Lecture 1

Internet & Network Edge (Normal user access networks)

- Internet is a network of connected devices known as **hosts** or **end systems.**
- Hosts access the internet through access networks.

Wireless Access Networks	
Wireless Lan	**Wide-area wireless access**
Within Building (100 ft)	10 KM
802.1 1b/g/n/ac (Wifi)	Cellular (3G, 4G)

Physical Media	
Guided Media (Solid Media)	**Unguided Media (Signals propagate freely)**
• Fiber optics , LAN • Twisted Pair Cable (RJ45) • Coaxial Cable (Use by StarHub)	• Wi-Fi • Cellular • Radio Channel

Network Core

Routing and Addressing

- Routers determine source - destination route taken by packets.
  - (Routing algorithms -> by looking at destination address)

How is data transmitted through internet?

Circuit Switching	**Packet Switching**

• Dedicated circuit per call
• All circuit needs to be reserved (e.g. telephone networks)
	• Data sent through internet in discrete “chunks”
	• 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 to the next link.

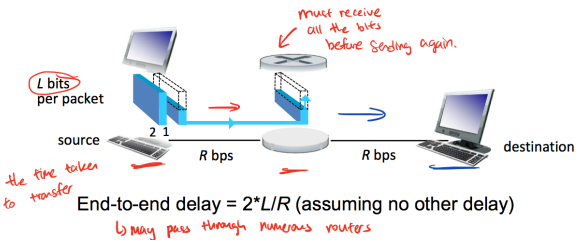
Pro’s & Con’s of Circuit Switching
• End-end resources allocated to and **reserved (cons of circuit switching)** for “call” between source & dest
  • call setup required
  • circuit-like **(guaranteed - pros of circuit switching)**
  • circuit segment idle if not used by call **(no sharing)**
  • commonly used in traditional telephone networks
  • divide link bandwidth into “pieces”
  • frequency divison
  • time division
	• Host break application message into smaller chunks, known as **packets**, of length **L** bits
	• Transmits **packets** onto **transmission rate R (the speed of transfer)**
	  • aka link capacity/link bandwidth
	• Packet transmission delay
	  = Time needed to transmit L-bit packet into link
	  = L (bits) / R (bits / sec)
	
	See below for example of End-to-end delay

Pro's & Cons of Packet Switching

- Internet is a packet switching network
- No need for resource reservation
- User A, B 's packets **share the same network resources**

(pros of packet switching)

- Resources are used on demand
- **Excessive congestion** is possible **(cons of packet switching)**



Delay, Loss and Throughput in networks

Packet Switching Network

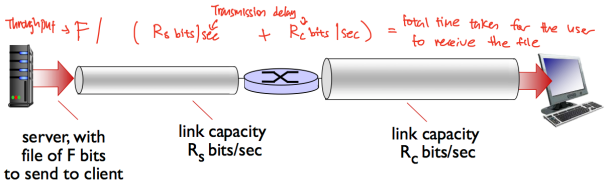
- To send a packet in packet switching network
- Routers determine source-destination route by packets.

To send a packet in packeting switching network :

1. Sender **transmit** a packet **onto** the **link** as a sequence of bits
2. Bits are **propagated** to the **next node** (e.g. a router) **on the link**
3. Router stores, processes and **forwards** the packet to the **next link**
4. Repeat step 2 & 3, till the packet received by the receiver

Throughput

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



How do **Delay** and **Loss** Occur?

End-to-end Packet Delay (Experience by users)

- End-to-end packet delay is the time taken for a packet to travel from source to destination.
  - **Transmission delay (L bits/ Transmission rate)**
    (depend on how fast is the **transmission rate**)
  - **Propagation delay (Length/ propagation speed)**
    (Usually very small, unless the **physical distance** is very long)
  - **Processing delay**
    (Depend on how **busy** the **network** is)
  - **Queuing delay (TUTORIAL 1 EXAMPLE)**
    (Depend on how **busy** the **network** is)

❖ 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}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Possibilities of packet loss

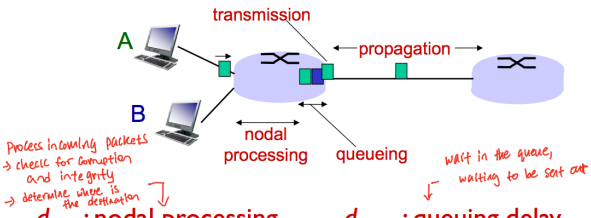
- Queue capacity is finite
  - If there are no place to store the packet
  - Router will drop the packet —> result in packet loss
  - **Packet loss** is been measured in term of probability
    (Not delay)

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

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

- **Protocols in Internet are organised into "layers"**
  - Each layer provide a service
  - Simple interface between layers
  - Hide details from each other
- **Benefits**
  - Allow identification, relationship of complex system's pieces
  - Modularisation eases maintenance, updating of system (will not affect the rest of system, when changing implementation of one layer)

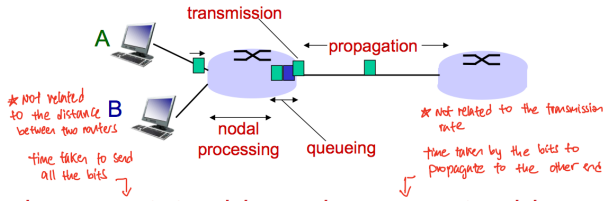
Four Sources of Packet Delay



d_{proc}: nodal processing
- check bit errors
- determine output link
- typically < msec

d_{queue}: queuing 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 (~2x10^8 m/sec)
- d_{prop} = *d/s*

d_{trans} and d_{prop} *very* different

Processing Delay	Occurs in the router when the router processes the packet (e.g., check bit errors or determine output link)
	In the milliseconds
Queuing Delay	Occurs in the router. Time spent waiting at output link for transmission. Dependant on queue size For this course, queuing delay is the time needed for packets before it to be transmitted $d_{queue} = \frac{nL + (L - x)}{R}$ where L is packet size, n is number of packets in front, and x is bytes currently being transmitted
	Depends on congestion level of router
Transmission Delay	Time taken to get the last byte of the packet into the link. Hence, if packet size is a lot bigger than link bandwidth, have to wait
	$d_{trans} = \frac{Packet\ Size}{Link\ Bandwidth}$
	Without Message Segmentation $t = \frac{File\ Size}{Link\ Bandwidth} \times no.\ of\ links$ Message Segmentation $t = time\ to\ one\ link \times no.\ if\ links + (no.\ of\ packets - 1) \times time\ to\ one\ link$ OR $t = time\ to\ first\ link \times (no.\ of\ packets + no.\ of\ links - 1)$ E.g., no. of packets = 4000, no. of links = 3 $t = L/R \times 4000 + 2$
Propagation Delay	Time taken for signal to travel from start to end of wire. Dependant on length of physical link
	$d_{prop} = \frac{Length\ of\ physical\ link}{Propagation\ Speed\ in\ medium}$
	$maximum\ bits\ in\ the\ link\ at\ given\ time = \min(bandwidth\ delay\ product, packet\ size)$ $bandwidth\ delay\ product = R \cdot d_{prop}$

Lecture 2 : Application Layer

Principles of Network Applications

What transport service does an app need?

Data Integrity
- apps such as file transfer or web transactions required **100% reliable** data transfer
- apps such as audio streaming can tolerate some data loss

Timing
- apps such as online games require low delay

Throughput
- **apps such as multimedia require minimum amount** of bandwidth to be **effective**
- apps such as file transfer make use of any throughput that is available

Security
- encryption, data integrity, authentication

Application layer Protocols

- **Messages exchanged**
  - request, response
- **Message Syntax**
  - which message fields & fields are delineated
- **Message semantics**
  - meaning of information in fields
- **Rules**
  - **When and how applications send & respond to messages**
- **Open protocols**
  - defined in relevant flow control
  - allows for interoperability
  - HTTP, SMTP
- **Proprietary protocols**
  - Skype

Possible structure of network applications :

Client - Server Architecture

Server :
- Waits for incoming requests
- Provides requested service to clients

Client :
- Initiate contact with sever
- Typically requests service from server

P2P Architecture

- Not always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers

Pros & Cons
- Highly scalable (Pros)
- Difficult to manage (Cons: user may join/leave dynamically)

Hybrid of Client - Server and P2P

- e.g. Instant messaging / facebook
- Chatting between two user is P2P
- Presence detection/location is centralised :
  - User registers its IP address with central server when it comes online
  - User find IP address of buddies by contacting the server

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

Internet Transport Protocols

TCP Service	UDP 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
	• **Unreliable data transfer**
	  • lost or corrupted data
	  • UDP will not resend
	• **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 or UDP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	TCP or UDP

Advantage of Message Segmentation

- Bits errors are **not tolerated** and if there is a single bit error, the whole message has to be **retransmitted**.
- When huge packets are send to the networks. Routers have to accommodate these huge packets and **smaller packets** have to queue behind and **suffer unfair delays**.

Disadvantages of Message Segmentation

- Packages have to be put in sequence at the destination. (Network may re-order packets)
- Message segmentation results in many smaller packets and each carry packet header of tens of bytes.

Web & HTTP

- Webpage consists of base html files & several referenced objects.
- Each object is addressable by URL ,e.g. **www.comp.nus.edu.sg/~cs2105/img/doge.jpg**
- www.comp.nus.edu.sg —> **host name**
- ~cs2105/img/doge.jpg —> **path name**

HTTP : Hypertext transfer protocol (HTTP 1.0 / HTTP 1.1)

- Web's application layer protocol
- Client/server model
  - Client : usually is browser that request, receives, and display web objects
  - Server : Web server sends object in response to request

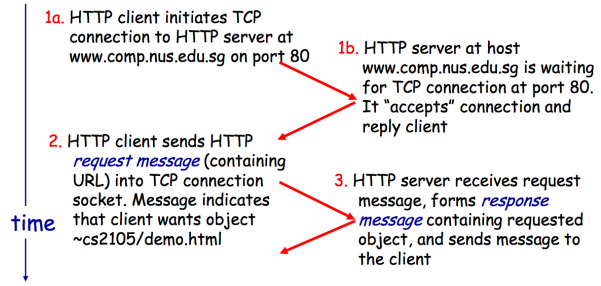
HTTP Over TCP

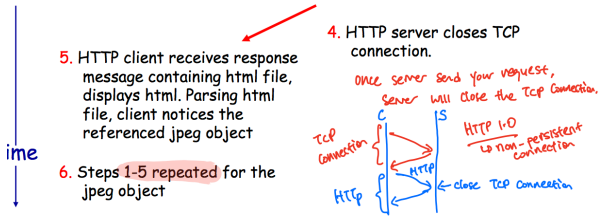
- **HTTP uses TCP** as transport service
  - Client initiate TCP connection to server.
  - Server accepts TCP connection request from client
  - HTTP messages are exchanged between browser (HTTP client) and Web server (HTTP server) over TCP connection
  - TCP connection closed.
  - **In another words, 3-ways handshakes**

Non-persistent HTTP Example

suppose user enters URL:
`www.comp.nus.edu.sg/~cs2105/demo.html`

(contains text, reference to a jpeg image)





- This is an example of *non-persistent connection* (http 1.0).

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
- client sends 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 /~cs2105/demo.html HTTP/1.1\r\n
Host: www.comp.nus.edu.sg\r\n
User-Agent: Mozilla/5.0\r\n
Connection: close\r\n
\r\n -> new line
      > break <br>
```

header lines

Extra "blank" line indicates the end of header lines

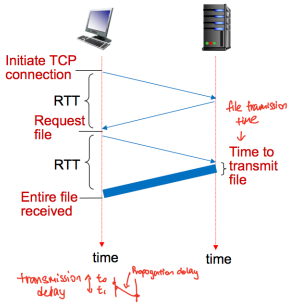
Non-persistent HTTP: Response Time

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 * RTT+ file transmission time



Example HTTP Response Message

status line (protocol status code) → `HTTP/1.1 200 OK`

header lines:

```
Date: Thu, 15 Jan 2015 13:02:41
  GMT
Server: Apache/2.4.6 (Unix)
Content-Type: text/html
\r\n
```

data, e.g., requested HTML file → `data data data data data ...`

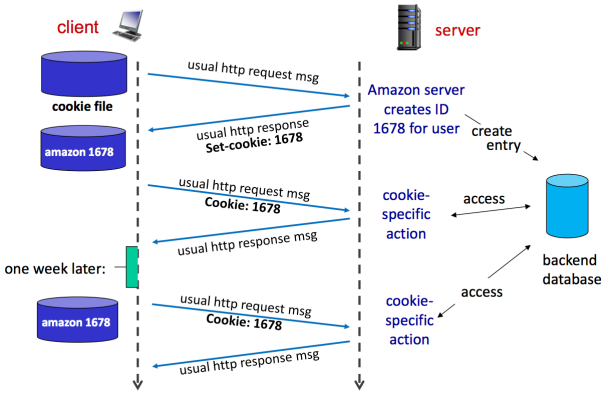
HTTP Response Status Code

- Status code appear in 1st line in server-to-client response message
- Some sample codes :
  - **200 OK**
    - Request succeeded, request object later in this message
  - **301 Moved Permanently**
    - Request object moved, new location specified later in this message (Location : …)
  - **403 Forbidden**
    - Sever declines to show the requested message
  - **404 Not Found**
    - Request document not found on this server
  - **400 Bad Request**
    - Request not supported by server or poor internet connection

Cookies

- HTTP is designed to be "**stateless**"
  - Server maintains no information about past client requests.
- We required such service to maintain states (history) at sever/client over multiple transactions
  - e.g. shopping carts, amazon suggestive purchase
- Cookie : http message carry "**state**"
  1. cookie header field of HTTP request / response messages
  2. cookie files kept on user's host, managed by user's browser
- Back-end database at Website

Keep User States with Cookie

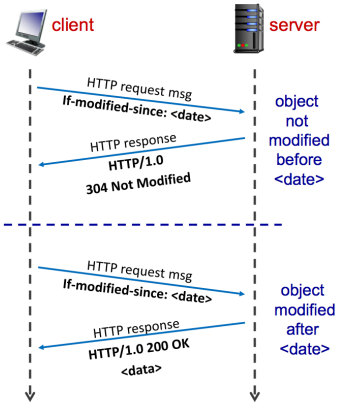


Conditional GET

- *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`



Domain Name System (DNS)

Two ways to identify a host :
- Hostname e.g. www.comp.nus.edu.sg
- IP Address e.g. 137.132.80.57

DNS translates between the two
- Client must carry out a **DNS query** to **determine the IP address** corresponding to the server name (Hostname) **prior to connection**

Root Server
- Answers request for records in the root zone by returning a list of the authoritative name servers for the appropriate top-level domain (TLD)

TLD Servers
- Responsible for com, org, net, edu, … and all top-level country domains, e.g. uk, sg , jp

Authoritative servers
- Organization own DNS server, providing authoritative hostname to IP mappings for organisation named hosts (e.g. Web, mail)
- Can be maintained by organization or service provider

nslookup
- Display the user server/IP address —> find out the true name of the IP address
  - public name : comp.nus.edu.sg
  - canonical (true/official) name : www0.comp.nus.edu.sg

DNS: Resource Records (RR)

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

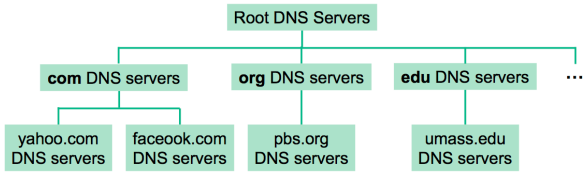
RR format: `(name, value, type, ttl)`

type = A
- `name` is hostname
- `value` is IP address

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

type = CNAME
- `name` is alias name for some "canonical" (the real) name
- `value` is canonical name
- `www.comp.nus.edu.sg` is really `www0.comp.nus.edu.sg`

type = MX
- `value` is name of mail server associated with `name`



A client want 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

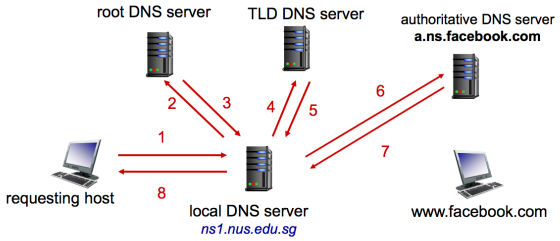
DNS cache poisoning

Rogue DNS records are introduced into a DNS resolver's cache causing the name server to return an incorrect IP address.
e.g. DDOS on a particular machine.

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 Query



- Once a name server learns mapping, it *caches* mapping
  - cache entries expire after some time (TTL).
- DNS runs over **UDP**.

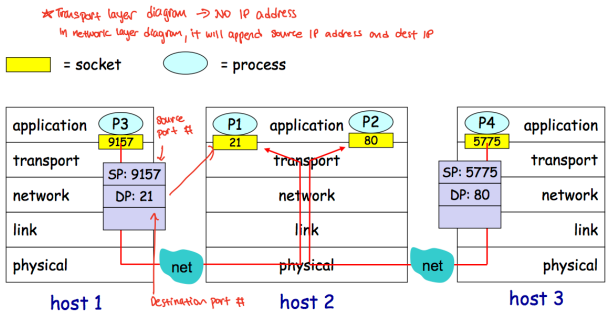
Lecture 3

Socket Programming with TCP and UDP

Processes

- Application run in hosts as processes.
  - Within the same host, two processes communicate using inter-process communication
  - Processes in different hosts communicate by exchanging messages.
  - **IP Address** is used to **identify a host device** but insufficient to identify a process running inside the host as many processes may run concurrently in a host
- **Protocol Service :**
  - Deliver packet to the right host : IP address of the host
  - Dispatch packet to the right process in the host : port number of the process

Transport Layer Diagram



use IP address + port number to locate a process

Given MSS is 1000 Bytes & file size of 9990 bytes

UDP will send the file as 1000bytes + 10bytes (header)
TCP will send the file as 1000bytes + 20 bytes(header)

Hence, the last segment for :
TCP - 1010 bytes
UDP - 910 bytes

- **Process** is **identified by** (**IP address, port number)**
  - Port number is 16-bit integer and unique for every process (1 - 1023 are reserved for standard use)
  - For e.g. HTTP Server : 80 , Mail Server : 25, SSH : 22
- **Socket** is the software interface between application processes and transport layer protocols
  - Process sends/receives message to/from its socket
  - Two types of sockets
    - Stream Socket **(TCP Socket)** that uses TCP (Reliable) as its transport layer protocol
      - Connection - oriented , reliable
    - Datagram Socket **(UDP Socket)** that uses UDP (Unreliable)
      - Connection-less, unreliable

TCP Socket

- TCP require **3 way handshake**
- Client must contact server
  - Server process must first be running if not exception will be thrown.
  - Sever must have created socket that waits for client's contact
- Server **use (Client IP + Port #)** to distinguish clients
- In TCP, two processes communicate as if there is a pipe between them. The pipe remains in place until one of the two process closes it.
  - When one of the processes want to send more bytes to other process, it simply write data to the pipe.
  - The sending process **does not** need to attach destination address and the port number to the bytes as it is already reliable

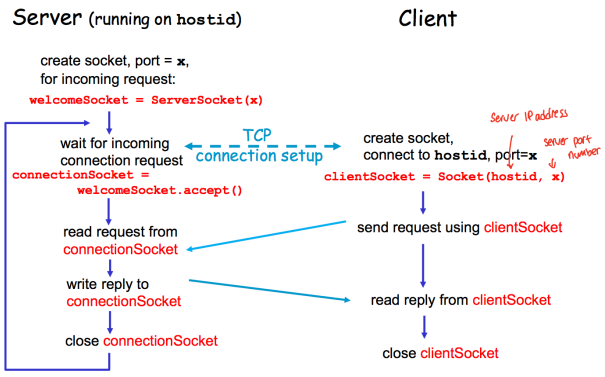
UDP Socket

- Client contacts server by
  - Creating client-local socket
  - Specifying IP address and port number of server processes to **each packet**
- Server **use one socket to serve all clients**
- **No connection is established before sending data**
- Transmitted data may be lost or received out-of-order

Socket Programming with TCP

- With TCP Sockets, a process establishes a connection to another process
- While the connection is in place, data flows between the process in continuous stream
- When contacted by client, TCP creates a new socket for server process to communicate with client

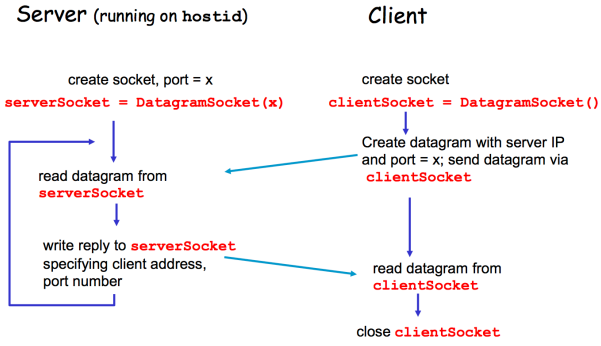
TCP: Client/server Socket Interaction



Socket Programming with UDP

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

UDP: Client/server Socket Interaction



Example: TCP Echo Server (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleTCPEchoServer {

    public static void main(String[] args) throws IOException {

        int port = 5678; // server listens to this example port

        // server is waiting
        ServerSocket welcomeSocket = new ServerSocket(port);

        while (true) {  // server is always alive
            Socket connectionSocket = welcomeSocket.accept();

       // to continue next page
```

This package defines Socket and ServerSocket classes

accept() method returns a new socket to communicate with client socket

```
            System.out.println("Connected to a client...");

            Scanner scanner = new
                Scanner(connectionSocket.getInputStream());
            // read data from the connection socket
            String fromClient = scanner.nextLine();

            PrintWriter toClient = new PrintWriter(
                    connectionSocket.getOutputStream(), true);

            // write data to the connection socket
            toClient.println(fromClient);
        }
    }
}
```

read from socket

text only

write to socket

telnet localhost 2005

end of while loop, loop back and wait for another client connection

Example: TCP Echo Client (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleTCPEchoClient {

    public static void main(String[] args) throws IOException {

        String serverIP = "127.0.0.1";  // local host, example
        int serverPort = 5678;          // just an example

        // create a client socket and connect to the server
        Socket clientSocket = new Socket(serverIP, serverPort);

        // read user input from keyboard
        Scanner scanner = new Scanner(System.in);
        String fromKeyboard = scanner.nextLine();

       // to continue next page
```

Example: TCP Echo Client (2/2)

```
        // create output stream to server
        PrintWriter toServer = new
             PrintWriter(clientSocket.getOutputStream(), true);
        // write user input to the socket
        toServer.println(fromKeyboard);

        // create input stream from server
        Scanner sc =
                 new Scanner(clientSocket.getInputStream());
        // read server reply from the socket
        String fromServer = sc.nextLine();

        // show on screen
        System.out.println("Echo from server: " + fromServer);

        clientSocket.close();
    }
}
```

Example: UDP Echo Server (1/2)

```
import java.io.*;
import java.net.*;

class SimpleUDPEchoServer {

    public static void main(String[] args) throws IOException {

        int port = 5678; // server listens to this example port
        DatagramSocket serverSocket = new DatagramSocket(port);

        byte[] rcvBuffer = new byte[1024];

        while (true) {  // server is always alive
            DatagramPacket rcvedPkt = new
                 DatagramPacket(rcvBuffer, rcvBuffer.length);

            serverSocket.receive(rcvedPkt);

       // to continue next page
```

receive() method blocks till a packet is received

Example: UDP Echo Server (2/2)

```
            String rcvedData = new String(rcvedPkt.getData(),
                                          0, rcvedPkt.getLength());

            InetAddress clientAddress = rcvedPkt.getAddress();

            int clientPort = rcvedPkt.getPort();

            byte[] sendData = rcvedData.getBytes();

            DatagramPacket sendPkt =
                 new DatagramPacket(sendData, sendData.length,
                                    clientAddress, clientPort);

            serverSocket.send(sendPkt);
        }
    }
}
```

extract client address and port number

end of while loop, loop back and wait for another client connection

Example: UDP Echo Client (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleUDPEchoClient {

    public static void main(String[] args) throws IOException {

        InetAddress serverAddress =     // server IP address
                 InetAddress.getByName("localhost");
        int serverPort = 5678;          // just an example

        // create a client socket
        DatagramSocket clientSocket = new DatagramSocket();

        // read user input from keyboard
        Scanner scanner = new Scanner(System.in);
        String fromKeyboard = scanner.nextLine();

       // to continue next page
```

translate hostname to IP address using DNS

create a client socket

Example: UDP Echo Client (2/2)

```
        // create a datagram and send to server
        byte[] sendData = fromKeyboard.getBytes();
        DatagramPacket sendPkt = new DatagramPacket(sendData,
                 sendData.length, serverAddress, serverPort);

        clientSocket.send(sendPkt);

        // receive a packet sent by server from socket
        byte[] rcvBuffer = new byte[1024];
        DatagramPacket rcvedPkt = new DatagramPacket(rcvBuffer,
                                                     rcvBuffer.length);
        clientSocket.receive(rcvedPkt);

        System.out.println("Echo from server: " +
                           new String(rcvedPkt.getData(), 0,
                                      rcvedPkt.getLength()));

        clientSocket.close();
    }
}
```

create a datagram to send

read a datagram from server

Lecture 4 UDP, Design Reliable Protocol

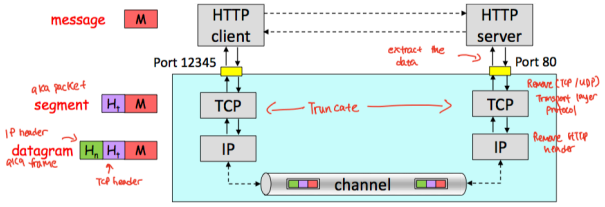
Transport Layer Service

Transport layer protocols run in hosts.

- **Sender side** : breaks app message into segments(packets), 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.

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 number.**



Connectionless Transport : UDP

- The job of delivering the data in a transport-layer segment to the correct socket is call **demultiplexing**.
- The job of gathering data chunks at the source host from different sockets, encapsulating each data chunks with header information to create segments, and passing the segments to the network layer is called **multiplexing**.

UDP : User Datagram Protocol

- UDP adds very little service on top if IP:
  - **Connectionless multiplexing/ 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 #.
    - **UDP receiver** receives a UDP segment:
      - Checks destination port # in segment
      - Directs UDP segment to the socket with the port #
      - IP datagrams (from different sources) with the same destination port # will be directed to the same UDP socket at destination

Sender :

- compute checksum value
- put checksum value into UDP checksum field

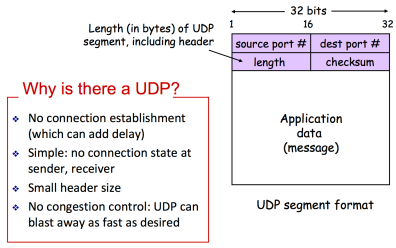
Receiver :

- compute checksum of received segment
- check if computed checksum equals checksum field value
  - No = error detected
  - Yes = No error detected

- **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**

Why UDP?

UDP Header



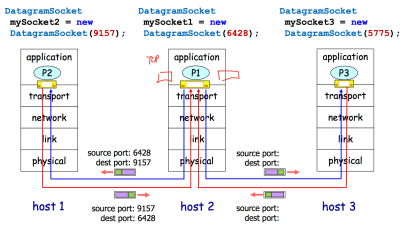
Why is there a UDP?

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

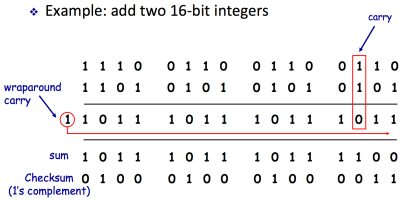
- **Checksum**
  - **Goal :** to detect erros (flipped bits) in transmitted segment
  - **How is UDP check sum computed?**
  1. Treat UDP segment as a sequence of 16-bit integer
  2. Apply binary addition on every 16-bit integer(checksum field is currently 0)
  3. Carry (if any) from the most significant bit will be added to the result
  4. Compute 1's complement to get UDP checksum

When will checksum failed?
If two bits is been corrupted at the same position, checksum will remain unchanged and receiver will fail to detect this error.

Why uses 1s complement scheme? How does the receiver detect errors?
Receiver could adds all the 16-bit words. If the sum contains a zero, there has been an error. All **one-bit errors** will be detected, but some two-bit errors can be undetected.



- Example: add two 16-bit integers



x	y	x ⊕ y	carry
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	1

Principles of Reliable Data Transfer

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

Reliable Transfer over Unreliable Channel

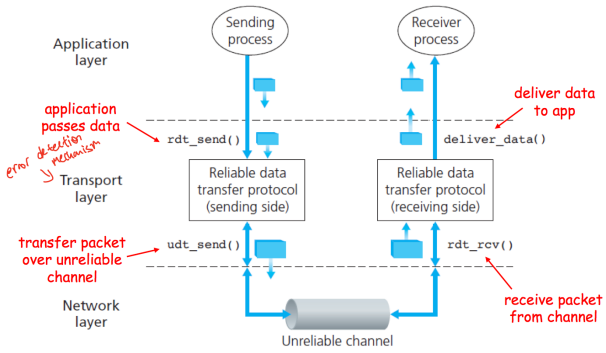
- Underlying network may
  - corrupt packets (1101 —> 1111)
  - drop packets (packet loss in the network)
  - re-order packets
  - delivery packets after an arbitrary **long delay**
- End-to-end reliable transport service should
  - guarantee packets delivery and correctness
  - delivery packets (to applicants) in the same order they are sent

Reliable Data Transfer Protocols

- Characteristics of unreliable channel will determine the complexity of reliable data transfer protocols (**rdt**)

Could implement reliability checking and recovery mechanisms at application layer

Reliable Data Transfer: Service Model



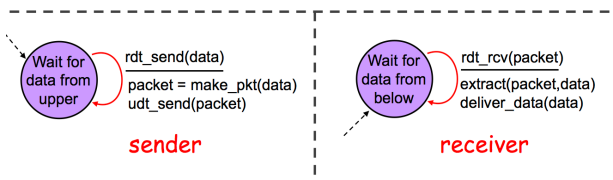
Finite State Machine (FSM) use to describe sender and receiver of a protocol

- **Event causing**
- **Action taken**

RDT 1.0

rdt 1.0 : Perfectly Reliable Channel

- **Assume** underlying channel is **perfect**
- Separate FSMs for sender, receiver :
  - Sender send data into underlying channel
  - Receiver read data from underlying channel



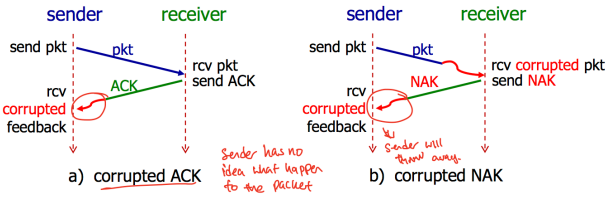
RDT 2.0

rdt 2.0 : Channel with Bit Errors

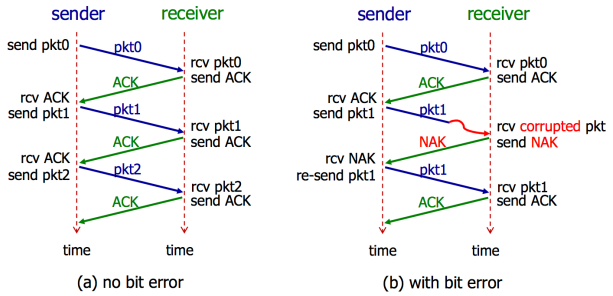
- **Assumption**
  - underlying channel may flip bits in packet
- Receiver may use **checksum** to detect bit errors
- How to **recover** from bits errors?
  - Acknowledgements (ACKs) : receiver tells sender that packet received is OK.
  - Negative acknowledgements (NAKs) : receiver tell sender that packet has errors so **sender** will **retransmit** the packet **on receipt of NAK**
- **Stop and wait protocol**
  - Sender sends one packet at a time, then waits for receiver response

What happen if ACK/NAK is corrupted?

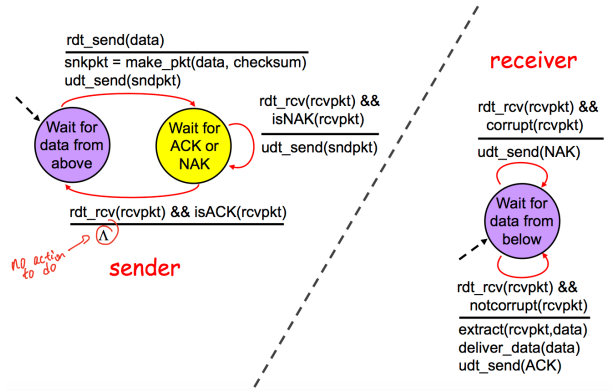
- Sender does not know what happen. So Sender should just retransmit.



rdt 2.0 In Action



rdt 2.0: FSM

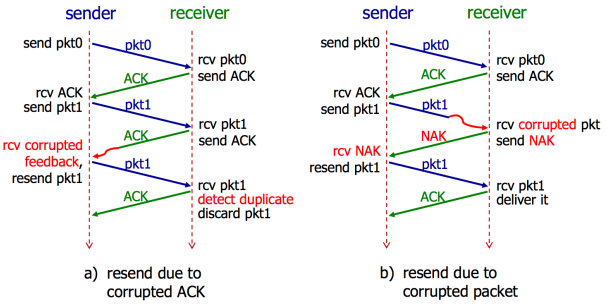


RDT 2.1

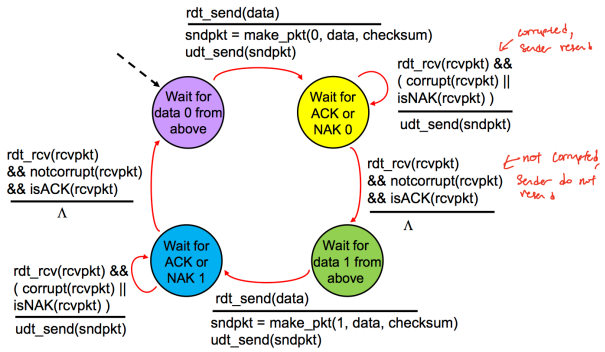
rdt 2.1 : rdt 2.0 + Packet seq #

- **To handle duplicates**
  - Sender transmits current packet if ACK/NAK is garbled
  - Sender add **sequence number** to each packet
  - Receiver discards duplicate packet

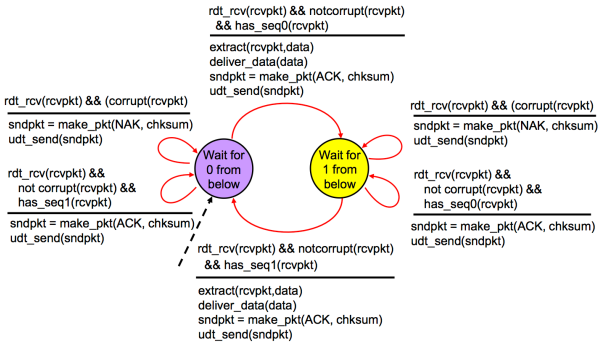
rdt 2.1 In Action



rdt 2.1 Sender FSM



rdt 2.1 Receiver FSM

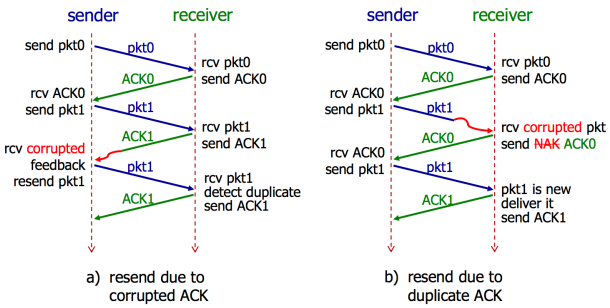


RDT 2.2

rdt 2.2 : a NAK-free protocol

- **Use ACKs only**
- Instead of sending NAK, receiver sends ACK for the last packet received OK.
  - Now receiver must include seq# of the packet being ACK
  - **Duplicate ACKs** at sender results in the same action as **NAK**

rdt 2.2 In Action

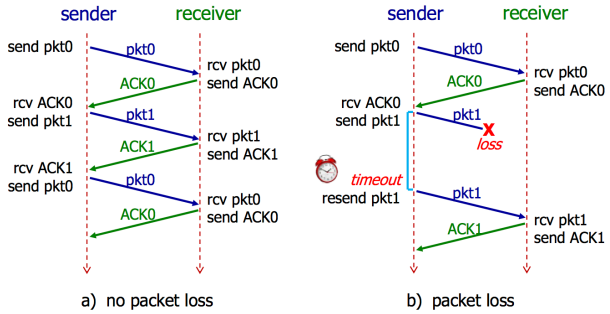


RDT 3.0

rdt 3.0 : Channel with Errors and Loss

- **Assumption : underlying channel**
  - may flip bits in packet
  - may lose packets
  - may incur long packet delay
  - but **won't** reorder packet
- **To handle packet loss:**
  - Sender waits "reasonable" amount of time for ACK
  - Sender retransmit if no ACK is received till timeout
  - Even when it is **delayed** but **not lost**, Timeout will still be **triggered** for **retransmission**
  - Retransmission will generate duplicates in the case but receiver may use seq # to detect it
  - Receiver must specify seq # of packet being ACKed.

rdt 3.0 In Action



Performance of rdt3.0

- rdt3.0 works, but performance stinks.
- Example: packet size = 8000 bits, link rate = 1 Gbps:

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

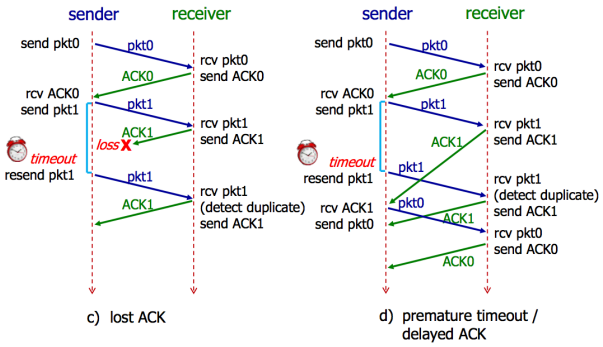
  - If RTT = 30 msec, sender sends 8000 bits every 30.008 msec.

$$throughput = \frac{L}{RTT + d_{trans}} = \frac{8000}{30.008} = 267 \text{ kbps}$$

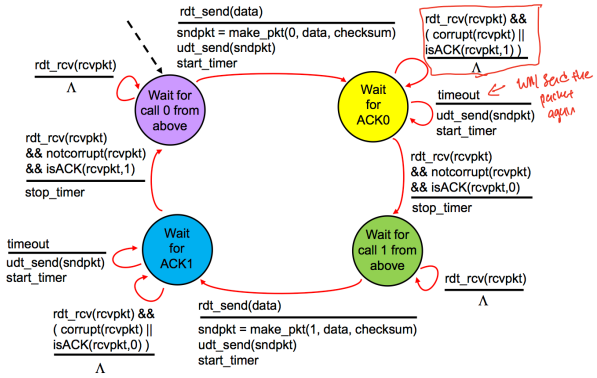
  - U_{sender}: *utilization* – fraction of time sender is busy sending

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

rdt 3.0 In Action

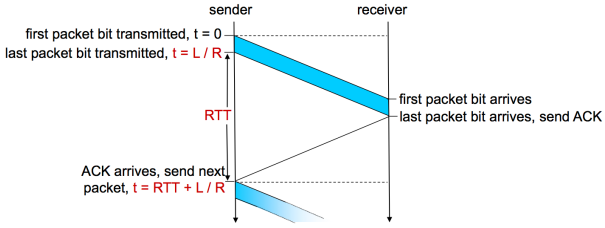


rdt 3.0 Sender FSM



rdt3.0: Stop-and-wait Operation

- Network protocol limits use of physical resources!



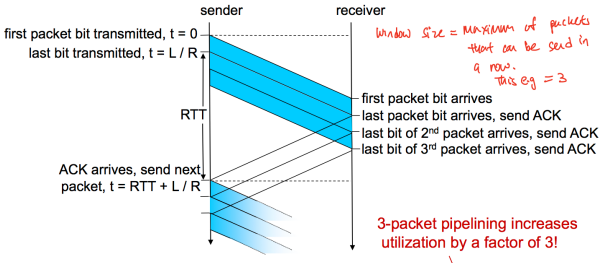
$$U_{sender} = \frac{L / R}{RTT + L/R} = \frac{0.008}{30 + 0.008} = 0.00027$$

Use pipelining could increase the performance
Two generic forms of pipelined protocols :

- Go-Back-N
- Selective repeat

Defence Mechanism	Problem	rdt1.0	rdt2.0	rdt2.1/2.2	rdt3.0
Timeout & retransmission	Packet Delay				Y
Sequence Number	Duplicate data			Y (1s & 0s)	Y (1s & 0s)
Acknowledgement	Packet Loss/Corruption		Y	Y	Y
Checksum	Corruption		Y	Y	Y

Pipelining: Increased Utilization



$$U_{sender} = \frac{3 * L / R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.00081$$

rdt Version	Scenario	Features Used
1.0	no error	nothing
2.0	data Bit Error	checksum, ACK/NAK
2.1	data Bit Error ACK/NAK Bit Error	checksum, ACK/NAK, Sequence Number
2.2	Same as 2.1	NAK free
3.0	data Bit Error ACK/NAK Bit Error packet Loss	checksum, ACK/NAK, sequence Number, timeout/re-transmisson

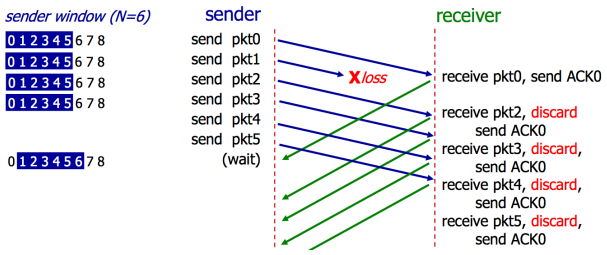
Dig program : dig-t a www.abc.com

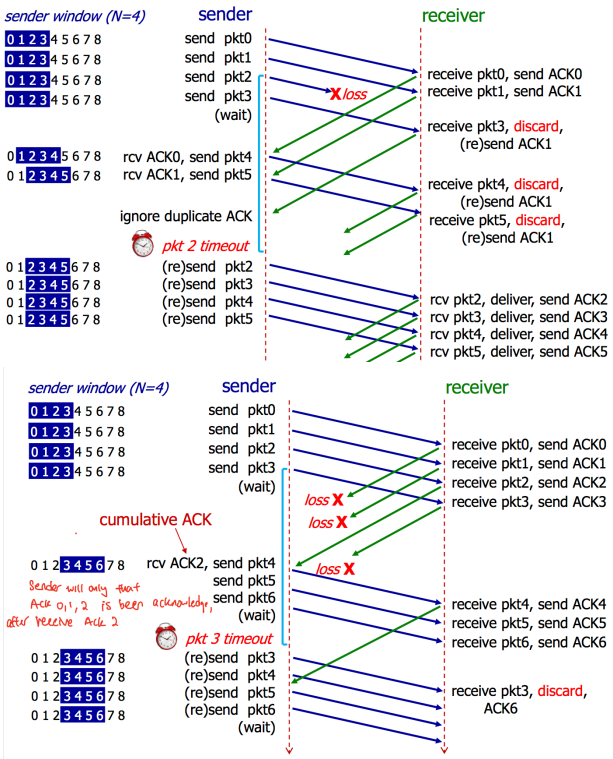
If IP address of the web page has been queried by another computer, local DNS should keep the knowledge and answers the query quickly, if not it will be very long.

Pipelining Protocols

Go-Back-N : Key Features

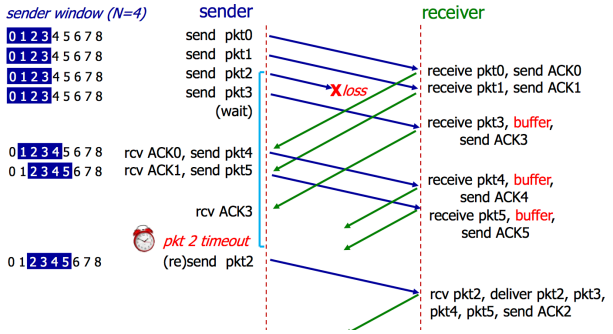
- GBN Sender
  - can have up to **N (window size)** unACKed packets in pipeline
  - insert k-bits sequence number in packet header
  - use a "**sliding window**" to keep track of unACKed packets
  - keep a **timer** for the **oldest** unACKed packet.
  - **timeout(n)** : **retransmit** the packet n and all subsequent packets in the window
- GBN Receiver
  - only ACK packets that arrive in order
    - only need to remember expected SeqNum
  - discard out-of-order packets and ACK the last-in-order seq #
    - **cumulative ACK :** ACK "m" means all packets up to m are received.





Selective Repeat : Key Features

- **Sender** maintains time for each unACKed packet
  - if next available seq # in window, send packet
  - **Timeout(n)**
    - when timer expires, retransmit only that unACKed packet , 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** individually ACK all correctly received packets
  - **pkt(n) in [rcvbase, rcvbase + N - 1]**
    - send ACK(n)
    - out-of-order : buffer (save)
    - in-order deliver (also deliver buffered, in-order pkts), advance window to next not-yet received pets
  - **pkt(n) in [rcvbase - N, rcvbase + 1]**
    - ACK(n)
  - otherwise
    - Ignore



Lecture 4: 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

Lecture 5 : Connection-oriented transport : TCP

In contrast to UDP, **TCP** is **complex** and is describe in tens of RFCs, with new mechanisms or tweaks introduced throughout the years.

Connection-Oriented

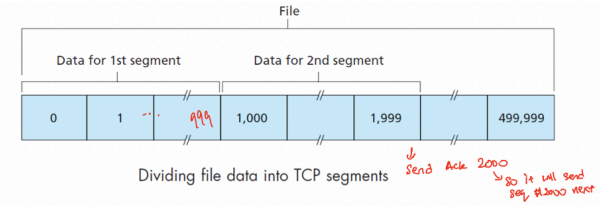
- handshaking (exchanging of control message) before sending app data

Reliable, in-order byte steam

- Application passes data to **TCP** and TCP form packets in view of **MSS** (maximum segment size) < 1500 bytes
  (TCP header <= 20 bytes)
- For **UDP**, app forms packets : **DatagramPacket**)

TCP sequence number

- TCP sequence number : "byte number" of the first byte of data in a segment



- Seq. # of 1st TCP segment: 0, 2nd TCP segment: 1000, 3rd TCP segment: 2000, 4th TCP segment: 3000, etc.

TCP Sender Events (simplified)

```
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (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;

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

        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;

} /* end of loop forever */
```

Sender keep one timer only

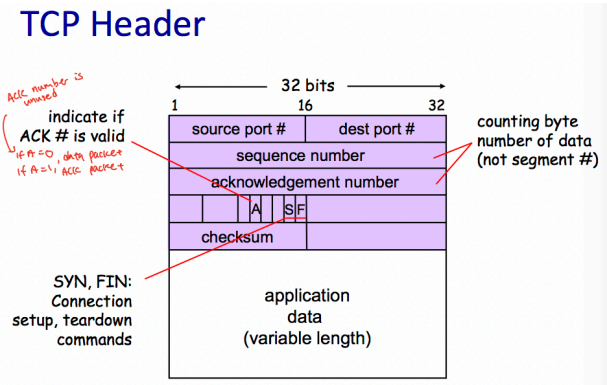
Retransmit only oldest unACKed packet

Cumulative ACK

Connection-oriented De-mux

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

TCP Header

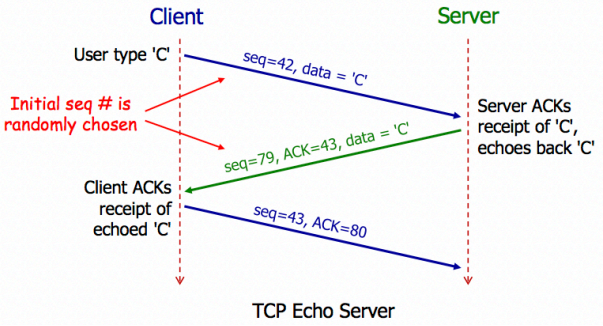


TCP Ack Number

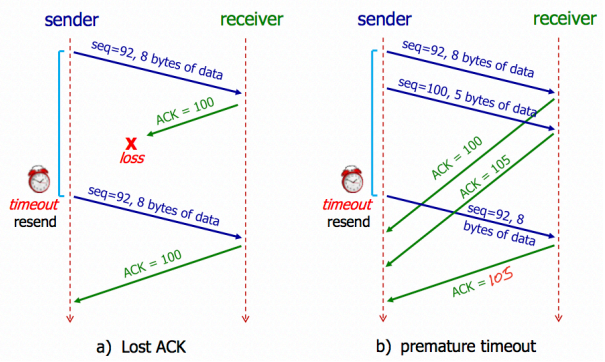
- Acknowledgement number
  - seq # of the next byte of data expected by receiver
  - TCP ACKs up to first missing byte in the stream (cumulative ACK)
  - **ACK (m) == receiver receive up to m - 1**

TCP Echo Server

- TCP (and also UDP) is a full duplex protocol
  - bi-directional data flow in the same TCP connection
  - Initial seq# is a offset to every packet



TCP Timeout / Retransmission



TCP Receiver Events

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 formed and detected)	Immediately send *duplicate ACK*, indicating seq. # of next expected byte (cumulative ACK)
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap

TCP Timeout Value

How does a TCP set appropriate timeout value?

- **too short timeout** : premature timeout and unnecessary retransmission
- **too long timeout** : slow reaction to segment loss
- **Timeout interval** must be longer than Round trip time (RTT)

TCP Fast Retransmission

- Timeout period is often relatively long
- If sender receives **3 duplicate ACKs** for the same data, it supposes that **segment after ACKed data** is **lost**.
- Action : **resend segment (even before timer expires)**

TCP Timeout Value

- TCP computes (and keeps updating) timeout interval based on estimated RTT.

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
```

(typical value of α : 0.125)

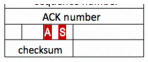
```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

(typical value of β : 0.25)

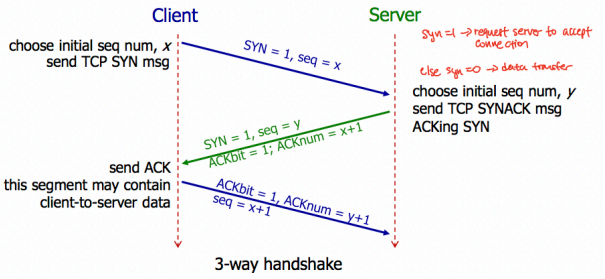
```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

"safety margin"

Establishing Connection

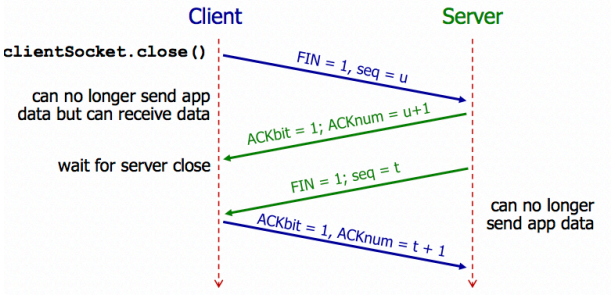


- Before exchanging data, TCP sender and receiver "shake hands".
  - Agree on connection and exchange connection parameters.



Closing Connection

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

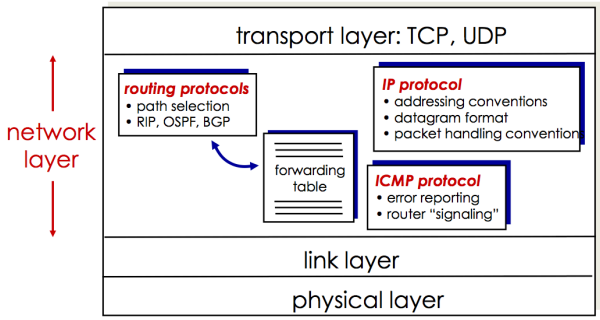


CS2105 Lecture 6

Protocols Recaps

rdt Version	Scenario	Features Used
1.0	no error	nothing
2.1	data packet Bit Error feedback packet Bit Error	checksum, ACK/NAK, sequence number
2.2	data packet Bit Error feedback packet Bit Error	checksum, ACK, sequence number
3.0	data packet Bit Error feedback packet Bit Error packet Loss	checksum, ACK, sequence number, timeout/re-transmission

	Go-Back-N	Selective Repeat
Out-of-order packets	ignore	buffer
ACK	cumulative	individual
Timer	earliest unACKed	one per unACKed packet
Retransmission	all unACKed	one unACKed



IP Address

- IP address is used to identify a host (or router)
- How does a host get an IP address?
  - Manually configured by system administrator
  - Assigned by DHCP server

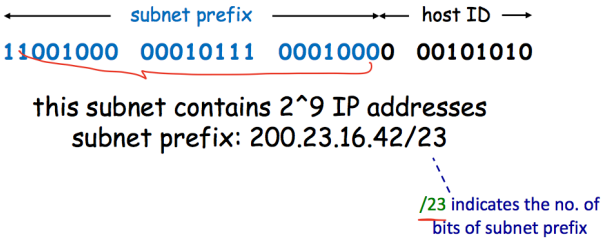
IP Address, Subnet & Network Interface

An IP address is associated with a **network interface.**

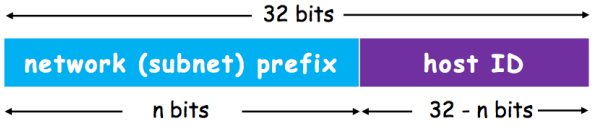
- Host usually have one or two network interfaces (wired ethernet / wifi)
- Router has multiple interfaces

IP Address : CIDR (Classless Inter-domain Routing)

- Subnet prefix of IP address if of arbitrary length
- Address format : a.b.c.d/x , where x is the number of bits in subnet prefix of IP address



An IP address logically comprises two parts:

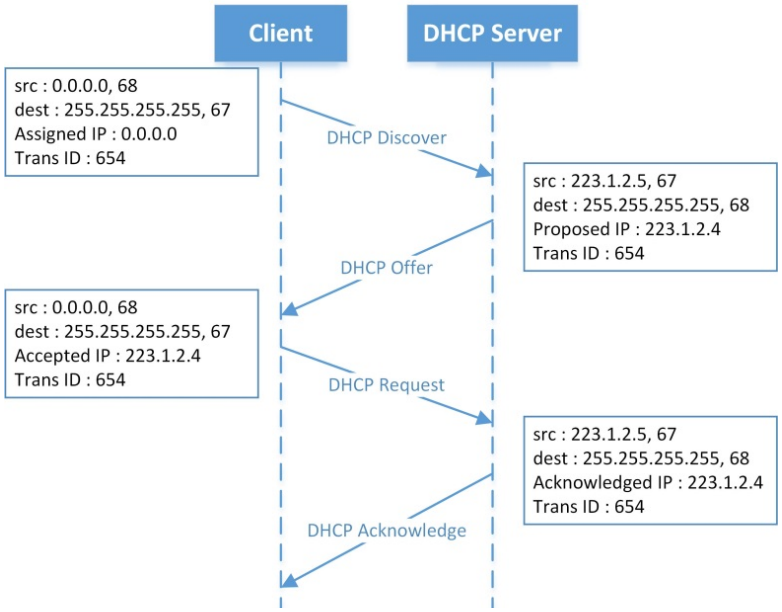


Subnet is formed by group of "directly" interconnected hosts.

1) Hosts in the same subnet have the same network prefix of IP address.
2) Hosts in the same subnet can physically reach other without intervening router.
3) They connect to the "outside world" through router

Subnet mask is used to determine what subnet IP address belongs to.

DHCP (Dynamic Host Configuration Protocol)



Discovery
DHCP server Port *always* **68**
DHCP client Port *always* **67**
Source IP 0.0.0.0 is special for DHCP
Dest IP 255.255.255.255 is broadcast addr

Offer
≥ one DHCP server may reply to client

Request
Client will broadcast their accepted IP

Acknowledge
Owning DHCP server will acknowledge

DHCP allows a host to dynamically obtains its IP address from DHCP server when it joins.

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

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

DHCP may provide a host **additional network information** :

1) IP address of first-hop router
2) IP address of local DNS server
3) Network mask (network prefix vs host ID of IP address)

DHCP runs over **UDP (Fast)**

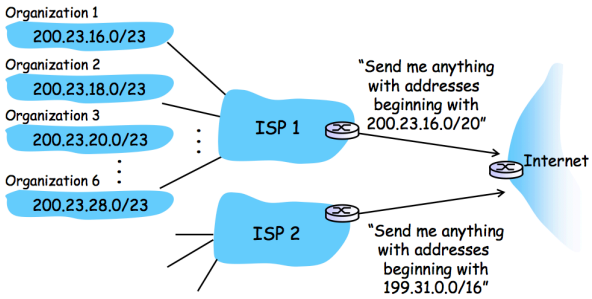
1) DHCP **server** port number : **67**
2) DHCP **client** port number : **68**

Some Special IP Addresses

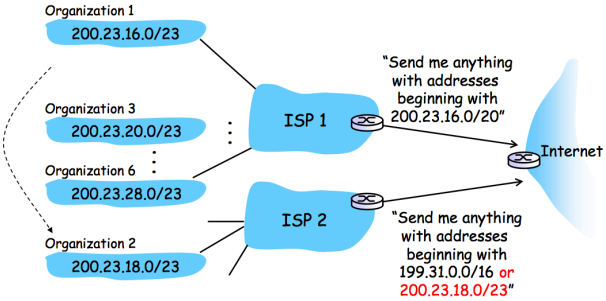
Special Addresses	Present Use
0.0.0.0/8	Non-routable meta-address for special use
127.0.0.0/8	Loopback address. A datagram sent to an address within this block loops back inside the host. This is ordinarily implemented using only 127.0.0.1/32.
10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	Private addresses, 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.

IP Address Allocation & Hierarchical Addressing

Hierarchical addressing allows efficient advertisement of routing information:



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

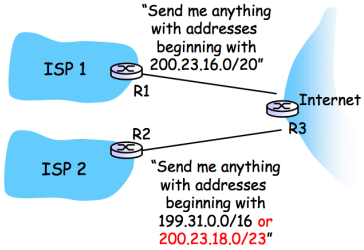


Longest Prefix Match (1/2)

- Question: which router to deliver to,
  - if a packet has destination IP 200.23.20.2? ISP 1
  - if a packet has destination IP 200.23.19.3? ISP 2

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 ⇨
  - (Binary: 11001000 00010111 00010100 00000010)
- Packet with destination IP 200.23.19.3 ⇨
  - (Binary: 11001000 00010111 00010011 00000011)

Forwarding Table at R3

Match with all netmask in the forwarding Table

match the longest prefix

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

An organisation could **obtain a block of IP addresses** by buying from registry or rent from ISP's address spacing.

ISP could get a block of addresses from ICANN (Internet Corporation for Assigned Names and Numbers)

1) Allocates Addresses
2) Manages DNS
3) Assigns domain names, resolve disputes

CS2105 Lecture 7 : IP & Routing

Intra-AS routing (routing among the organisation)
- Finds a good path between two routers within an AS
- Commonly used protocols : **RIP, OSPF**
- Single admin (no policy decisions is needed)
- Routing mostly focus on performance (find a least cost path between two vertices in a graph)

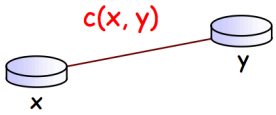
Distance Vector Algorithm

- Routers know physically-connected neighbours and link costs to neighbours
- **Routers** exchange "local views" with neighbours and **update own "local views"** based on neighbour's view

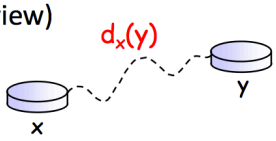
- Iterative process of computation

1) Swap local view with direct neighbours
2) Update own's local view
3) Repeat 1-2 till no more change to local view

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



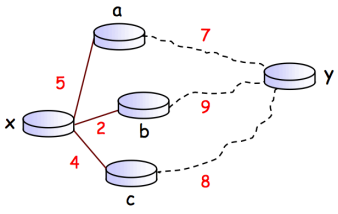
$d_x(y)$: the cost of the least-cost path from x to y (from x's view)



Bellman-ford equation

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

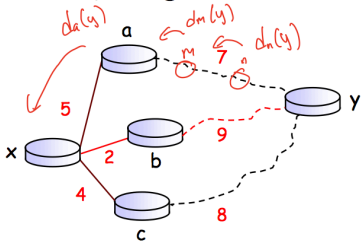
where min is taken over all direct neighbors v of x



$$d_x(y) = min_v \{ c(x,a) + d_a(y), \; c(x,b) + d_b(y), \; c(x,c) + d_c(y) \}$$

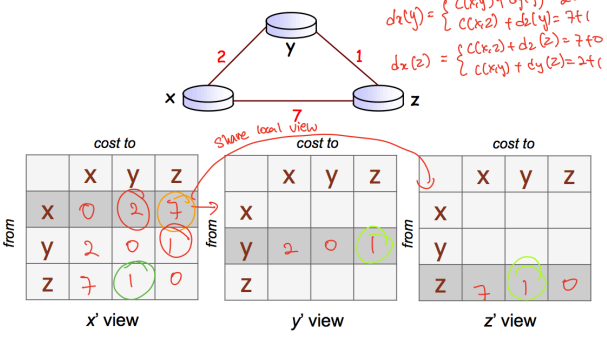
$$= min\{12, 11, 12\} = 11$$

- 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.



Now x knows, to reach y, packet should be forward to b and the total cost would be 11.

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

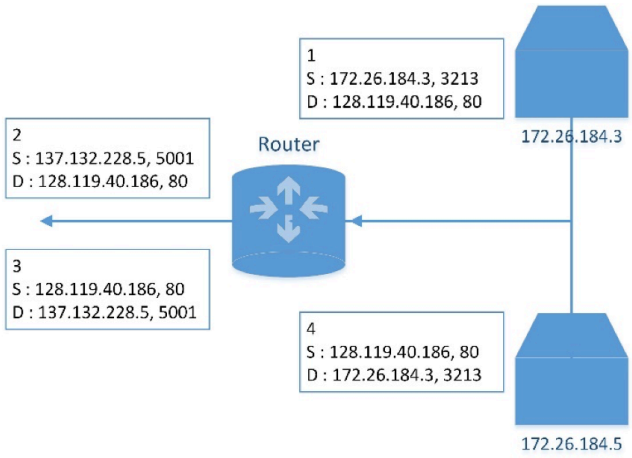


- Each router x,y,z **sends its distance** vectors to its directly connected neighbours
- When x finds out that y is advertising a path to z that is cheaper than x currently knows
  - **x updates its distance vector to z** accordingly
  - x will **note down that all packets for z** should be sent to y. This **info** will be **used** to **create forwarding table of x**
- After every routers has **exchanged several rounds of updates** with its direct neighbours, all **routers will know the least-cost paths to all other routers.**

Routing Protocols: Routing Information Protocol (RIP), UDP Port 520

- Uses hop count as cost metric, i.e., will take shorter path even if it's highly congested
- Entries in routing table of each router are the subnet masks (e.g., 255.255.254.0/23)
- RIP runs on UDP faster, don't need 3-way handshake as in TCP
- Routing table exchanged every 30s over **UDP port 520**
- If no update from neighbour router for 3min, remove neighbour from routing table

NAT : Network Address Translation



All datagrams **leaving local network** have the same source NAT IP address : **137.132.228.5**

Within local network, hosts uses private IP address to communicate : **172.26.184.***

1. Outgoing
REPLACE source IP & port ➔ NAT IP & new port

2. Outgoing
REMEMBER mapping in translation table

NAT translation table	
WAN side	LAN side
137.132.228.5, 5001	172.26.184.3, 3213

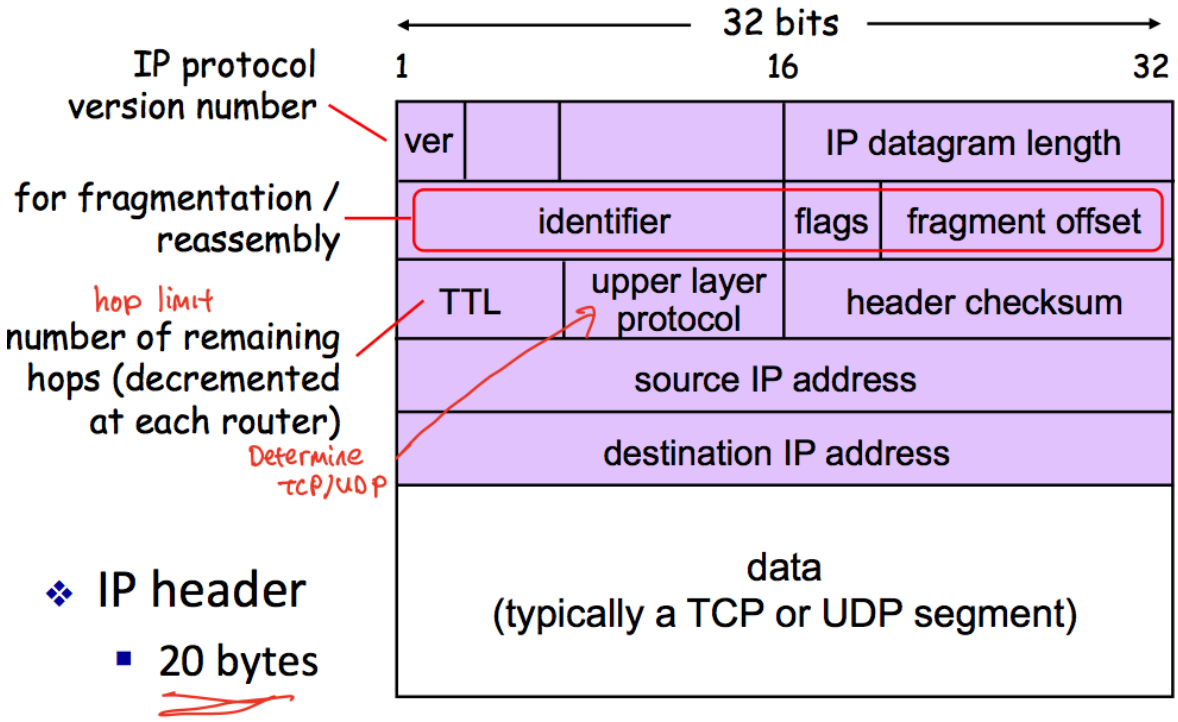
3. Incoming
REMEMBER mapping from translation table

4. **Incoming**
REPLACE NAT IP & new port ➔ source IP and port

Motivation and Benefits

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

IPV4 Datagram Format



- IP header
  - 20 bytes

IP Fragmentation & Reassembly

- 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 in expressed in unit of 8-bytes.
- Example
  - 1200 byte IP datagram
  - MTU = 500 bytes

carry 480 bytes of data

offset = 480/8

use to identify same packet (never change)

length	ID	flag	offset	…
= 1200	= x	= 0	= 0	

one large datagram divided into 3 smaller datagrams

length	ID	flag	offset	…
= **500**	= x	= **1**	= **0**	
= **500**	= x	= **1**	= **60**	
= **240**	= x	= **0**	= **120**	

ICMP (Internet Control Message Protocol)

- Used by hosts & routers to communicate network-level information
  - **Error reporting** : unreachable host / network/ port/ protocol
  - Echo request/ reply **(used by ping)**
  - **ICMP** messages are carried in **IP datagrams** (ICMP header starts after IP header)
    - IP + TCP + Data
    - IP + UDP + Data
    - IP + ICMP

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

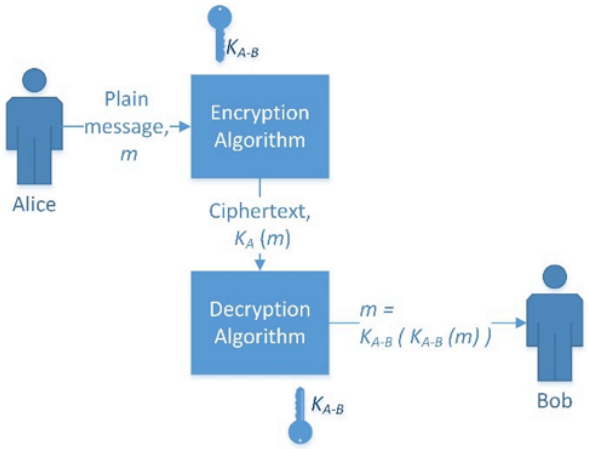
CS2105 Lecture 8 Network Security

Cryptography often uses keys :
1) Algorithms are known to everyone
2) Only “keys” are secret

Objectives	Strategy	Mechanism
Message confidentiality	Encrypt message so nobody can read	1) Symmetric key 2) Public key
Message Integrity	Implement check that original message is not modified	Cryptographic hash functions
Message Authenticity	Ensure message is indeed sent by sender	1) Message Authentication Code 2) Digital Signature

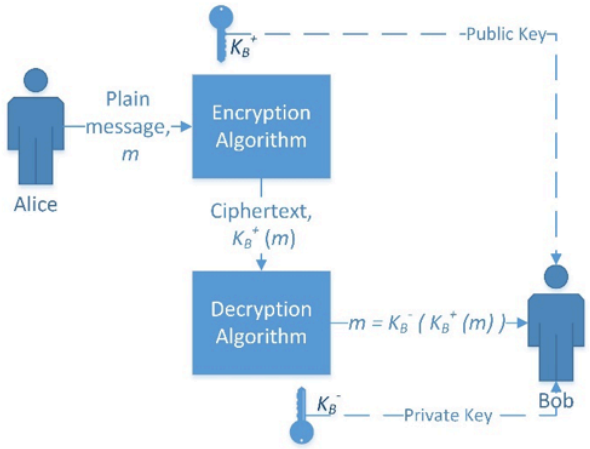
Symmetric Key vs Public Key

Symmetric Key



Con: How do you agree on a key in the first place?
⇨ Key distribution is a problem
Popular Algorithms: Data Encryption Standard (100 times faster than RSA), Advanced Encryption Standard, WEP for WiFi

Public Key

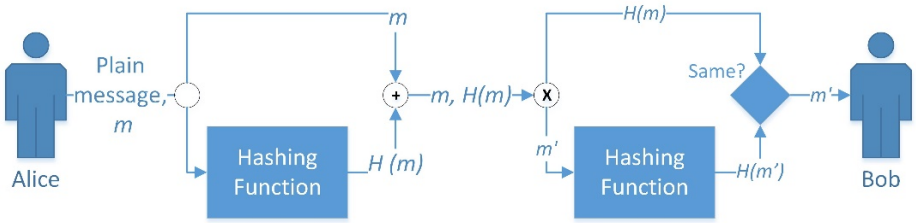


Alice can encrypt with private key which Bob can then decrypt with public key ➔ Alice is the real sender

Popular Algorithms: RSA (Slower, does not require key sharing)

In practice, Alice and Bob use RSA to exchange symmetric key (session key) K_S.
Once both have K_S, they use symmetric key cryptography

Hashing



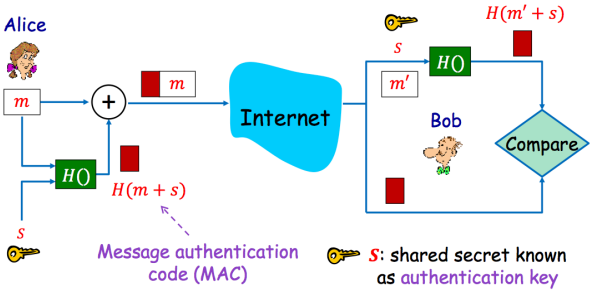
Hashing should be one-way & no two message should have same hash
➔ $H(m) \neq H(m')$

Popular Algorithms: MD5, SHA-1

Message Authentication Code vs Digital Signature

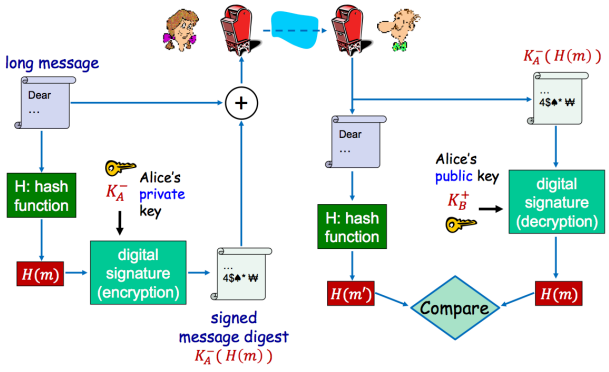
Message Authentication Code	Digital Signature
1) Can detect accidental and intentional changes to a message. 2) Can affirm to the receiver, the message's origin Shared key is part of the message digest generation, i.e., MD5 enhanced. Sends $m + H(m + s)$ Receiver extracts m' and generates $H(m' + s)$ using own copy of s and compares the two digest Since key is shared, impossible to key who authored the message	1) Verifiable : recipient can verify sender has signed the document. 2) Non-repudiation : Third party is confident that the document is signed by sender Sender encrypts message (or hash) with own private key ➔ the sent message must be written by sender Since key is private, message must be authored by sender

Message Authentication Code



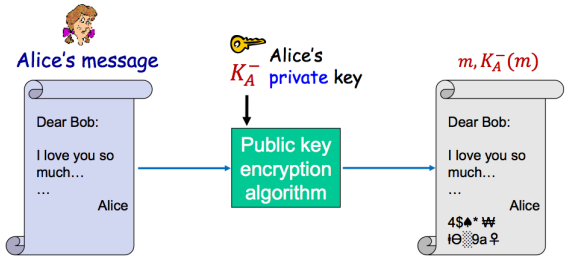
- MAC proves to Bob the origin of message (i.e. Alice) and verifies message integrity.

Digital Signature = Signed Message Digest

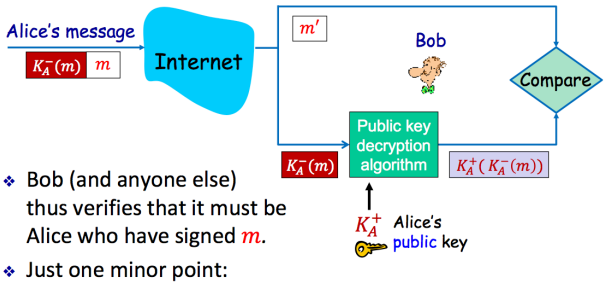


Digital Signature Example (1/2)

- Alice signs m by encrypting it with her private key K_A^-, creating a "signed" message, $K_A^-(m)$.
  - Send both m and $K_A^-(m)$ to Bob.



Digital Signature Example (2/2)



- Bob (and anyone else) thus verifies that it must be Alice who have signed m.
- Just one minor point:
  - Public key encryption is very slow.
  - Efficiency is a concern if m is long.

SSL : Secure Sockets Layer

SSL is a widely deployed security protocol.

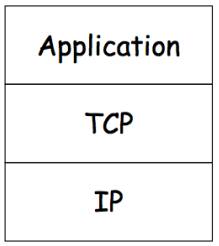
- Application to TCP applications
- A variation is TLS (Transport Layer Security)
- Support by all modern browsers and web servers

Common SSL symmetric ciphers

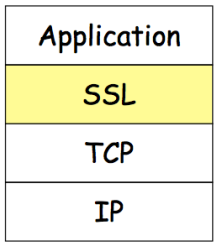
- DES - Data Encryption Standard : block
- 3DES - Triple strength : block
- RC2 - Rivest Cipher 2 : block
- RC4 Rivest Cipher 4 : stream

SSL public key encryption

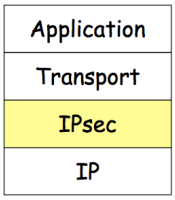
- RSA

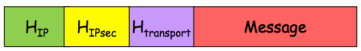


Normal Application



Application with SSL





Packet structure w/ IPsec

Security Goal	Hash function	MAC	Digital signature
Accidental damage to message	Yes	Yes	Yes
Message integrity	No	Yes	Yes
Message authenticity	No	No	Yes
Kind of keys	None	Symmetric key	Public key

CS2105 Lecture 9

Link Layer

- Network layer provides communication service between two hosts
- IP datagram may travel through multiple routers and links before it reaches destination
- Link layer sends datagram between adjacent nodes over a single link
- IP datagrams are encapsulated in link-layer frames for transmission
- Different link-layer protocols may be used on different links

---

Possible Link Layer Service

- Framing
  - Encapsulate datagram into frame, adding header and trailer
- Link access control —> Coordinate which node to send/transmit first
  - When multiple nodes share a single link, it is required to coordinate which nodes can send frames at a **certain point of time.**
- Reliable Delivery
  - Seldom used on low bit-error link (fiber optics) but often use on error-prone link (Wifi)
- Error detection
  - Errors usually caused by signal attenuation or noise
  - Receiver detects presence of errors (**may signal sender for retransmission or simply drops frame)**
- Error Correction
  - Receiver identifies and corrects bit error(s) without resorting to retransmission

---

Link Layer Implementation

- Link layer is implemented in “adapter” (hardware) (aka NIC - Network Interface Card) or on a chip
- Implemented in hardware to **achieve higher processing speed**

Parity Checking

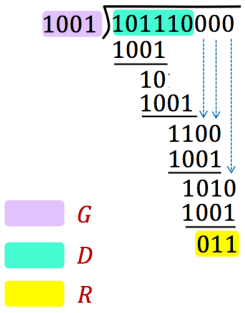
Single Bit Parity
- Failed when two bits changed
- Can detect single bit error in data

Two-dimensional bit parity
- Can **detect and correct** single bit errors in data
- Can **detect** any two bit errors in data

Cyclic Redundancy Check (CRC)

Powerful error-detection coding that is widely used in practice (e.g., Ethernet, Wi-Fi) Success rate : 99.99%
- D: data bits, viewed as a binary number.
- G: generator of $r+1$ bits, agreed by sender and receiver beforehand.
- R: will generate CRC of r bits.

Example: $r = 3$



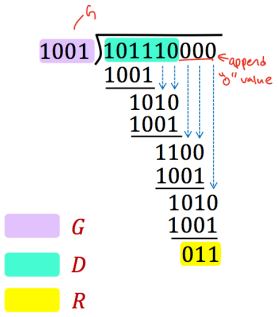
CRC calculation is done in bit-wise XOR operation without carry or borrow.

Sender sends (D, R) CRC value

101110011

Receiver knows G, divides (D, R) by G.
- If non-zero remainder: error is detected!

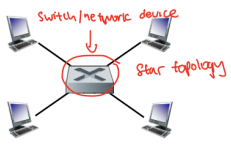
Example: $r = 3$



Types of Network Link

Type 1: point-to-point link
- A sender and a receiver connected by a dedicated link
- Example protocols: Point-to-Point Protocol (PPP), Serial Line Internet Protocol (SLIP)
  - No need for multiple access control

A host connects to router through a dedicated link



A point-to-point link between Ethernet switch and a host

Type 2: broadcast link (shared medium)
- Multiple nodes connected to a shared broadcast channel.
- When a node transmits a frame, the channel broadcasts the frame and each other node receives a copy.

802.11 Wi-Fi

Satellite

Ethernet with bus topology

Multiple Access Protocols

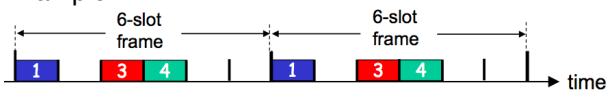
- In broadcast channel, if two or more nodes transmit simultaneously —> **collision occur**
  - Each nodes receives multiple frame at the same time
    - Frames collide at nodes and none will be correctly read
- **Distributed algorithm** that determines how nodes share channel (example, when a node can transmit)
- Coordination about channel sharing **must use channel itself.**
  - **No out-of-band channel signalling**

- **Channel Partitioning**
  - divide channel into smaller "pieces"
  - allocate piece to node for exclusive use
- **Taking turns**
  - nodes take turns to transmit
- **Random Access**
  - channel is not divided, collisions are possible
  - "recover" from collisions

Channel Partitioning Protocols

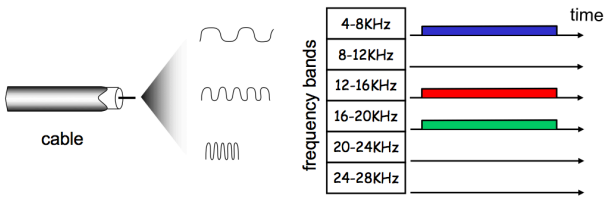
TDMA (Time division multiple access)
- Access to channel in "rounds"
- Each nodes get fixed length slot (length = frame transmission time) in each round
- Unused slots go idle

Example :



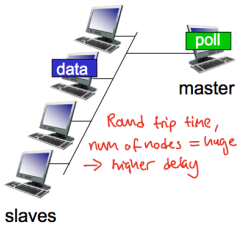
FDMA (frequency division multiple access)
- Channel spectrum is divided into frequency bands
- **Each nodes** is assigned a **fixed frequency band**
- Unused transmission time in frequency bands go idle.

Example :

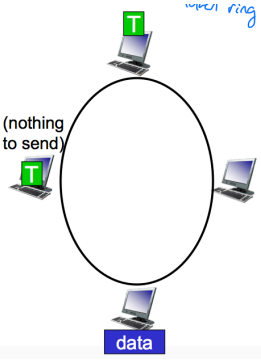


Taking Turns Protocols

Polling:
- master node "invites" slave nodes to transmit in turn.
- concerns:
  - polling overhead
  - single point of failure (master node)



Token passing:
- control token is passed from one node to next sequentially.
- concerns:
  - token overhead
  - single point of failure (token)



Random Access Protocols

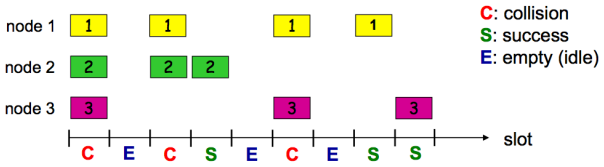
When node has packet to send
- Two or more transmitting nodes —> **Collision happens**

Random access protocols specify
- how to detect collisions
- how to recover from collisions (e.g. delayed retransmissions)

Slotted ALOHA

Assumptions :
- All frames are of equal size
- Time is divided into slots of equal length (length = time to transmit 1 frame)
- Nodes start to transmit only at the beginning of a slot

Operations :
- Listens to channel while transmitting **(Collision detection)**
- **Retransmit** a frame in subsequent slot with probability p until success

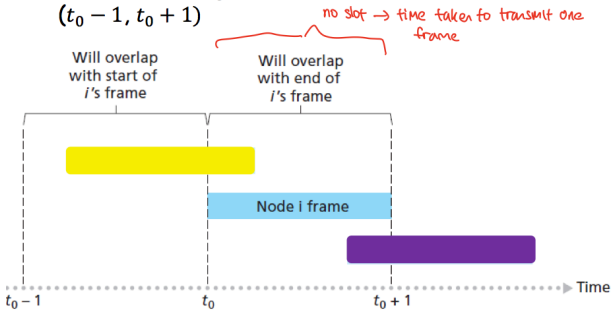


Pure (unslotted) ALOHA

- Even simpler : no slot, no synchronisation
- When **fresh frame exist**, **transmit immediately**

However, chances of collision increases

- frame sent at t_0 collides with other frames sent in $(t_0 - 1, t_0 + 1)$



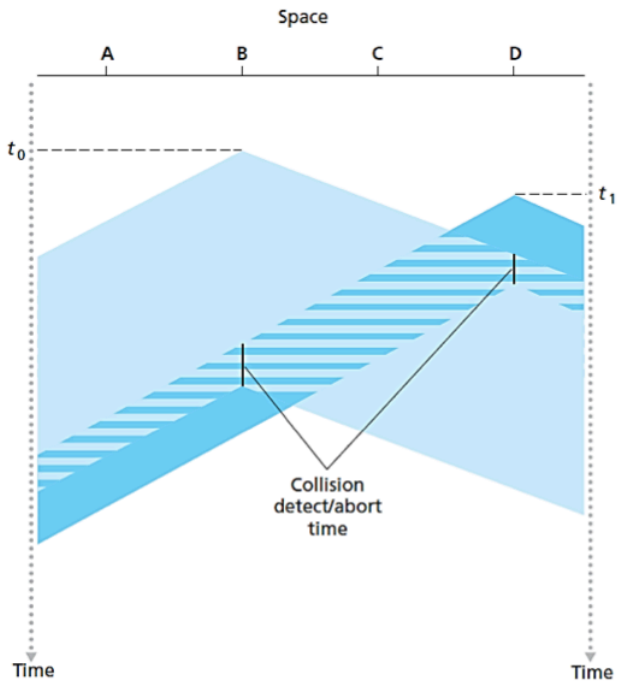
Carrier Sense Multiple Access - Collision Detection (CSMA/CD)

CSMA Overview

Sense the channel before transmission, if (idle) { transmit } else if (busy) { defer sending, choose ***k={1…2^m-1}*** }

Collisions can still occur because of propagation delay

Carrier Sense Multiple Access/Collision Detection

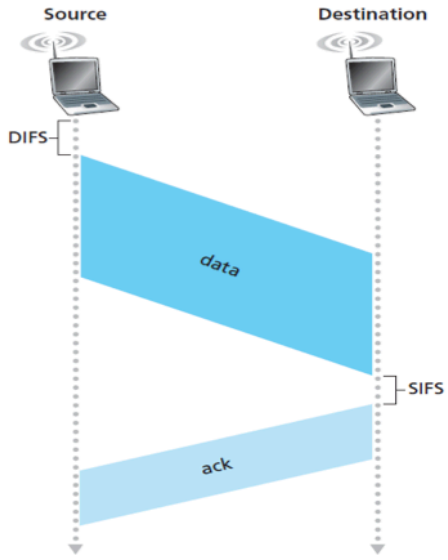


Used in Ethernet

Principles:
1) To ensure that d_{prop} will never be too large such that a collision is unnoticed
   - ⇨ min frame size = bandwidth×RTT
   - ⇨ min frame size, $L > 2 \times d_{prop} \times R$
2) Once a **collision in detected while transmitting, operation is aborted**
3) **After aborting**, NIC enters binary back-off
   a. **After m^{th} collision, NIC chooses K at random from where $P(K) = \frac{1}{2^m}$**
   b. **NIC waits $K \times 512$ bit time**

Carrier Sense Multiple Access - Collision Avoidance (CSMA/CA)

Carrier Sense Multiple Access/Collision Avoidance



Used in WiFi

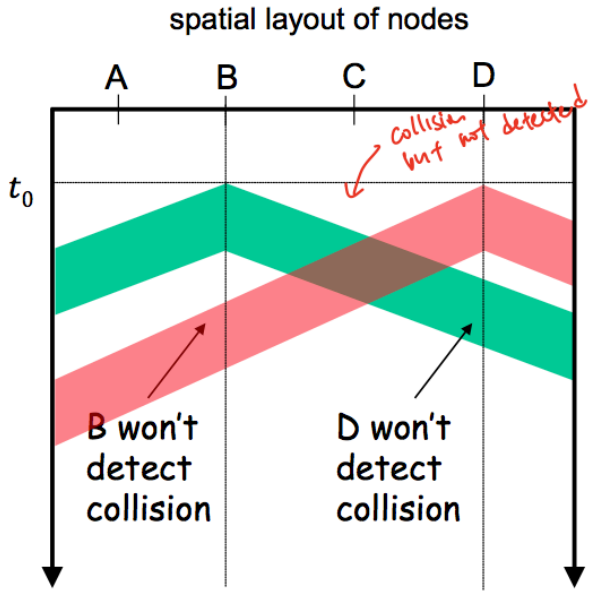
Process:

1) If sensed that channel is idle, begin transmission
2) Else choose a backoff value using exponential backoff
3) Transmit the entire frame and wait for acknowledgement
4) If ACK is received, begin next frame

A lot of waiting because the whole point is to avoid collisions since collisions cannot be detected. Hence, when one sends, it sends the WHOLE THING

Minimum Frame size

- What if the frame size is too small?
  - Collision happens but may not be detected by sending nodes.
    - No retransmission!
- For example, Ethernet requires a minimum frame size of 64 bytes.



CS2105 Lecture 10

MAC Address

- Every adapter (NIC) has a mac address
  - Used to send and receive link layer frames
  - **MAC address (48 bits) burned in NIC** (First three bytes identify vendor)
- When an adapter receive a frame, it checks if the destination MAC address of the frame matches its own MAC address
  - If **yes**, adapter extracts the enclosed datagram and passes it to the protocol stack
  - if **no,** adapter discards the frame without interrupting the hosts.

IP Address	**MAC Address**
• 32 bits in length • Network-layer address used to move datagram from source to destination • Dynamically assigned to facilitate routing	• 48 bits in length • Each network has a unique mac address • Link-layer address used to move frame over every single link • **Permanent**

Switches & Routers

Routers	**Switches**
• Understand both IP and MAC address • Check IP address • Store-and-forward • Compute route to destination • Require manual configuration	• Check MAC Address • Store-and-forward • Forward frame to outgoing link or broadcast • Plug and use

Local Area Network (LAN)

- Is a computer network that interconnects computer
  - IBM Token RING
  - Ethernet
  - Wi-Fi

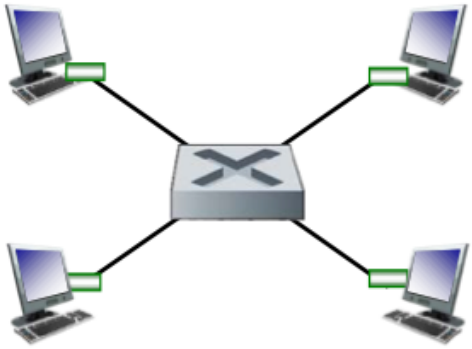
Ethernet

- "Dominant" wired LAN technology
  - Developed in mid 1970s

All nodes can collide with each other

Ethernet with bus topology

Nodes do not collide with each other



Ethernet with star topology

Ethernet Data Delivery Service

- **Connectionless :**No handshaking between sending and receiving NICs
- **Unreliable :** Receiving NIC does not send ACK or NAK to sending NIC
  - **Data in dropped frames will only be recovered if sender uses high layer rat**
- Ethernet's multiple access protocol : CSMA/CD with binary backoff

Collisions may happen in Ethernet of bus topology

Example :
- A sends a frame at time t
- A's frame reaches D at time t + d (propogation delay)
- D begins transmission at time t + d - 1 and collides with A's frame

Ethernet CSMA/CD Algorithm

1. NIC receives datagram from network layer, creates frame.
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame!
4. If NIC detects another transmission while transmitting, aborts and sends jam signal.
5. After aborting, NIC enters binary back-off:
   - after m^{th} collision, NIC chooses K at random from {0, 1, 2, …, 2^m-1}.
   - NIC waits K*512 bit times, returns to Step 2.

waiting time = transmission delay of 512 bits

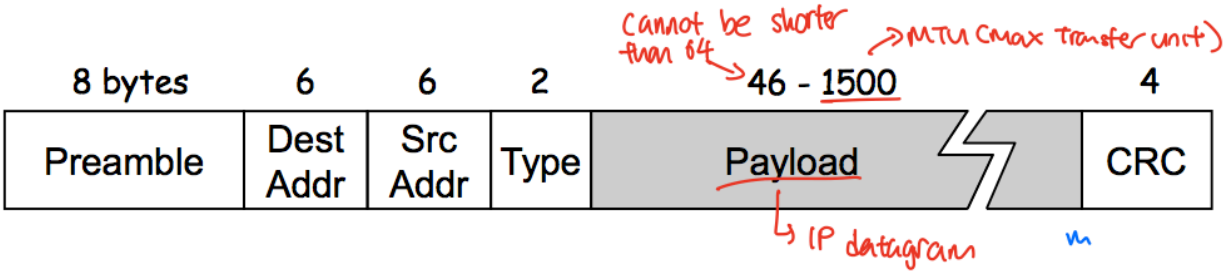
Exponential backoff:

- After 1st collision: choose K at random from {0, 1}; wait K * 512 bit transmission times before retransmission.
- After 2nd collision: choose K from {0, 1,…, 2^2-1}.
  …
- After m^{th} collision, choose K at random from {0, 1, …, 2^m-1}

- *Goal*: adapt retransmission attempts to estimated current load
  - More collisions implies heavier load.
  - longer back-off interval with more collisions.

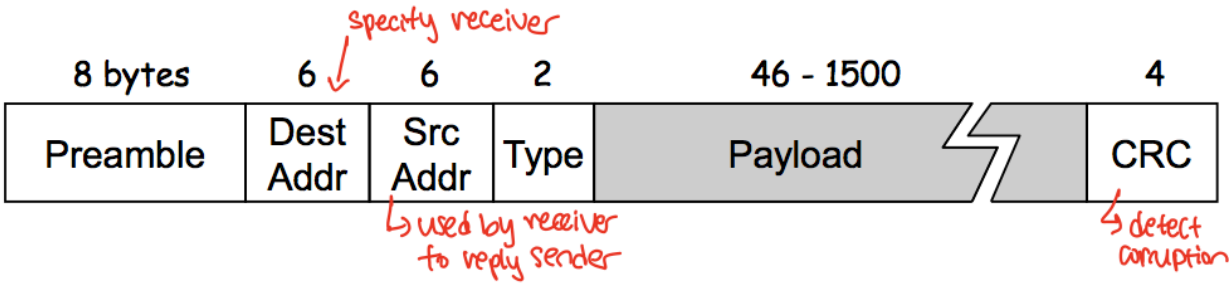
Ethernet Frame Structure (1/2)

- Sending NIC (adapter) encapsulates IP datagram in Ethernet frame.



- *Preamble:*
  - 7 bytes with pattern **10101010** followed by 1 byte with pattern **10101011**.
  - used to synchronize receiver and sender clock rates.

↳ to know how wide a bit is.



- *Source and dest MAC address:*
  - If NIC receives a frame with matching destination address, or with broadcast address, it passes data in the frame to network layer protocol.
  - Otherwise, NIC discards frame.
- *Type:* Indicates higher layer protocol (mostly IP).
- *CRC:* corrupted frame will be dropped.

.

Each IP node (host / router) has an ARP table , used to store mapping of IP address and MAC address of other nodes in the same subnet

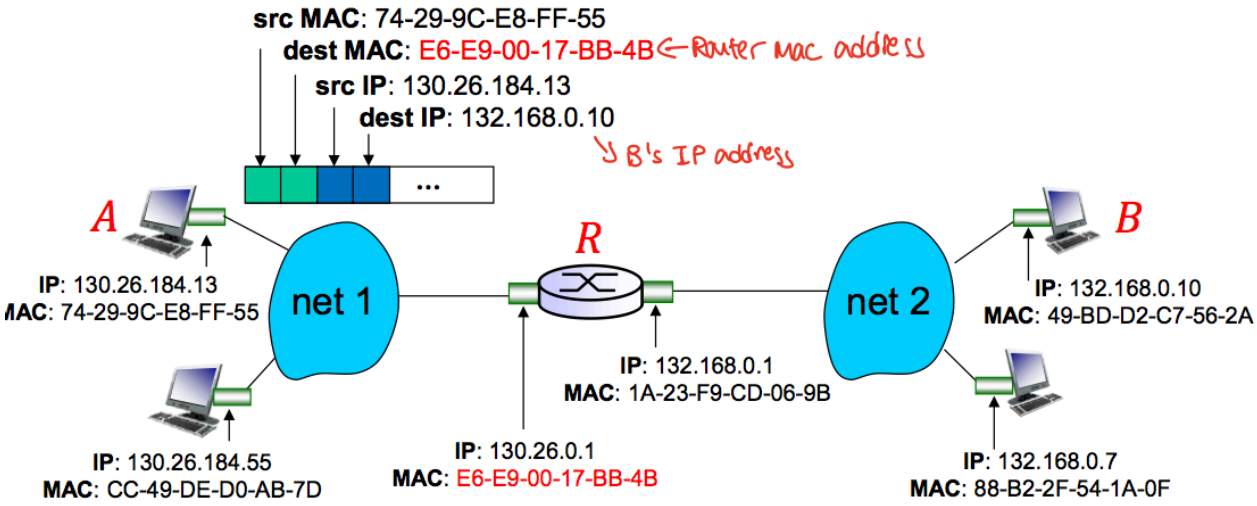
Sending Frame in the Same Subnet

- Suppose A wants to send data to B. They are in the same subnet.
  1. If A knows B's MAC address from its ARP table
     - create a frame with B's MAC addresses and send it.
     - Only B will process this frame.
     - All other nodes will receive but ignore this frame.

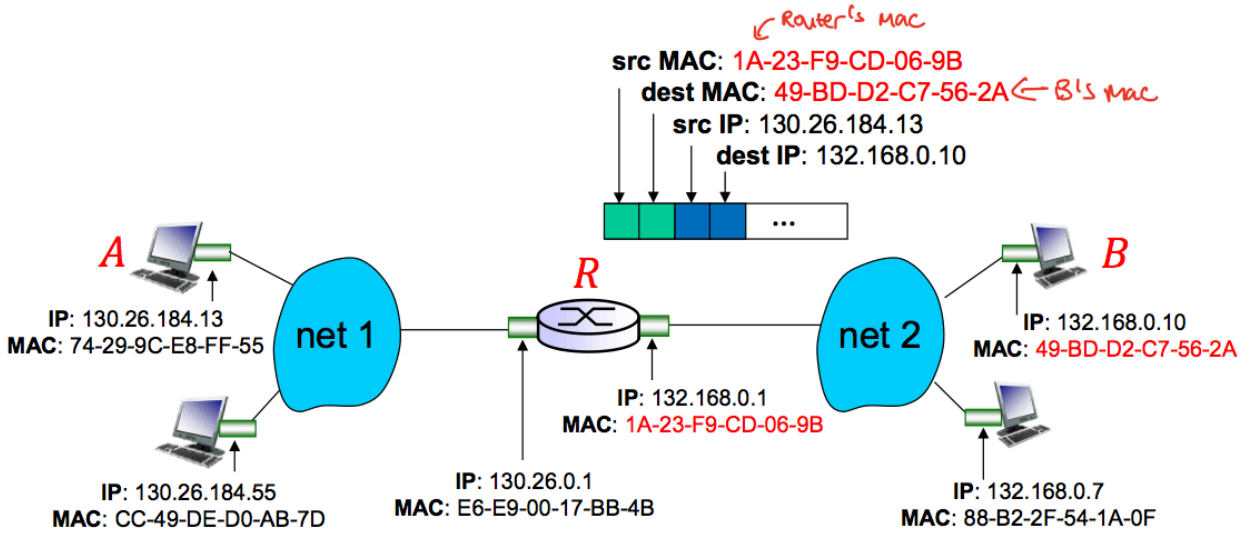
- What if B's MAC address is not in A's ARP table?
  1. A broadcasts an ARP query packet, containing B's IP address.
     - Dest MAC address set to FF-FF-FF-FF-FF-FF
     - All the other nodes in the same subnet will receive this ARP query packet, but only B will reply it.
  2. B replies to A with its MAC address.
     - Reply frame is sent to A's MAC address.
  3. A caches B's IP-to-MAC address mapping in its ARP table.

Sending Frame to Another Subnet

- A sends datagram to B in another subnet.
  - A should create a link-layer frame with (1) R's MAC address (2) B's IP address as destination.



  - R will move datagram to outgoing link and construct a new frame with B's MAC address.

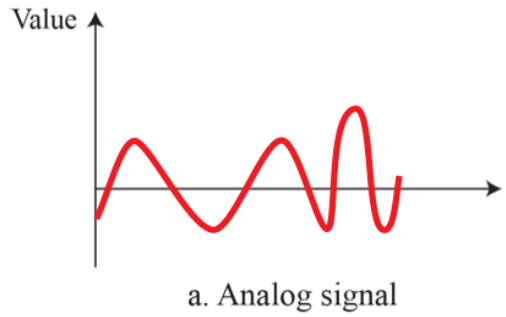


Lecture 11

Digital and Analog Signals

Analog signal

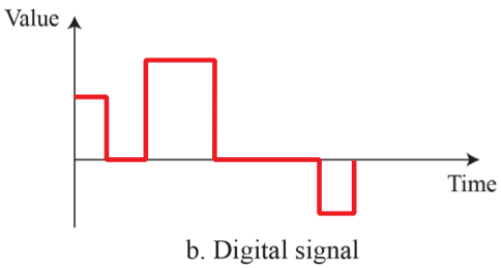
- Is continuous with infinitely many levels



a. Analog signal

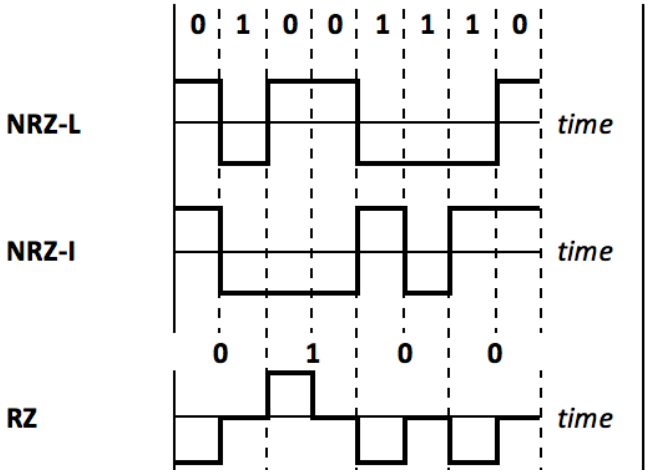
Digital signal

- Has a limited number of defined values



b. Digital signal

Digital Signals



Non-Return-to-Zero-Level
Absolute voltage level determines value of bit
e.g., High ➔ 0, Low ➔ 1

Non-Return-to-Zero-Invert
Invert the voltage if bit 1 is encountered
Starting volt level should be specified

Return-to-Zero
Three voltage levels, always return voltage to zero halfway through bit interval.

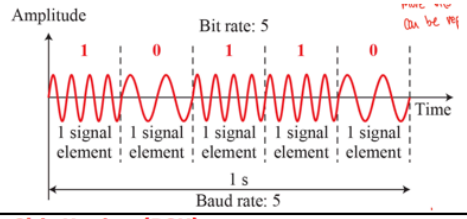
$$\textbf{Shannon Capacity} = B \times \log_2(1 + SNR)$$

$$\textbf{Sine wave} = A\sin(2\pi f t + \Phi)$$

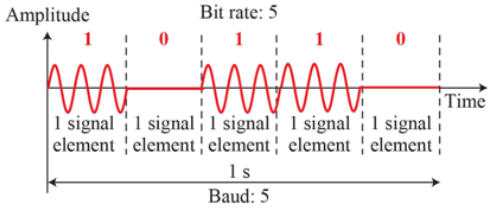
Bandwidth of a channel, B = Highest f − Lowest f
Signal to Noise Ratio, SNR = Strength of Signal/Noise

Signal to noise (SNR): measuers the strength of signal over noise
High SNR : able to decode signal
LOW SNR : unable to decode signal

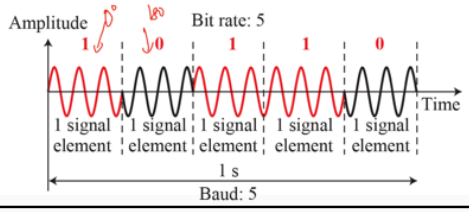
Frequency Shift Keying (FSK)
Amplitude and phase remain constant
FSK limited by bandwidth
2 *freq* ➔ 1 bit
4 *freq* ➔ 2 bits
BUT **more susceptible to noise**



Amplitude Shift Keying (ASK)
Vary the **amplitude** to signal data values
ASK susceptible to noise



Phase Shit Keying (PSK)
Change the phase of the signal to represent data
More phases
➔ Difference between phases is smaller
➔ Channel can distort phases
E.g., 8-PSK means 3-bits per signal



Quadrature Amplitude Modulation (QAM)
Combination of ASK and PSK.
⇨ Every signal differ in either amplitude or phase
A signal unit in a 2^k-QAM represents k bits
Baud rate is number of signal units ***sent*** per second
Bit rate is number of bits receiver receives per second
⇨ Bit Rate = μ×Baud Rate×k
μ is proportion of signals received by receiver

Glossary

Request-header

Accept: Used to specify certain media types acceptable for the response, e.g., JSON
Accept-Charset: Indicate what character sets are acceptable for the response
Authorization: Consists of credentials containing the authentication information of the user agent
Expect: To indicate server behaviours requested by client. If unable to fulfil, respond with ERROR 417
From: SHOULD contain email address for human user
Host: Specifies internet host and port number of resource being requested
If-Modified-Since: If requested variant not modified since time specified, server will NOT return anything (304)
Max-Forwards: A mechanism for TRACE and OPTIONS methods to limit number of forward requests
Range: Byte-range
Referer: Used to generate lists of back-lists to resources for interest, logging, optimized caching
User-Agent: Information about user agent originating the request. Product token listed in order of significance

Response-header

Date: The date and time at which the message was originated
Server: Software used by origin server to handle request, e.g. Apache
Accept-Ranges: What partial content range types this server supports
Content-Type: Indicates the media type of the entity-body sent
Content-Length: The length of the **response body** in bytes
Status: Status of the HTTP response, e.g. 200
Keep-Alive: timeout-time host will allow idle connection to remain open before closing; max-maximum number of requests that a client will make on the persistent connection

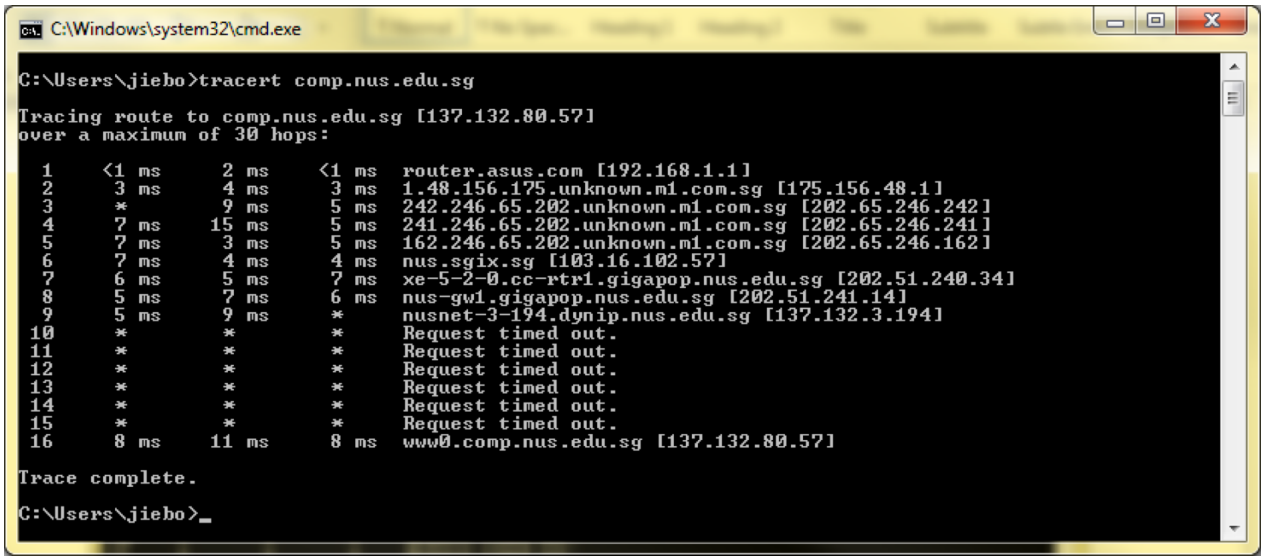
Layers shortcut

Layer	Overview	Objectives	Protocols/Mechanisms	
Application-layer message	Supports network applications, **client-server/P2P/Hybrid** e.g. FTP, SMTP, HTTP, DNS Assumes that socket is black box so just pick up the data. Therefore, transport layer should take note of reliability, throughput, delay and security	Types of messages exchanged Message syntax Message semantics Rules for when and how applications send & respond to messages		
Transport-layer segment	**Process-to-process data transfer** e.g., TCP, UDP	Data Integrity Handle delays/reordering Throughput Security	Reliable data transfer	
Network-layer datagram	Routing of datagrams from source to destination, **host-to-host** e.g., IP Runs on routers	Delivers packets to receiving hosts	DHCP[1] CIDR[2] RIP[3]	NAT[4] IP Fragmentation ICMP[5]
Link-layer frame	Data transfer between neighbouring network elements e.g., Ethernet, WiFi	Error Detection Error Correction	CRC (Cyclic Redundancy Check) TDMA FDMA	ALOHA CSMA MAC Addressing ARP
Physical	Bits "on the wire"			

Network Commands

Traceroute

tracert <IP Address OR URL> (windows) / traceroute <IP Address or URL> (Mac)



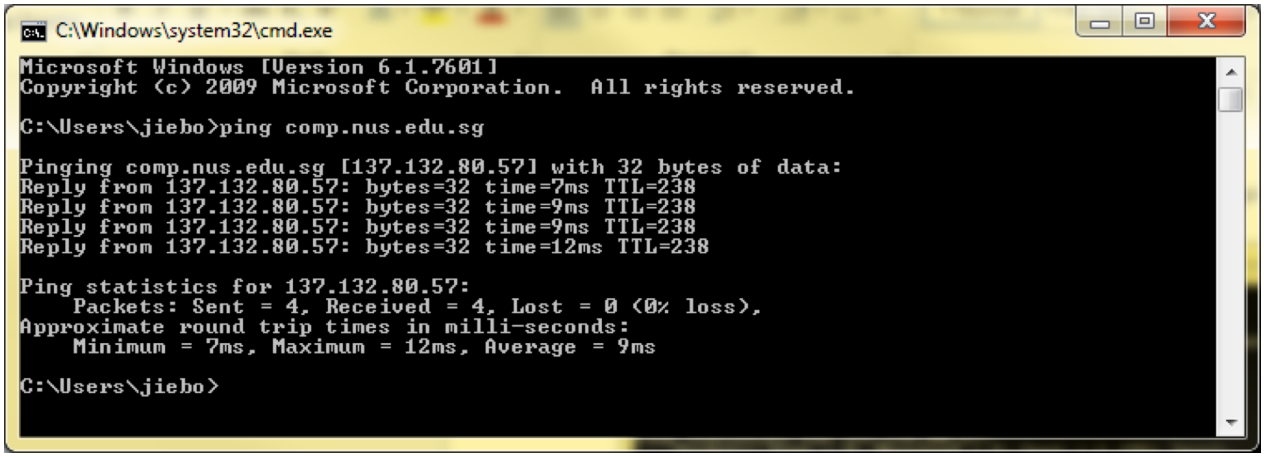
1 is the internet gateway on my network. i.e., my home's IP address
2 is the ISP
6 is the Singapore Internet Exchange
7 – 8 is the Singapore Open Exchange
9 is Dynamic DNS Service
16 is the computer which `comp.nus.edu.sg` is hosted on

Each hop is tested 3 times 3 columns of timings
Timeout meant that there was no response from the router, so another one was tried

```
-d
   Specifies to not resolve addresses to host names

-h maximum_hops
   Specifies the maximum number of hops to search for the target

-j host-list
   Specifies loose source route along the host-list

-w timeout
   Waits the number of milliseconds specified by timeout for each
   reply

target_host
   Specifies the name or IP address of the target host
```

Ping

ping <IP Address> l-L <number> *(to increase packet size)*

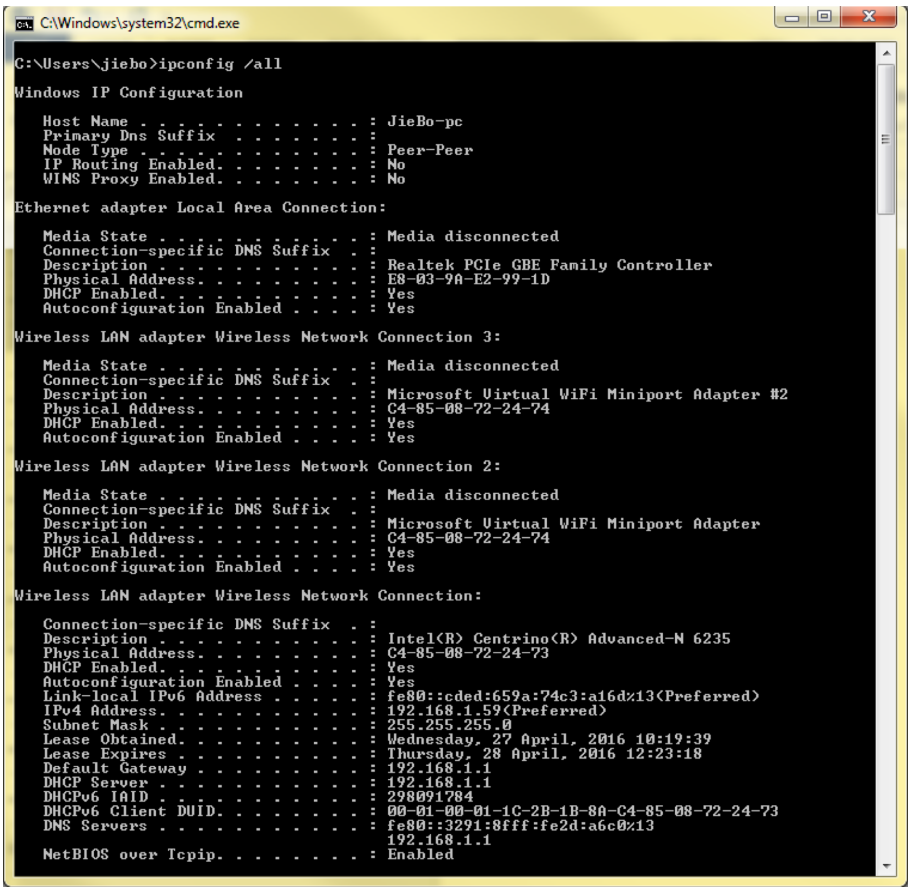


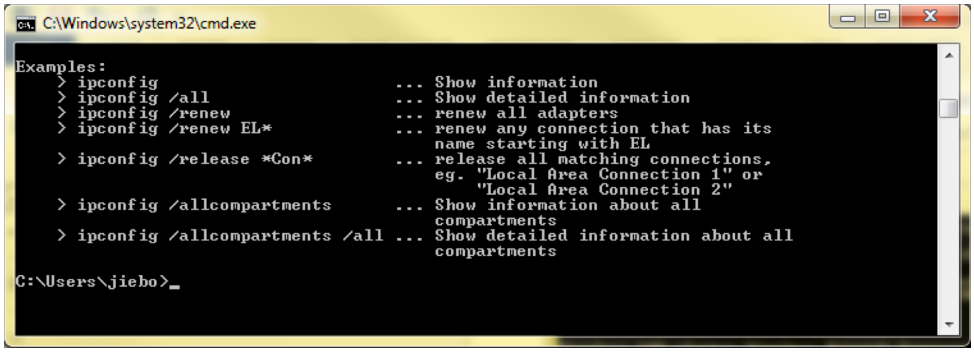
Outcome	Probable Cause
Four replies (see above)	Workstation able to communicate with specified host at TCP/IP level
Four timeouts	TTL expired: - Communication problems between two machines (cable/routing table) - Has communications but too slow for ping (network congestion?) - Firewalls blocking ICMP
Some replies, some timeouts	Network congestion, bad network cabling, or faulty hardware
Transit failed	TCP/IP not configured correctly on current workstation.
Can ping IP but not domain name	DNS server may be wrongly configure

To troubleshoot:

1) Ping default gateway (see ipconfig) Can reach gateway router
2) Ping DNS server (see ipconfig) Can reach DNS server
3) nslookup (see nslookup) the destination domain name DNS server can resolve the domain name
4) Ping the returned IP addr from (3) Can reach destination server

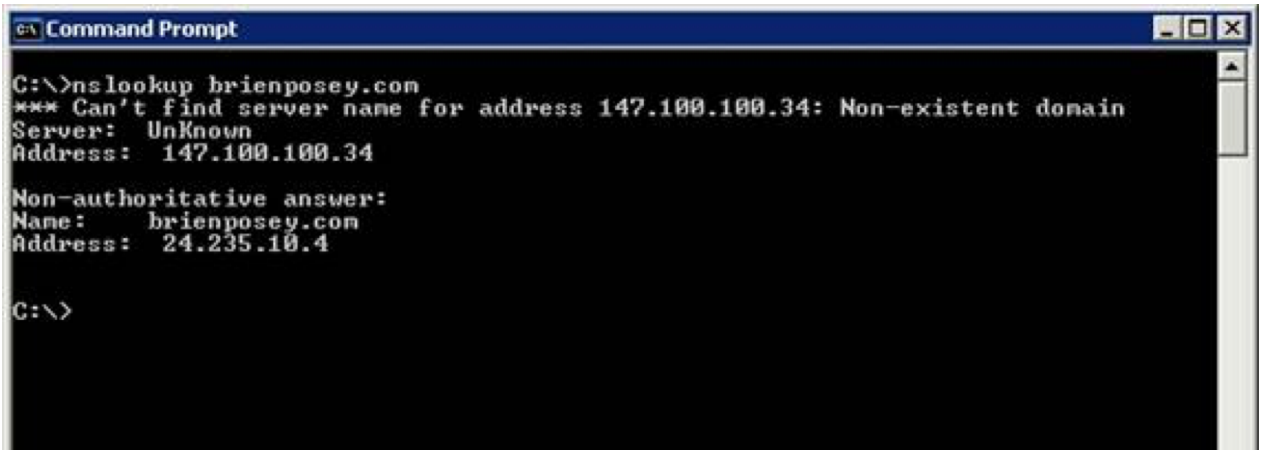
Ipconfig





`ipconfig /flushdns` to clear DNS cache value on your computer. DNS uses TTL (Time-To-Live) value which let the intermediate name servers to cache DNS information. If you changed your DNS settings, and your computer doesn't see the change immediately, you may perform "ipconfig /flushdns" to clear the DNS cache.

NSlookup



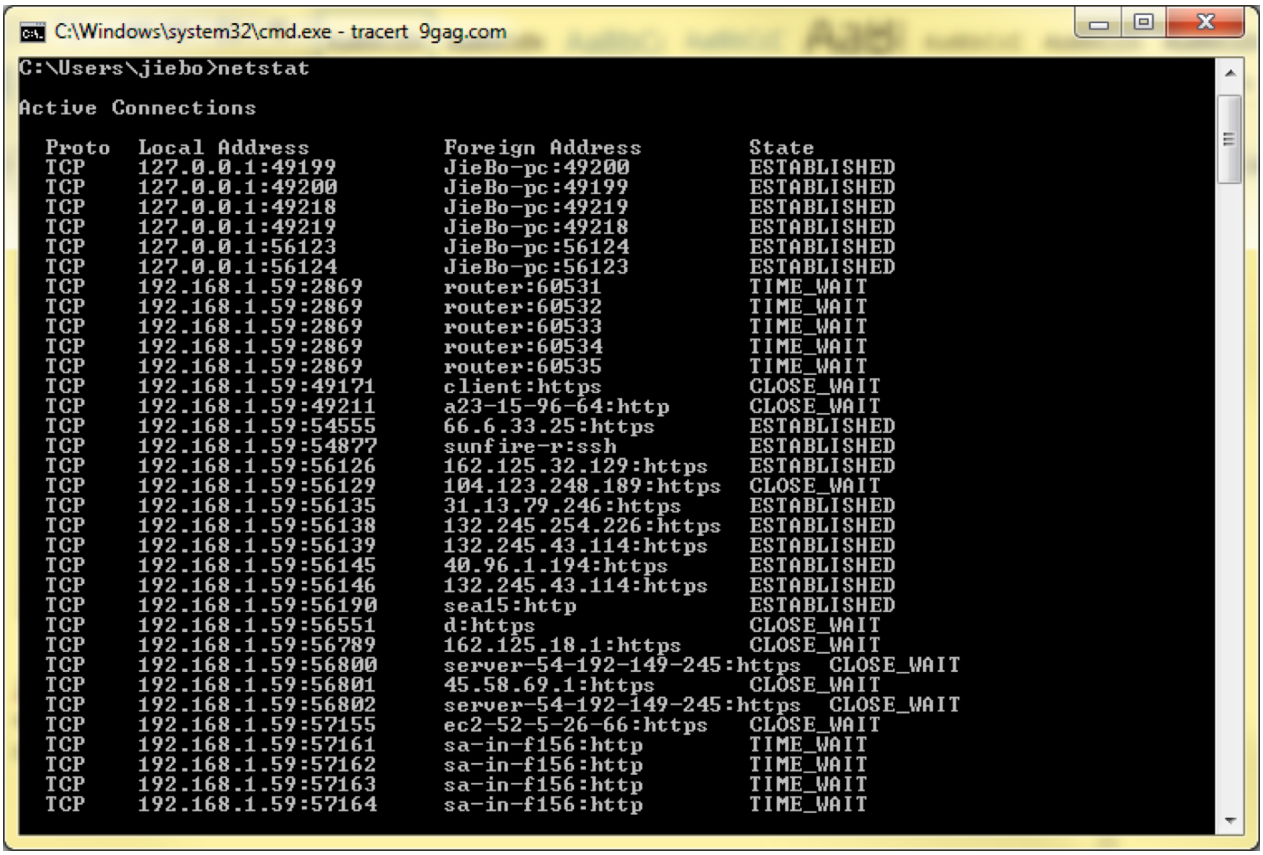
If IP Address is listed, DNS query was successful.

If IP addr is different from the destination IP, DNS poisoning may have occurred. In which case,

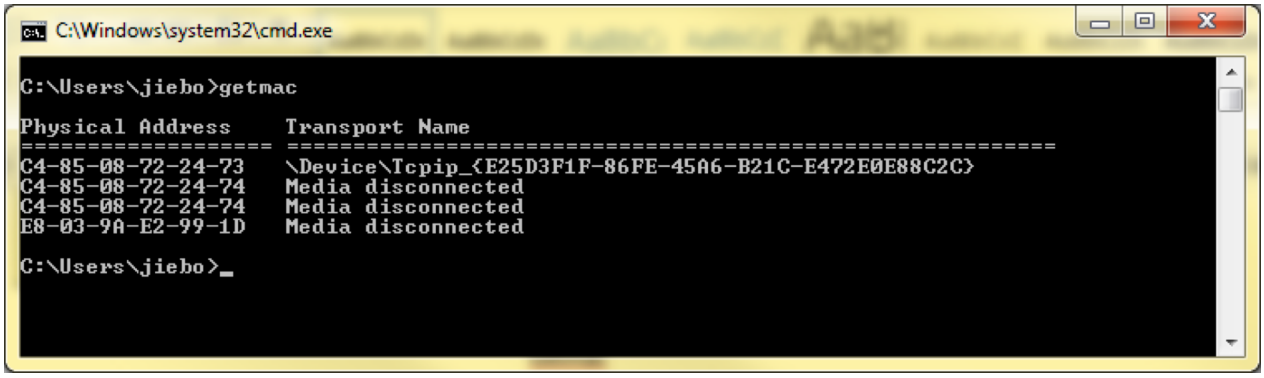
`ipconfig /flushdns`

Netstat

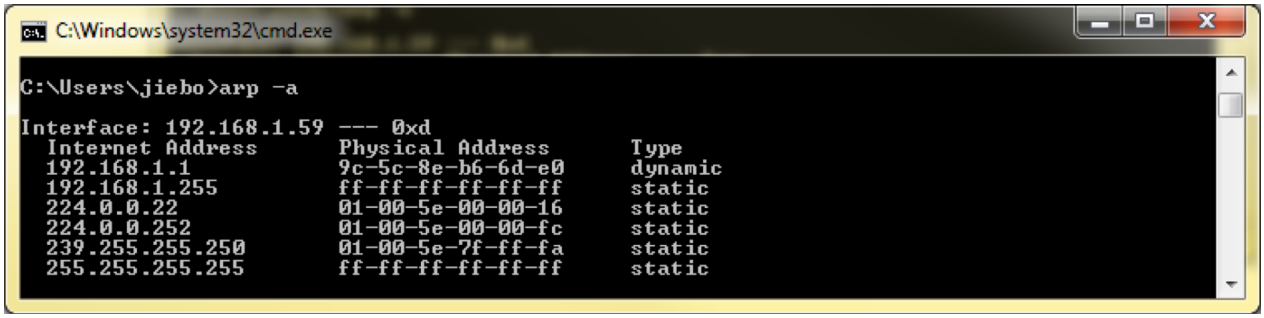
`netstat` used to view your active network connections and TCP/IP connections. Determine what ports are open and used, what programs are using your ports and what kind of TCP and UDP connections are present.



getmac


C:\Windows\system32\cmd.exe
C:\Users\jiebo>getmac
Physical Address Transport Name
C4-85-08-72-24-73 \Device\Tcpip_{E25D3F1F-86FE-45A6-B21C-E472E0E88C2C}
C4-85-08-72-24-74 Media disconnected
C4-85-08-72-24-74 Media disconnected
E8-03-9A-E2-99-1D Media disconnected
C:\Users\jiebo>

ARP


C:\Windows\system32\cmd.exe
C:\Users\jiebo>arp -a
Interface: 192.168.1.59 --- 0xd
Internet Address Physical Address Type
192.168.1.1 9c-5c-8e-b6-6d-e0 dynamic
192.168.1.255 ff-ff-ff-ff-ff-ff static
224.0.0.22 01-00-5e-00-00-16 static
224.0.0.252 01-00-5e-00-00-fc static
239.255.255.250 01-00-5e-7f-ff-fa static
255.255.255.255 ff-ff-ff-ff-ff-ff static