to TCP and TCP forms packets in view of MSS.

© CS2105

TCP Header



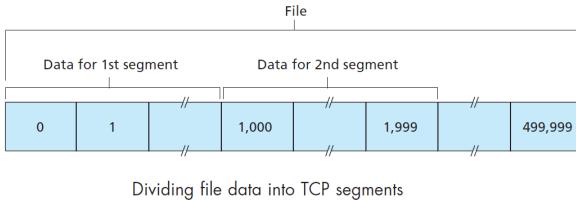
Lectures 4&5 - 61

© CS2105

TCP Sequence Number

| | |
|-----------------|-------------|
| source port # | dest port # |
| sequence number | |
| ACK number | |
| | |
| checksum | |

- ❖ "Byte number" of the first byte of data in a segment.
- ❖ Example: send a file of 500,000 bytes; MSS is 1,000 bytes.



- ❖ Seq. # of 1st TCP segment: 0, 2nd TCP segment: 1,000, 3rd TCP segment: 2,000, 4th TCP segment: 3,000, etc.

© CS2105

TCP ACK Number

| | |
|-----------------|-------------|
| source port # | dest port # |
| sequence number | |
| ACK number | |
| A | |
| checksum | |

- ❖ Seq # of the next byte of data expected by receiver.

| Sequence number of a segment | Amount of data carried | Corresponding ACK number |
|------------------------------|------------------------|--------------------------|
| 0 | 1,000 | 1,000 |
| 1,000 | 1,000 | 2,000 |
| 2,000 | 1,000 | 3,000 |
| 3,000 | 1,000 | 4,000 |
| ... | ... | ... |

- ❖ TCP ACKs up to the first missing byte in the stream (**cumulative ACK**).

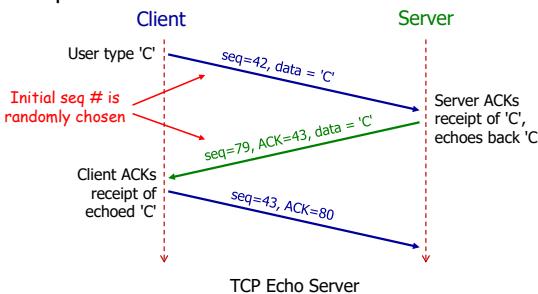
- Note: TCP spec doesn't say how receiver should handle out-of-order segments - it's up to implementer.

© CS2105

Lectures 4&5 - 64

Example: TCP Echo Server

- ❖ TCP (and also UDP) is a full duplex protocol
 - bi-directional data flow in the same TCP connection.
- ❖ Example:

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumberloop (forever) {
switch(event)

event: data received from application above
create TCP segment with sequence number NextSeqNum
if (timer currently not running)
start timer
pass segment to IP
NextSeqNum=NextSeqNum+length(data)
break;

Sender keeps one timer only

event: timer timeout
retransmit not-yet-acknowledged segment with
smallest sequence number
start timer
break;

Retransmit only oldest unACKed packet

event: ACK received, with ACK field value of y
if (y > SendBase) {
SendBase=y
if (there are currently any not-yet-acknowledged segments)
start timer
}
break;

Cumulative ACK

} /* end of loop forever */

© CS2105

Lectures 4&5 - 66

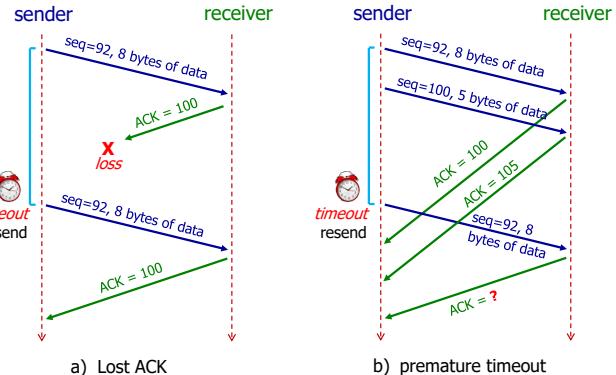
TCP ACK Generation [RFC 5681]

| Event at TCP 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 | Immediately send ACK, provided that segment starts at lower end of gap |

© CS2105

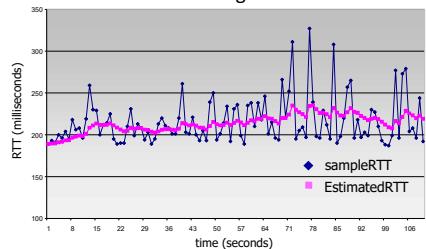
Lectures 4&5 - 67

TCP Timeout / Retransmission



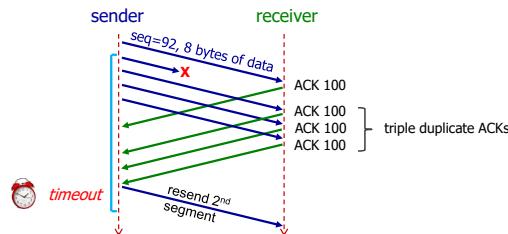
TCP Timeout Value

- How does TCP set appropriate timeout value?
 - too short timeout:** premature timeout and unnecessary retransmissions.
 - too long timeout:** slow reaction to segment loss.
 - Timeout interval must be longer than RTT – but RTT varies!



TCP Fast Retransmission [RFC 2001]

- Timeout period is often relatively long.
 - long delay before resending lost packet
- Fast retransmission:**
 - Event:** If sender receives 4 ACKs for the same segment, it supposes that segment is lost.
 - Action:** resend segment (even before timer expires).

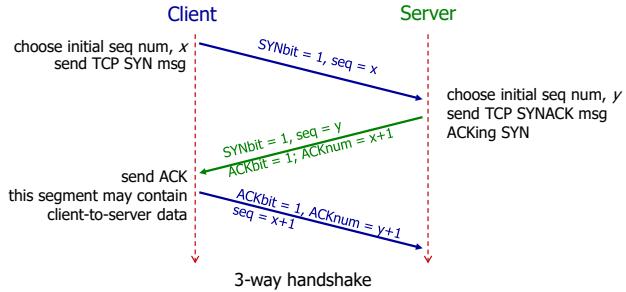


Establishing Connection

| source port # | dest port # |
|-----------------|-------------|
| sequence number | |
| ACK number | |
| A | S |
| checksum | |

- Before exchanging data, TCP sender and receiver “shake hands”.

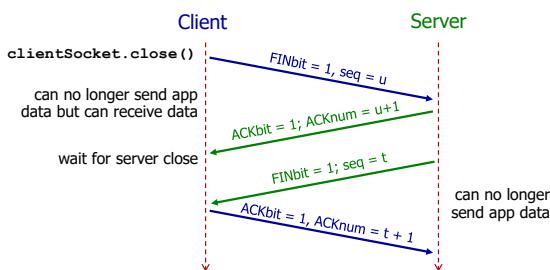
- Agree on connection and exchange connection parameters.



Closing Connection

| source port # | dest port # |
|-----------------|-------------|
| sequence number | |
| ACK number | |
| A | F |
| checksum | |

- Client, server each close their side of connection.
 - send TCP segment with FIN bit = 1



Lectures 4&5: Summary

Go-back-N

- Sender can have up to N unACKed packets in pipeline
- Receiver only sends **cumulative ACKs**
 - Out-of-order packets discarded
- Sender sets timer for the oldest unACKed packet
 - when timer expires, retransmit **all** unACKed packets

Selective Repeat

- Sender can have up to N unACKed packets in pipeline
- Receiver sends **individual ACK** for each packet
 - Out-of-order packets buffered
- Sender maintains timer for **each** unACKed packet
 - when timer expires, retransmit only that unACKed packet

Lectures 4&5: Summary

- Connection-oriented transport: TCP

- Segment structure
- Reliable data transfer
- Setting and updating retransmission time interval
- 3-way handshake
- Fast retransmission

CS2105

An Aρεσομέ Introduction to Computer Networks

Lectures 6&7: The Network Layer



National University of Singapore
Department of Computer Science
School of Computing

Lectures 6&7: The Network Layer

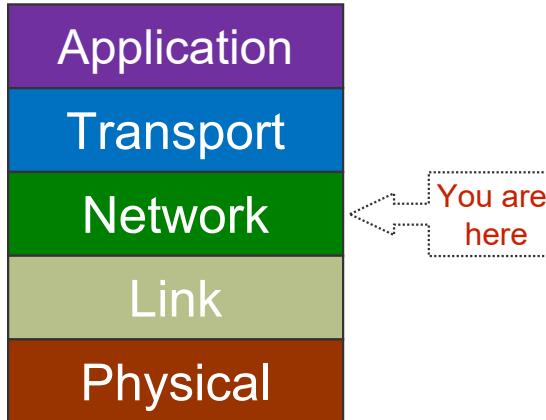
After this class, you are expected to understand:

- ❖ the basic services network layer provides.
- ❖ the purpose of DHCP and how it works.
- ❖ IP address, subnet, subnet mask and address allocation.
- ❖ how longest prefix forwarding in a router works.
- ❖ the purpose of routing protocols on the Internet.
- ❖ the principle of Bellman-Ford equation.
- ❖ the workings of distance vector algorithm.
- ❖ the purpose of NAT and how it works.
- ❖ the Internet Protocol (IP) and how datagram fragmentation works.

Lectures 6&7: Roadmap

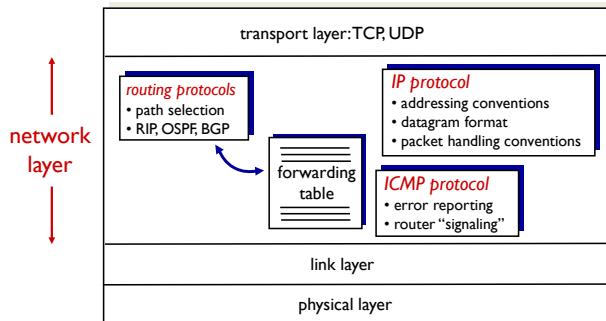
- 4.1 Overview of Network Layer
- 4.2 What's Inside a Router
- 4.3 The Internet Protocol (IP)
- 5.2 Routing Algorithms
- 5.6 ICMP

Kurose Textbook, Chapters 4&5
(Some slides are taken from the book)



Network Layer Services

- ❖ Network layer delivers packets to receiving hosts.
 - Routers examine header fields of IP datagrams passing it.



IP Address

- ❖ **IP address** is used to identify a host (or a router).
 - A 32-bit integer expressed in either binary or decimal

Binary: 00000001 00000010 00000011 10000001
 ↓ ↓ ↓ ↓
 1 2 3 129

Decimal: 1 2 3 129

- ❖ How does a host get an IP address?

- manually configured by system administrator, or
- automatically assigned by a **DHCP (Dynamic Host Configuration Protocol)** server.

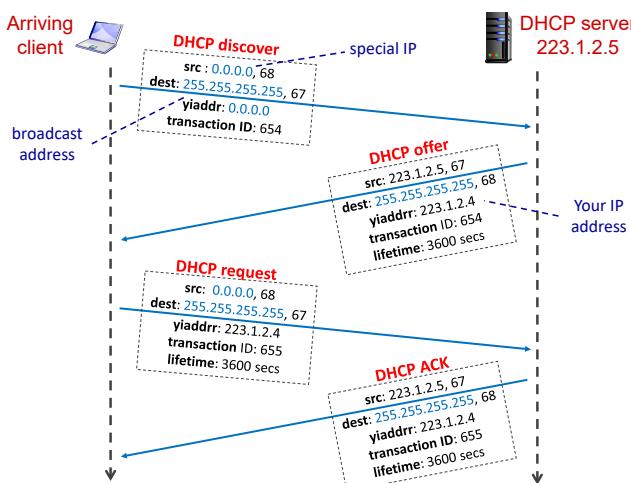
DHCP

- ❖ **DHCP** allows a host to dynamically obtain its IP address from DHCP server when it joins network.

- IP address is renewable
- allow reuse of addresses (only hold address while connected)
- support mobile users who want to join network.

- ❖ **DHCP**: 4-step process:

- 1) Host broadcasts “**DHCP discover**” message
- 2) DHCP server responds with “**DHCP offer**” message
- 3) Host requests IP address: “**DHCP request**” message
- 4) DHCP server sends address: “**DHCP ACK**” message



More on DHCP

- In addition to host IP address assignment, DHCP may also provide a host additional network information:
 - IP address of first-hop router
 - IP address of local DNS server
 - Network mask (indicating network prefix versus host ID of an IP address)
- DHCP runs over UDP
 - DHCP server port number: 67
 - DHCP client port number: 68

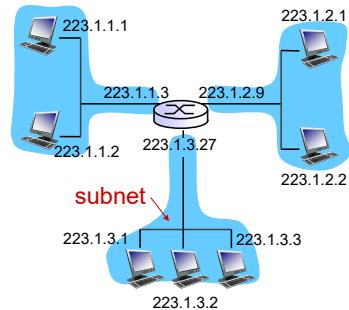
Some Special IP Addresses

| Special Addresses | Present Use |
|---|---|
| 0.0.0.8 | Non-routable meta-address for special use |
| 127.0.0.8 | Loopback address. A datagram sent to an address within this block loops back inside the host. This is ordinarily implemented using only 127.0.0.1/32. |
| 10.0.0.8 /12 172.16.0.0/12 192.168.0.0/16 | Private addresses, can be used without any coordination with IANA or an Internet registry. |
| 255.255.255.255/32 | Broadcast address. All hosts on the same subnet receive a datagram with such a destination address. |

The full list of special IP addresses can be found in RFC5735:
<https://tools.ietf.org/rfc/rfc5735.txt>

IP Address and Network Interface

- An IP address is associated with a **network interface**.
 - A host usually has one or two network interfaces (e.g. wired Ethernet and WiFi).
 - A router typically has multiple interfaces.



A network consisting of 3 subnets
(first 24 bits of IP addr. are network prefix)

IP Address and Subnet

- An IP address logically comprises two parts:

The diagram shows the structure of an IP address into network (subnet) prefix and host ID. The total width is 32 bits. The network prefix is n bits, and the host ID is 32 - n bits.
- Subnet** is a network formed by a group of “directly” interconnected hosts.
 - Hosts in the same subnet have the same network prefix of IP address.
 - Hosts in the same subnet can physically reach each other without intervening router.
 - They connect to the outside world through a router.

IP Address: CIDR

- The Internet’s IP address assignment strategy is known as **CIDR**.
 - Subnet prefix of IP address is of arbitrary length.
 - Address format: **a.b.c.d/x**, where x is the number of bits in subnet prefix of IP address.

11001000 00010111 00010000 00101010

this subnet contains 2^9 IP addresses
subnet prefix: 200.23.16.42/23

/23 indicates the no. of bits of subnet prefix

Subnet Mask

- Subnet mask** is used to determine which subnet an IP address belongs to.
 - made by setting all subnet prefix bits to "1"s and host ID bits to "0"s.
- Example: for IP address 200.23.16.42/23:

| IP address in binary | ← subnet prefix → | host ID → |
|-------------------------------------|-------------------------------------|---------------|
| 11001000 00010111 00010000 00101010 | | |
| Subnet mask | 11111111 11111111 11111110 00000000 | |
| Subnet mask in decimal | | 255.255.254.0 |

Quiz

- For the following 4 IP addresses, which one is in a different subnet from the rest 3?
 - 172.26.185.128/26
 - 172.26.185.130/26
 - 172.26.185.160/26
 - 172.26.185.192/26

IP Address Allocation

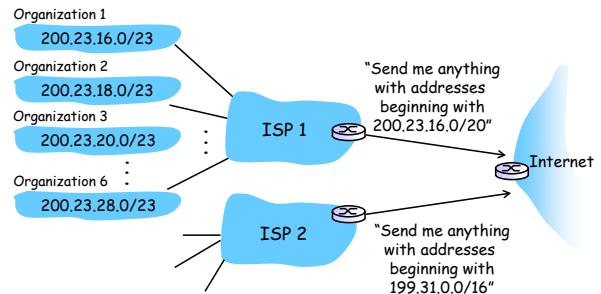
- ❖ **Q:** How does an organization obtain a block of IP addresses?
- ❖ **A:** Buy from registry or rent from ISP's address space.

| | Binary Address | Decimal Address |
|----------------|--|-----------------|
| ISP's block | 11001000 00010111 0001 0000 0 00000000 | 200.23.16.0/20 |
| Organization 1 | 11001000 00010111 0001 0000 0 00000000 | 200.23.16.0/23 |
| Organization 2 | 11001000 00010111 0001 0010 0 00000000 | 200.23.18.0/23 |
| Organization 3 | 11001000 00010111 0001 0100 0 00000000 | 200.23.20.0/23 |
| ... | ... | ... |
| Organization 6 | 11001000 00010111 0001 1010 0 00000000 | 200.23.28.0/23 |

use 3 more bits to differentiate 6 organizations

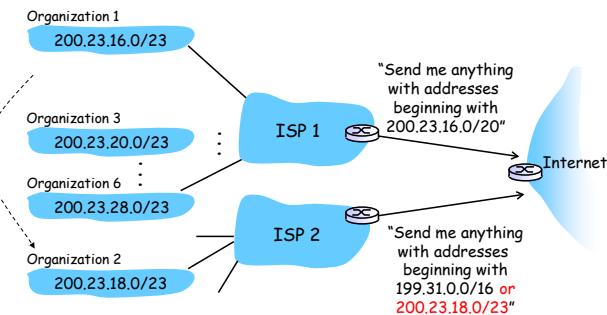
Hierarchical Addressing

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical Addressing

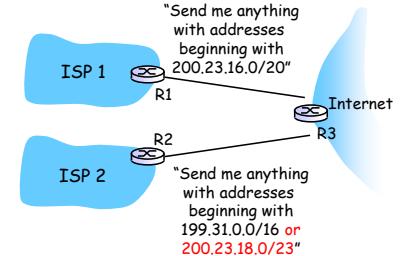
Suppose Organization 2 now switches to ISP 2, but doesn't want to renumber all of its routers and hosts.



Longest Prefix Match (1/2)

- ❖ **Question:** which router to deliver to,
 - if a packet has destination IP 200.23.20.2?
 - if a packet has destination IP 200.23.19.3?

| Forwarding Table at R3 | |
|------------------------|----------|
| Net mask | Next hop |
| 200.23.16.0/20 | R1 |
| 200.23.18.0/23 | R2 |
| 199.31.0.0/16 | R2 |
| ... | ... |



Longest Prefix Match (2/2)

- ❖ Packet with destination IP 200.23.20.2 → R1
 - (Binary: 11001000 00010111 00010100 00000010)
- ❖ Packet with destination IP 200.23.19.3 → R2
 - (Binary: 11001000 00010111 00010011 00000011)

| Forwarding Table at R3 | | |
|------------------------|-------------------------------------|----------|
| Net mask | Net mask in binary | Next hop |
| 200.23.16.0/20 | 11001000 00010111 00010000 00000000 | R1 |
| 200.23.18.0/23 | 11001000 00010111 00010010 00000000 | R2 |
| 199.31.0.0/16 | 11000111 00011111 00000000 00000000 | R2 |
| ... | ... | ... |

IP Address Allocation

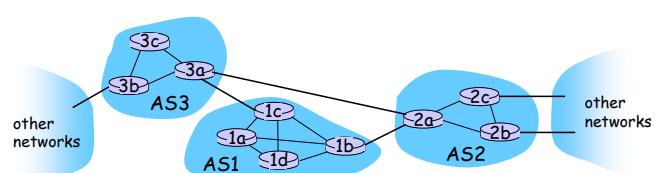
- ❖ **Q:** How does an ISP get a block of addresses?
- ❖ **A:** ICANN: Internet Corporation for Assigned Names and Numbers
 - Allocates addresses
 - Manages DNS
 - Assigns domain names, resolves disputes

Lectures 6&7: Roadmap

- 4.1 Overview of Network Layer
- 4.2 What's Inside a Router
- 4.3 The Internet Protocol (IP)
- 5.2 Routing Algorithms
 - 5.2.2 The Distance Vector Routing Algorithm
- 5.6 ICMP

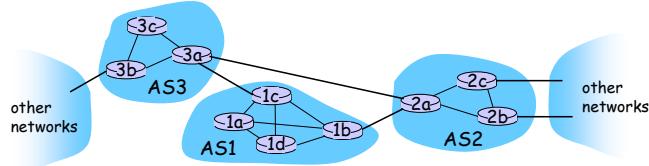
Internet: Network of Networks

- ❖ The Internet is a “network-of-networks”.
 - A hierarchy of Autonomous Systems (AS), e.g., ISPs, each owns routers and links.
- ❖ Due to the size of the Internet and the decentralized administration of the Internet, routing on the Internet is done hierarchically.

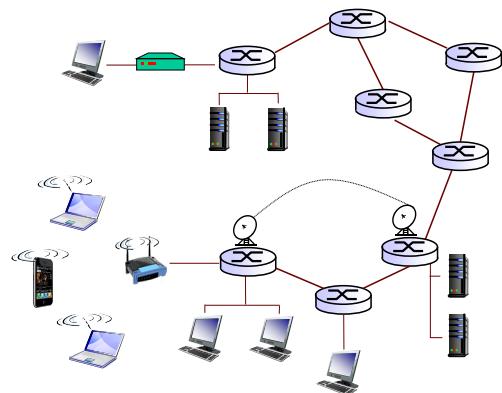


Routing in The Internet

- ❖ **Intra-AS routing**
 - Finds a good path between two routers within an AS.
 - Commonly used protocols: **RIP, OSPF**
- ❖ **Inter-AS routing (not covered)**
 - Handles the interfaces between ASes.
 - The de facto standard protocol: **BGP**

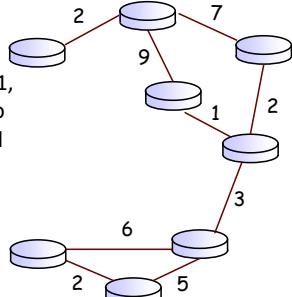


Abstract View of Intra-AS Routing



Abstract View of Intra-AS Routing

- ❖ We can associate a **cost** to each link.
 - cost could always be 1, or inversely related to bandwidth, or related to congestion.

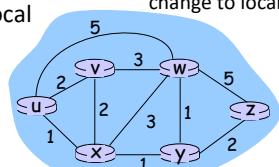


Routing: finding a least cost path between two vertices in a graph

Routing Algorithms Classification

"distance vector" algorithms

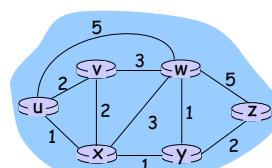
- ❖ Routers know physically-connected neighbors and link costs to neighbors.
- ❖ Routers exchange “local views” with neighbors and update own “local views” (based on neighbors’ view).
- ❖ Iterative process of computation
 1. Swap local view with direct neighbours.
 2. Update own’s local view.
 3. Repeat 1 - 2 till no more change to local view.



Routing Algorithms Classification

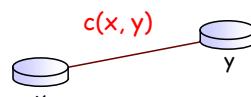
"link state" algorithms

- ❖ All routers have the complete knowledge of network topology and link cost.
 - Routers periodically broadcast link costs to each other.
- ❖ Use Dijkstra algorithm to compute least cost path locally (using global map).
 - Non-examinable ☺

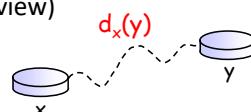


Some Graph Notations

- ❖ $c(x, y)$: the cost of link between routers x and y
 - $= \infty$ if x and y are not direct neighbours



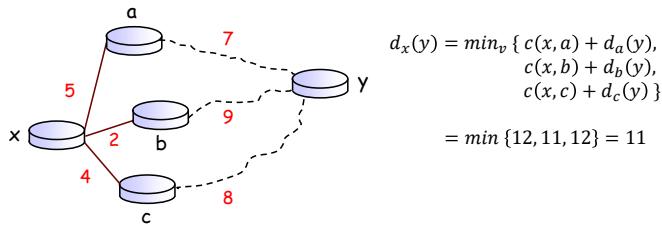
- ❖ $d_x(y)$: the cost of the least-cost path from x to y (from x ’s view)



Bellman-Ford Equation

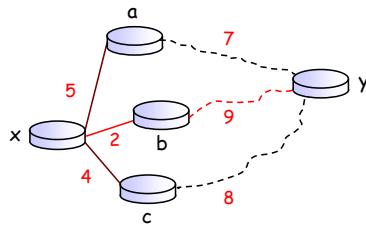
$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$$

where min is taken over all direct neighbors v of x



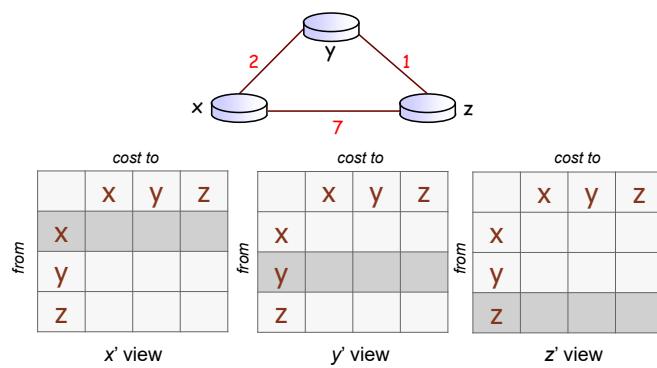
Bellman-Ford Equation

- To find the least cost path, x needs to know the cost from each of its direct neighbour to y.
- Each neighbour v sends its **distance vector** (y, k) to x, telling x that the cost from v to y is k.



Bellman-Ford Example

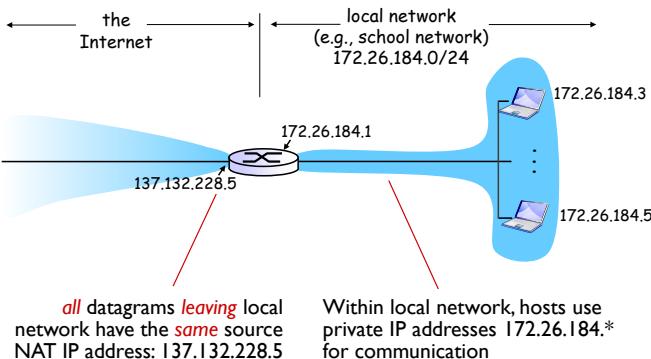
$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$$



RIP

- RIP (Routing Information Protocol) implements the DV algorithm. It uses **hop count** as the cost metric (i.e., insensitive to network congestion).
- Entries in the routing table are aggregated subnet masks (so we are routing to destination subnet).
- Exchange routing table every 30 seconds over UDP port 520.
- “Self-repair”: if no update from a neighbour router for 3 minutes, assume neighbour has failed.

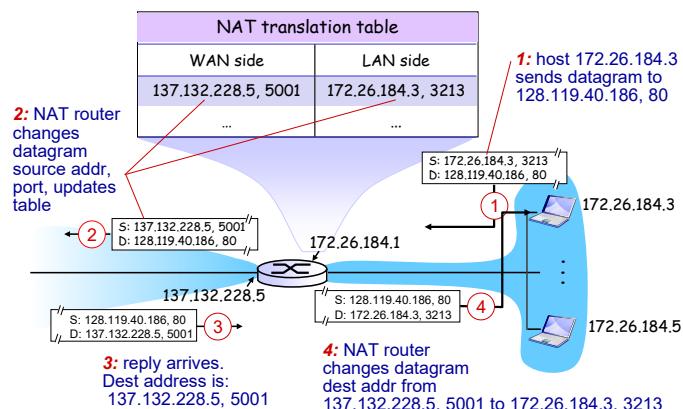
NAT: Network Address Translation



NAT: Implementation

- NAT routers must:
 - Replace (source IP address, port #) of every **outgoing datagram** to (NAT IP address, new port #).
 - Remember (in NAT translation table) the mapping from (source IP address, port #) to (NAT IP address, new port #).
 - Replace (NAT IP address, new port #) in destination fields of every **incoming datagram** with corresponding (source IP address, port #) stored in NAT translation table.

NAT: Illustration



NAT: Motivation and Benefits

- ❖ No need to rent a range of public IP addresses from ISP: just one public IP for the NAT router.
- ❖ All hosts use private IP addresses. Can change addresses of hosts in local network without notifying the outside world.
- ❖ Can change ISP without changing addresses of hosts in local network.
- ❖ Hosts inside local network are not explicitly addressable and visible by outside world (a security plus).

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

4.2 What's Inside a Router

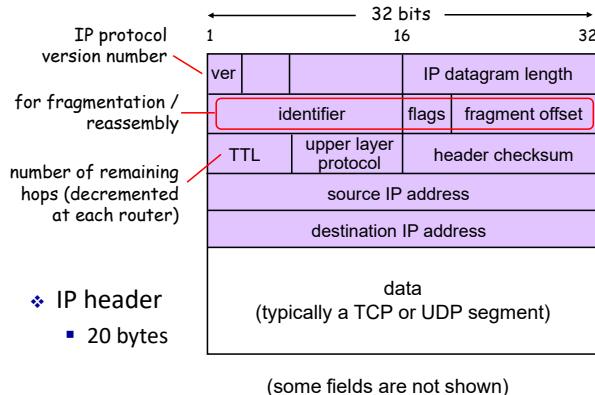
4.3 The Internet Protocol (IP)

- 4.3.1 IPv4 Datagram Format
- 4.3.2 IPv4 Datagram Fragmentation
- 4.3.5 IPv6 (non-examinable)

5.2 Routing Algorithms

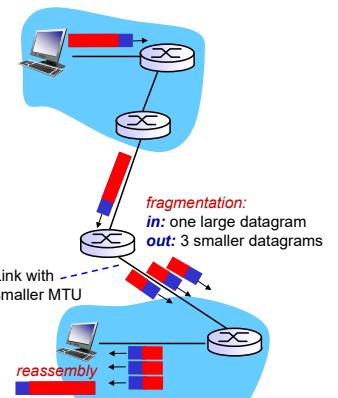
5.6 ICMP

IPv4 Datagram Format

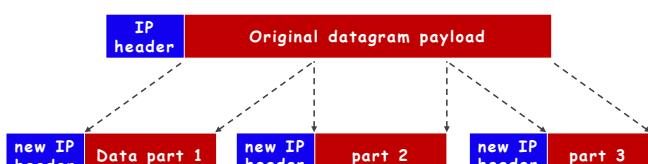


IP Fragmentation & Reassembly

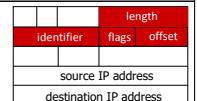
- ❖ Different links may have different **MTU (Max Transfer Unit)** – the maximum amount of data a link-level frame can carry.
- ❖ “Too large” IP datagrams may be fragmented by routers.



IP Fragmentation Illustration

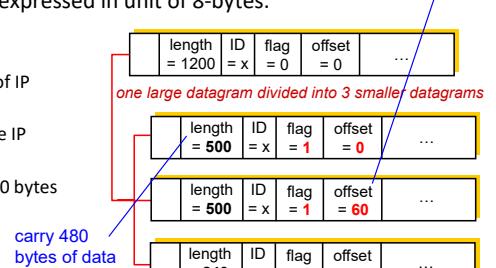


- ❖ Destination host will reassemble the packet.
- ❖ IP header fields are used to identify fragments and their relative order.



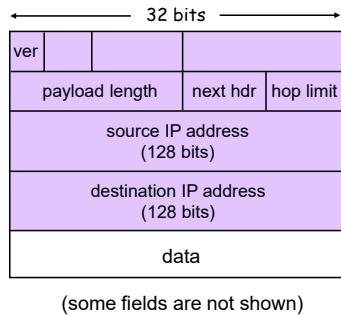
IP Fragmentation

- ❖ Flag (frag flag) is set to
 - 1 if there is next fragment from the same segment.
 - 0 if this is the last fragment.
- ❖ Offset is expressed in unit of 8-bytes.
- ❖ Example
 - 20 bytes of IP header
 - 1,200 byte IP datagram
 - MTU = 500 bytes



IPv6**Non-examinable**

- IPv6 is designed to replace IPv4.
- Primary motivation: 32-bit IPv4 address space is soon to be completely allocated.
- IPv6 datagram:
 - 40 byte header



Example IPv6 address (in hexadecimal):
2001:0db8:85a3:0042:1000:8a2e:0370:7334

Lectures 6&7: Roadmap

- 4.1 Overview of Network Layer**
- 4.2 What's Inside a Router**
- 4.3 The Internet Protocol (IP)**
- 5.2 Routing Algorithms**
- 5.6 ICMP** **Self-read**

ICMP**Self-read**

- ICMP (Internet Control Message Protocol)** is used by hosts & routers to communicate network-level information.
 - Error reporting: unreachable host / network / port / protocol
 - Echo request/reply (used by ping)
- ICMP messages are carried in IP datagrams.
 - ICMP header starts after IP header.

ICMP Type and Code**Self-read**

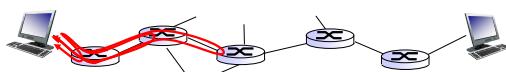
- ICMP header: Type + Code + Checksum + others.

| Type | Code | Description |
|------|------|-----------------------|
| 8 | 0 | echo request (ping) |
| 0 | 0 | echo reply (ping) |
| 3 | 1 | dest host unreachable |
| 3 | 3 | dest port unreachable |
| 11 | 0 | TTL expired |
| 12 | 0 | bad IP header |

Selected ICMP Type and subtype (Code)

ping and traceroute**Self-read**

- The command **ping** sees if a remote host will respond to us – do we have a connection?
- The command **traceroute** sends a series of small packets across a network, and attempts to display the route (or path) that the messages would take to get to a remote host.

**Lectures 6&7: Summary**

- An IP address is associated with a network interface. A device may have multiple network interfaces, thus multiple IP addresses.
- DHCP automates the assignment of IP addresses in an organization's network.
- On TCP/IP networks, subnets are defined as all devices whose IP addresses have the same network (subnet) prefix.
- Subnet mask is useful in checking if two hosts are on the same subnet.

Lectures 6&7: Summary

- Routing is the process of selecting best paths in a network.
- NAT** maps one IP addresses space into another.
 - Commonly used to hide an entire private IP address space behind a single public IP address.
 - NAT router uses stateful translation tables to remember the mapping.
- ICMP** is used by routers to send error messages.
 - E.g. when TTL is 0, a packet is discarded and an ICMP error message is sent to the datagram's source address.