~~2.4~~
~~2.5~~
~~2.6~~
~~2.7~~

~~3.1~~
~~3.2~~
~~3.3~~
~~3.4~~

Network architecture
- 5 layers
  - Application
    - Network applications reside, message, attachment ect
    - Application layer packet known as a message (in book)
  - Transport
    - Transports application layer between application endpoints
    - Two transport protocol
      - TCP
        - Connection oriented service
        - Guranteed delivery of application layer message and flow control
        - Breaks long messages into shorter segments and provides congestion control
      - UDP
        - Connectionless service to its application
        - No reliability, no flow control, no congestion control.
    - Transport layer packet known as segment
  - Network
    - Packets known as datagrams
    - Transport layer Passes segment and destination addresses to network layer
    - IP protocol
      - Defines fields in datagram as well as how end systems and routers act on these fields
    - Routing protocol
      - Determines the route that datagrams take between source and destination
  - Link
    - Moves packet from one node to another
    - network layer Passes datagram down to link layer
    - Link layer moves to next node
    - Link layer passes datagram up to network layer
    - Link layer protocol (wifi, eithernet)
    - Link layer packets known as frames
  - Physical
    - Moves individual bits within the frame from one node to the other

Application architecture
- Designed by application developer
- Dictates how application is structured over various end systems
- Two predominant architecture paradigms
  - Client-server architecture
    - Always-on host

- Server
  - Services request from other hosts called clients
- Clients do not directly communicate with each other
- Server has a fixed IP address
- Datacenter
  - Houses large number of host used to create a powerful virtual center

Peer to peer architecture

- Minimal to no reliance on datacenter
- Exploits direct communication between pairs of connected hosts called peers
- Peers
  - Not owned by service providers
  - Normal users
- 3 major challenges
  - ISP friendly
    - Residential ISP asymmetrical, more down stream than upstream traffic
  - Security
    - Highly distributes and open nature, thus difficult to secure
  - Incentives
    - Convincing users to volunteer bandwidth, storage, and computation resources to application
- This is bit torrent, skype ect

Hybrids include aim, uses client-server to find IP and then uses peer to peer for rest

Processes communicating

- Processes on two different end systems communicate by exchanging messages across computer network
- Sending process creates and sends messages into the network
- Receiving process receives these messages and possibly responds by sending messages back
- One labeled client process / downloading and other server process / uploading
- Client server processes
  - Exchange messages with a web server process
- P2P
  - File is transferred in process in one peer to process in another peer

Socket

- Interface between application layer and transport layer within a host
- Referred to as application programming interface (API) between application and network layer
- Programming interface with which networks are built
- Application developer has control on everything on the application side of the socket but little control on the transport side
  - Choice of transport protocol (TCP or UDP)
  - Ability to fix a few transport layer parameters such as max buffer and max segment size

Addressing processes

- Two pieces of info needed for connection
  - address of the host
    - IP address
      - 32 bit number, uniquely identifies host
  - Identifier needed that specifies the receiving process int he destinations host
    - Port number
      - Specific port numbers
        - Web server
          - Port 80
        - Mail server process (SMTP)

Port 25

Important things
- Reliable data transfer
  - Guarantees that message will reach destination
- Loss-tolerant applications
  - Not all the data may reach destinations
- Bandwidth-sensitive applications
  - Force transport layer to secure a certain amount of bandwidth/ throughput
- Elastic application
  - Uses whatever bandwidth/ throughput is available
- Timing guarantees
  - Used for real time application
- Security
  - Transport layer can provide security services

Internet does
- Gives two transport protocols available
  - TCP
    - Connection oriented and reliable data transfer service
    - Connection oriented service
      - Has client and server exchanger transport layer control information with each other before the application level messages begin to flow
      - Prepares the client and server for onslaught of packets
      - After "handshake" TCP connection is said to exis between the sockets of two processes
      - Full duplex connection
        - Can send messages at the same time
      - When application finishes sending messages connection torn down
    - Reliable data transfer service
      - Can rely on TCP to deliver all data sent without error and in proper order whne one application passes a stream of bytes into a socket it can count on TCP to deliver the same stream of bytes to the receiving socket with no missing or duplicate bytes
    - Congestion control
  - UDP
    - Lightweight
    - Connectionless
    - Unreliable data transfer
    - No congestion control

Application layer protocol
- How application processes on different end systems pass messages to each other

- Defines
  - The type of messages exchanged, for example request messages and response messages
  - The syntax of the various message types such as the fields in the message and how the fields are delineated
  - The semantics of the fields that is the meaning of the information in the fields
  - Rules for determining when and how a process sends messages and responds to messages

Hypertext transfer protocol(HTTP)
- RFC 1945 RFC 2616
- Implemented in client and server
- Web page
  - Consist of objects
    - Most consist of a base HTML file
    - If we have a page with HTML text and five JPEG
      - Base HTML file references the other objects in the page with the objects url
        - Each Url has two components
          - Host name of the server
          - Objects path name
        - http://www.someschool.edu/somedepartment/picture.gif
          - Someschool.edu is host name
          - /somedepartment/picture.gif is path name
  - Uses TCP
  - Stateless protocol
    - HTTP servers maintain no information about clients
  - Non-persistent connection
    - Each request/response pair is sent over a separate TCP connection
    - Process
      - 1 HTTP client process initiates TCP connection to server someschool.edu on port 80
      - 2 HTTP client sends an HTTP request message to the server via its socket includes path name
      - 3 HTTP server process receives the request message via socket retrieves object /somedepartment/home.indx from storage encapsulates the object in an HTTP response message and sends the response message to client
      - 4 HTTP server process tells TCP to close connection
        - TCP doesnt close connection until it knows for sure client received info
      - 5 HTTP client receives the response message TCP connection terminated message indicates the encapsulated object is an HTML file the client extracts the file from the response message examines the HTML file and finds the references to the 10 jpegs
      - The first four steps are repeated for each of the referenced JPEG objects
  - Persistent connection
    - Each all request / response is sent over same TCP connection
- (RTT) Roundtrip time
  - Time it takes for a small packet to to travel from client to server and then back to client
    - Packet propagation delays
    - Queuing delays
    - Packet processing delays

- HTTP uses persistent connections with pipelining

(HTTP)Hypertext transfer protocol

- HTTP message format
    - Two types of messages
        - Request message
            - Get/ somedir/page.html HTTP/1.1
                - Request line
                    - Three fields
                        - Method field
                            - Get
                                - Majority of HTTP request use this
                                - Used when browser request an object
                            - Post
                                - Uses entity body
                                - Forms and search
                            - Head
                                - Debugging
                                - Leaves out object
                            - Put
                                - Used in conjunction with web publishing tools
                                - Uploads object to specific path on server
                            - Delete
                                - Allows to delete an object on a web server
                        - Url field
                            - Insert url here
                        - HTTP version field
            - Host: www.someschool.edu
                - Header lines
                - Specifies host on which objects reside
            - Connection: close
                - Header lines
                - Doesnt bother with persistent connection
            - User-agent: mozilla/5.0
                - Header lines
                - Browser type
            - Accept-language: fr
                - Header lines
                - Preferred language

- (HTTP)Hypertext transfer protocol
    - HTTP message format
        - Two types of messages
            - Response message
                - HTTP/1.1 200 ok
                    - Status line
                        - Protocol version
                        - Status code
                            - 200 ok
                                - Request succeeded and information is returned
                            - 301 moved permanently
                                - Requested object has been permanently moved
                                - New url specified in location: of response message
                            - 400 bad request
                                - Generic error code indicating that the request could not be understood by the server
                            - 404 not found
                                - Requested document does not exist on this server
                            - 505 http version not supported
                                - Requested HTTP protocol version is not supported by the server
                        - Corresponding status message
                - Connection: close
                    - Header line
                - Date: tue, 09, aug, 2011 15:44:04 GMT
                    - Header line
                - Server: Apache/2.2.3 (centOS)
                    - Header line
                - Last modified: tue, 09, aug, 2011 15:44:04 GMT
                    - Header line
                - Content length: 6821
                    - Header line
                    - Number of bytes
                - Content type: text/html
                    - Header line
                    - What entity body contains
                - (Data, data, data)
                    - Entity Body
- Cookies
    - Four components
        - 1 Cookie header line in the HTTP response message

2 a cookie header line in the HTTP request message
3 a cookie file kept on the users end system and managed by the users browser
4 a back end database at the website

Web cache
- 1 browser establishes TCP connection to web cache and sends in HTTP request for the object to the web cache
- 2 web cache checks to see if it has a copy of the object stored locally if so web cache returs object within HTTP response message to client browser
- 3 web cache does not have object web cache opens a TCP connection to the origin server, then sends HATTP request for object into the cache to server TCP connection after receiving this request the origin server sends the object to the web cache
- 4 web cache receives HTTP response stores a copy in its local storage sends a copy to client browser

Total response time
- Sum of
  - LAN delay
  - Access delay
    - Delay between two routers
  - Internet delay
    - RTT that the internet side of access link
- Hit rates fraction of request satisfied by cache
- Content distribution networks
  - Make caches

Web caches
- Conditional GET
  - HTTP request must use GET method
  - HTTP request must include if-modified-since header

FTP
- User must provide a user identification and password to access remote account
- After the user can transfer from local system to remote system
- User interacts through FTP user agent
- User first provides hostname of remote host
  - Causes FTP client process in the local host to establish a TCP connection with FTP server process in the remote host
    - User provides id and password, sent over TCP as FTP commands
      - Server authorizes user, user free to manipulate data
- Uses two TCP connection to transfer files
  - Control connection
    - Used for sending control information between two host
      - Identification
      - Password
      - Commands to change remote directory
      - Commands to put and get files
    - Set up first
    - Persistent
  - Data connection
    - Used to actually send and transfer files
    - Non persistent
- Out of band
  - Sends control information in separate connection
  - HTTP is in band
- Server must maintain state

- Must associate control connection with specific user account and keep track of current users directory as the user wanders the remote directory tree
- Commands
  - USER username:
    - Used to send the user identification to server
  - PASS password:
    - Used to send the user password to the server
  - LIST:
    - Used to ask the sever to send back a list of all the files in the current remote directory the list of files is sent over data connection
  - RETR filename:
    - Used to retrieve (GET) a file from the current directory of the remote host this command causes the remote host to initiate a data connection and send data
  - STOR filename:
    - Used to store (PUT) a file into the current directory of remote host
- Reply
  - 331 Username ok, password required
  - 125 data connection already open; transfer starting
  - 425 can't open data connection
  - 452 error writing file

2.4

Electronic mail in the internet

Internet mail system

- User agent
  - To manipulate messages
- Mail servers
  - Core of email infrastructure
  - Every user has a mailbox assigned to them on the server
    - Authenticated with user and pass
- Simple mail transfer protocol
  - Principle application layer protocol for internet electronic mail
  - Uses TCP
    - Client side
      - Executes senders mail server
    - Server side
      - Executes on recipients mail server

SMTP

- Restricts body to 7-bit ASCII
- Process
  - Invokes user agent
  - Enters information
  - Composes message
  - Sends message
    - Placed in message queue
    - Client side sees message in message queue
    - Opens TCP connection to SMTP server running server side mail server
      - SMTP handshaking
      - Client sends message through TCP connection
      - Server side receives message
      - Places it in server side mailbox
        - Server side receives it in user agent
- Direct TCP connection between mail servers

- Establishes TCP connection between mail servers and passes recipients address in the process

- HTTP V SMTP
    - HTTP
        - Pull protocol
            - Loads info on a web server
                - Info pulled from web
            - TCP connection initiated by receiver of info
        - Does not have to be in 7 bit ASCII
        - Encapsulates each object in own HTTP response
    - SMTP
        - Push protocol
            - Sending mail server pushes file to receiving server
            - TCP connection initiated by sender
        - Must be in 7-bit ASCII
        - All message objects in one message
- SMTP mail message formats
    - Header and body separated by blank line
    - Keyword followed by : followed by value
    - Required key words
        - From:
        - To:
    - Mail access protocol
        - POP3 post office protocol version 3
            - Simple
            - Begins when TCP connection is created between user agent and mail server
                - Port 110
                - Three phases
                    - Authorization
                        - User agent sends user name and pass
                    - Transaction
                        - User agent retrieves messages
                        - Marks messages for deletion
                        - Removes deletion marks
                        - Obtains mail statistics
                    - update
                        - Occurs after client has issued quit command ending POP3 session
                        - Server deletes messages marked for deletion
                - User agent issues command
                - Two possible responses
                    - +ok
                        - Previous command was fine
                    - -ERR
                        - Used by server to indicate that something was wrong with the previous command
                - Authorization phase
                    - User<username>
                        - Command for inputing user
                    - Pass<password>
                        - Command for inputing pass

- Transaction phase
  - Download and delete mode
    - Messages deleted of server as manipulated
    - List
      - Shows messages
    - Retr
      - Retrieves messages
    - Dele
      - Deletes messages
  - Download and keep
    - Message kept on server
- IMAP internet mail access protocol
  - Associates each message with a folder
  - Keeps user state information
- HTTP
  - Web based email
  - The user agent and mail server interaction is HTTP

2.5

DNS the internets directory service

- Hostname
  - Website human recognition
- IP address
  - Four bytes
  - Each period separates on of the bytes expressed in decimal notation
- DNS domain name system
  - Distributed database implements in a hierarchy of DNS servers
  - Application layer protocol that allows hosts to query the distributed database
  - UDP
  - Port 53
- Process
  - Client side of DNS application
  - Browser extracts host names from url passes to client side DNS
    - DNS client sends query containing host name to DNS server
    - DNS client receives reply which includes IP address for host name
    - Browser receives IP address from DNS it initiates TCP connection to HTTP server on port 80 of that address
- Host aliasing
  - Canonical hostname
    - Not userfriendly host name
  - Alias host name
    - User friendly
  - DNS can retrieve both
- Mail server aliasing
- Load distribution
  - Set of IP addresses is associated with host name

How DNS works

- Invokes client side and indicates which hostname needs to be translated
  - DNS user host sends query to networks
  - UDP port 53
    - Receives reply with desired mapping
      - Passed to invoking application
- Problems with centralized design
  - A single point of failure
    - If DNS server crashes so does internet

- Traffic volume
    - Single DNS server would have to handle all DNS queries
- Distant centralized database
    - A single DNS cannot be close to all querying clients
- Maintenance

Distributed hierarchical database
- Three classes
    - Root DNS servers
        - Sends it to TLD
        - 13 root DNS servers
        - A-M
        - Most in north america
    - (TLD)Top-level domain DNS servers
        - Sends it to authoritative
        - Responsible for com edu org ect
    - Authoritative DNS servers
        - Sends it to correct ip address
        - Houses DNS records that map IP and host name
    - Local DNS servers
        - Sends to root

DNS caching
- Cache every interaction
- Discarded after short interval 2 days

DNS records and messages
- RRs resource records
    - Host name to IP
    - Each DNS reply message carries one or more sreource records
    - RRs four tuple
        - Name , value
        - Type
            - Type = A
                - Name
                    - Host name
                - Value
                    - IP
            - Type = NS
                - Name
                    - Domain
                - Value
                    - Host name of authoritative DNS server that knows how to obtain IP
            - Type = cname
                - Name
                    - Domain
                - Value
                    - Canonical hostname
            - Type= MX
                - Name
                    - domain
                - Value
                    - Canonical name of mail server
        - TTL
            - Time to live of the resource record
            - When should be discarded

- Only two kinds of DNS messages
    - query
    - reply
- Semantics of DNS
    - First 12 bytes
        - Header section
            - Identification
                - 16 bit number
            - flags
                - reply/ query flag
                    - 0 = query
                    - 1= reply
                - Authoritative
                - Recursion desired
            - 4 number fields
                - Keeps track of flags
            - Questions
                - Info about the query being made
                    - Name field of query
                    - Type field of query
            - Answers
                - Contains resource codes for originally queried name
            - Authority section
                - Records of other authoritative servers
            - Additional section
                - Contains other helpful records

2.6

- Peer to peer architecture
    - Scalability
        - Distribution time
            - Time it takes to get a copy of the file to all Npeers
            - For client server architecture
                - $Dcs= \max\{(NF)\backslash U_s, F\backslash d_{min}\}$
                - N=peers
                - F=file size
                - Servers upload rate Us
                - Dmin = peer with lowes download rate
            - For P2P
                - Dp2p = max{(F\Us),(F\Dmin),(NF\(Us+nsigmaUi(i=1)))}
    - Bit torrent
        - Tracker
            - Infrastructure node
                - Keeps track of all the peers participating in the torrent
                - New person gets persistent connected to x peers
                - Must choose which chunks to download and which to upload
                    - Rarest first
                        - Determines the chunk that are rarest among her neighbors
                        - Request those chunks first
                    - Trading
                        - Gives priority to peers that supply her at highest rate
                        - Sends chunks to 4 highest supply rate
                        - Recalculates every 10seconds
                        - These 4 are unchoked

Every 30s picks one more neighbor at random and sends it chunks
This one is optimistically unchoked
DHTs Distributed hash tables
Puts small bits of info on many locations

2.7
Socket programming
UDP
When socket is created identifier called port number is assigned to it
Sends Destination IP and destination Port number
Were creating a programs
Client reads a line of characters from keyboard and sends data to server
Server receives data and converts the characters to uppercase
The server sends the modified data to the client
Client receives the modified data and displays the line on its screen
Client program
UDPClient.py
Server program
UDPClient.py
Process
Client creates socket
clientsocket= socket(AF_INET, SOCK_DGRAM)
Server creates socket, port = x
serversocket= socket(AF_INET,SOCK_DGRAM)
Client Create datagram with server ip and port=x; sent datagram vis client socket
Server read UDPsegment from server socket
Write reply to server socket, specifying client address, portnumber
Client reads datagram from client socket
Closes client socket
UDPClient.py
From socket import *
Enables us to create a socket
Servername='hostname'
Serverport=12000
Connects us to server, provide either ip or hostname for server name
Also put it correct port so UDP socket knows where to go
Clientsocket= socket(socket.AF_INET,socket.SOCK_DGRAM)
Creates the actual clients socket
First parameter indicates the address family
Second parameter indicates that the socket is of type SOCK_DGRAM (UDP socket)
Message = raw_input('Input lowercase sentence:')
Raw_input() is built in function in python
When executed client prompted with words input lowercase sentence :
Data from keyboard put in the variable message
Clientsocket.sendto(message,(servername,serverport))
Sendto() attaches the destination address(servername, serverport) to the message
Sends resulting packet into the client socket
Goest to server
Modifiedmessage, serveraddress= clientsocket.recvfrom(2048)
Received data put into variable modifiedmessage packets source address put into serveraddre
Serveraddress includes server name and port
Print modifiedmessage
Prints modified message to users display
Clientsocket.close()

Closses socket and then terminates

UDPServer.py
From socket import *
Serverport = 12000
Serversocket = socket(AF_INET,SOCK_DGRAM)
Serversocket.bind((‘’,serverport))
- Assigns the port number to server socket

Print “the server is ready to receive”
While 1:
- Message, clientaddress = serversocket.recvfrom(2048)
  - When packet arrives data is put in variable message source address is put in Variable clientaddress
- Modifiedmessage = message.upper()
  - Takes line sent by client puts it through function upper and saves it to modifiedmessage
- Serversocket.sendto(modifiedmessage, clientaddress)
  - Attaches the client address to new message and sends

TCP
- Connection oriented
- Process
  - Server
    - Create socket, port=x, for incoming request:
      - Serversocket= socket()
    - Close connectionsocket
    - Wait for incoming connection request:
      - Connectionsocket= serversocket.accept()
  - Client
    - Create socket, connect to server ip, port=x:
      - Clientsocket=socket()
      - Connects to welcoming socket
      - Connectionsocket created in server
      - Sends request using client socket
        - Request read in server connectionsocket
        - Write reply in connection socket
          - Read reply in clientsocket
          - Close client socket
            - Close connectionsocket
- TCPclient.py
- From socket import *
- Servername= ‘servername’
- Serverport= 12000
- Clientsocket = (AF_INET,SOCK_STREAM)
  - Creates the client socket
  - Firstparameter indicates the underlying network
  - second parameter indicates that the socket is of type sock_stream (Tcp)
- Clientsocket.connect((servername,serverport))
  - Initiates TCP connection between client and server
  - Connect() is address of the server side of the connection after execution connection is established
- Sentence=raw_input(‘input lowercase sentence:’)
  - Obtains sentence from user
- Clientsocket.send(sentence)
  - Sends the string sentence through client socket into TCP connection client waits for response
- Modifiedsentence= clientsocket.recv(1024)

- When response arrives they are placed in modified sentence

- Print ‘from server:’, modifiedsentence
  - Print variable
- Clientsocket.close()
  - Closes connection

- TCPServer.py
- From socket import *
- Serverport = 12000
- Serversockt= socket(AF_INET,SOCK_STREAM)
  - Associate server prot numer with this socket
- Serversocket.bind((‘’,serverport))
  - Seversocket is welcoming socket
  - Wait and listen for knock on door
- Serversocket.listen(1)
  - Waits for tcp connection requests from clients
  - Max number of qued connections (at least one)
- Print ‘the server is ready to receive’
- While 1:
  - Connectionsocket, addr= serversocket.accept()
    - When client sends request program starts accept function
    - Creates a new socket called connection socket dedicated to client
    - Complete handshake creating TCP connection between client socket and connectionsockket
  - Sentence= connectionsocket.recv(1024)
  - Capitalizedsentence= sentence.upper()
  - Connectionsocket.close()
    - After sending data server connectionsocket closes but welcome socket still there

Introduction to transport layer
- Network between host
- Transport between processes
- Extending host to host delivery to proccess to process delivery is called transport layer multipexing and demultiplexing

Demultiplexing
- Delivers data in transport layer segment to correct socket
- Source port number

Multiplexing
- Gathering data chunks at source host
- Encapsulating them with header information to create segments and passing segments to network layer
- Destination port number

Port number
- 16 bit number
- 0-65535
- 0-1023 = well known port numbers (restricted)

Each segment has a source and destination portnumber

Connectionless (UDP) multiplexing and demultiplexing

- Clientsocket= socket(socket.AF_INET,socket.SOCK_DGRAM)
- When UDP socket created like this transport layer automatically assigns portnumber between 1024-65535 unused in the host
- Clientsocket.bind((‘’,19157))
  - Binds socket to hardcoded portnumber
- Client, auto assign port
- Server, hardcoded port
- UDP fully identified with two tuple
  - Destination IP address
  - Destination port number
  - Return address
    - Source IP address
    - Source port number

Connection oriented (TCP) multiplexing and demultiplexing

- TCP socket identified by four tuple
  - Source IP address
  - Source portnumber
  - Destination IPaddress
  - Destination port number
- Segments with different source address go to different sockets
- TCP server has a welcoming socket on port 12000
- TCP client creates socket and sends a connection establishment request segment
  - Clientsocket= socket(AF_INET, SOCK_STREAM)
  - Clientsocket.connect((servername,12000))
    - Connection establishment request is nothing more than TCP segment with destination portnnumber 12000 and a special connection establishment bit set in the TCP header and a source port number
    - Server receives request it locates the welcoming socket on portnumber 12000
    - Creates a new socket
      - Connectionsocket, addr = serversocket.accept()
        - Server at transport layer notes four values
          - Source port number
          - Source IP address
          - Destination port number
          - Destination IP address
        - All arriving segments that match this 4 tuple relation will be sent to to this socket

Connectionless transport UDP

- DNS uses UDP
  - Finer application level control over what data is sent and when
    - As soon as data passed to UDP packages the data and pass it to network layer
  - No connection establishment
    - No delay to establish connection
  - No connection state
    - Server can support many more UDP than TCP connections because of lack of persistent connection
  - Small packet header overhead
    - UDP has 8 bytes of overhead
    - TCP has 20 bytes

UDP checksum
Sender side performs 1's compliment of the sum of all the 16 bit words in segment
All over flow wrapped around
0's and 1's flipped
At receiver side all 16 bit words are added including check sum thus = 1----
Can only detect odd number errors
If error found sends warning or discards
Principles of reliable data transfer
Reliable data transfer protocol
Sending side
rdt_send()
data transfer protocol invoked
Will pass data to be delivered to upper layer at receiving side
Make_pkt()
Makes packet
Udt_sent()
Unreliable data transfer
Will send packet
Receiving side
rdt_rcv()
Data transfer protocol invoked
Receives data to upper layer
Extract ()
Deliver_data()
When protocol wants data this is invoked
Sends data to upper layer
FSM finite state-machine
Separate FSM for sender and receiver
Reliable data transfer over a channel with bit errors rdt2.0 stop and wait protocol
Positive acknowledgments
Control message that confirms data was received with no errors
Negative acknowledgments
Control message that informs sender the data was received incorrectly and should be sent again
ARQ(automatic repeat reQuest) protocols
Error detection
Mechanism needed to allow to detect when but errors have occurred
Extra bit necessary
Stored in packet checksum field
Receiver feedback
ACK
Positive feedback
NAK
Negative feedback
1 bit long
Retransmission
Packet with error at receiver is sent again by sender
&&isACK()
Positive ok
&&isNAK()
Negative bad
&&corrupt()
Send udt for retransmission with nak message

- &&notcorrupt()
    - Extracts and sends to upper layer
- If ACK or NAK corrupted
    - Send again
- Data packets have sequence number field
    - Checks to see if it has received the packet before
- Alt bit protocol
    - &&has_seq0()
        - sequence number 0
    - &&has_seq1()
        - Sequence number 1
- Double ACK = NAK

Reliable data transfer over a lossy channel with bit errors

- Count down timer
    - Start timer each packet sent
    - Respond to timer
    - Stop timer
- Start_timer
    - Starts timer when udt sent
- Stop_timer
    - Stops timer when ACK received
- Timeout
    - Sends packet again
- U_{sender} = (L/R)/(RTT+(L/R))

Pipelining

- Sequence numbers must be increased
- Sender receiver protocols may buffer
- Two approaches to pipeline errors
    - GBN Go-back-N
        - Allowed to transmit multiple packets without acknowledgment but constrained to have no more than N packets of unacknowledged packets in the pipeline
        - Base
            - Sequence number of oldest unacknowledged packet
        - Nextseqnum
            - Smallest unused sequence number
                - Yet to be sent
        - Four intervals in range of sequence number identifiable
            - [0,base-1]
                - Packets transmitted and acknowledged
            - [base,nextseqnum]
                - Packets sent but not yet acknowledged
            - [nextseqnum, base+n-1]
                - Packets that can be sent immediately
            - Seqnum=>Base+n
                - Cannot be used until an unacknowledged packet currently in the pipeline
                - Has been acknowledged
        - N
            - Window size of seqnum transmitted but not acknowledged
        - Range of numbers [0,2_K-1]
        - Refuse_data()
            - Refuses data

- Three types of events
  - Invocation from above
    - When rdt_send() is called sender checks if window is full
    - If window not full packet sent
  - Receipt of an ACK
    - Acknowledgment for a packet with sequence number n is cummulative acknowledgment indicating that all the packets with a sequence number up to and including n have been correctly received at the receiver
  - A timeout event
    - Lost or overly delayed packets all packets that have yet to be acknowledged will be sent again

SR Selective repeat

- Acknowledges each packet individually and sends ACK message
- Out of order packets are buffered until all lower sequenced packets are received
- Window size must be half of sequence number size
- Sender events
  - Data received from above
    - Checks next available sequence number for packet, if sequence number is within senders window, packet sent, otherwise buffered or returned to upper layer
  - Timeout
    - Each packet has timer
    - If packet lost single corresponding packet resent
  - ACK received
    - Marks packet as received provided it is in window
    - If equal to send_base the window base is moved forward to the unacknowledged packet with the smallest sequence number
    - If window moves and there are now untransmited packets in the new window packets sent
- Receiver events
  - Packet with sequence number in [rcv_base, rcv_base+N-1] is correctly received
    - Selective ACK packet is returned to sender if packet is new buffered
  - Packet with sequence number in [rcv_base-n, rcv_base-1]is correctly received
    - ACK is sent even though its already been previously acknowledged
  - Other wise do nothing

$d_{end\text{-}to\text{-}end}=N(L/R)$the probability that there are i active users (and 35-i) inactive users is: Pi = $C(35,i)*0.1^i*0.9^{35-i}$
$d_{nodal}=d_{proc}+d_{queue}+d_{trans}+d_{prop}$//Suppose there are n users, and each user is active with probability p. The probability that at any
$d_{end\text{-}to\text{-}end}=N(d_{proc}+d_{trans}+d_{prop})$//time instance, the number of active users is less than or equal to c is given by:
$D_{prop}=d/s$/ $Utilization_{sender}=(nL/R)/(RTT+(L/R))$/ P1+P2+P3+…+Pc = C(n,1)p(1-p)n-1 + C(n,2)p2(1-p)n-2+C(n,3)p3(1-p)n-3+...C(n,c)pc(1-p)n-c
distribute F to N clients- client-server approach- Dc-s > max{NF/us,,F/dmin}// p2p approach-DP2P > max{F/us,,F/dmin,,NF/(us + Σui)}

Htttp Request message --Three fields--Method field --Get--Majority of HTTP request use this --Used when browser request an object --Post--Uses entity body --Forms and search --Head ---Debugging --Leaves out object--Put --Used in conjunction with web publishing tools Uploads object to specific path on server --Delete --Allows to delete an object on a web server --Url field --Insert url here--HTTP version field Response message ==Status code --200 ok--Request succeeded and information is returned --301 moved permanently --Requested object has been permanently moved--New url specified in location: of response message--400 bad request--Generic error code indicating that the request could not be understood by the server --404 not found --Requested document does not exist on this server --505 http version not supported Requested HTTP protocol version is not supported by the server --FTP--Commands --USER username: ----Used to send the user identification to server --PASS password:--Used to send the user password to the server --LIST:--Used to ask the sever to send back a list of all the files in the current remote directory the list of files is sent over data connection --RETR filename:--Used to retrieve (GET) a file from the current directory of the remote host this command causes the remote host to initiate a data connection and send data --STOR filename:--Used to store (PUT) a file into the current directory of remote host --Reply --331 Username ok, password required --125 data connection already open; transfer starting 425 can't open data connection --452 error writing file DNS-- Name , value --Type --Type = A--Name --Host name --Value --IP--Type = NS Name --Domain --Value --Host name of authoritative DNS server that knows how to obtain IP--Type = cname --Name--Domain --Value --Canonical hostname --Type= MX--Name --domain --Value --Canonical name of mail server --TTL--Time to live of the resource record --When should be discarded --Cookies --1 Cookie header line in the HTTP response message --2 a cookie header line in the HTTP request message 3 a cookie file kept on the users end system and managed by the users browser --4 a back end database at the website --Web cache 1 browser establishes TCP connection to web cache and sends in HTTP request for the object to the web cache--2 web cache checks to see if it has a copy of the object stored locally if so web cache returs object within HTTP response message to client browser --3 web cache does not have object web cache opens a TCP connection to the origin server, then sends HATTP request for object into the cache to server TCP connection after receiving this request the origin server sends the object to the web cache--4 web cache receives HTTP response stores a copy in its local storage sends a copy to client browser PIPELINING--Four intervals in range of sequence number identifiable-- [0,base-1]--Packets transmitted and acknowledged --[base,nextseqnum]--Packets sent but not yet acknowledged --[nextseqnum, base+n-1]--Packets that can be sent immediately --Seqnum=>Base+n --Cannot be used until an unacknowledged packet currently in the pipeline --Has been acknowledged --N --Window size of seqnum transmitted but not acknowledged --Range of numbers [0,2k-1]--SR WINDOW SIZE HALF THAT OF SEQUENUM

Network architecture-5 layers – Application= Network applications reside, message, attachment ect ! Application layer packet known as a message (in book)-Transport =Transports application layer between application endpoints Two transport protocol, TCP- Connection oriented service,Guranteed delivery of application layer message and flow control !! Breaks long messages into shorter segments and provides congestion UDP-Connectionless service to its application
! !! No reliability, no flow control, no congestion control. ! Transport layer packet known as segment. Network=Packets known as datagrams
Transport layer Passes segment and destination addresses to network layer
IP protocol
! Defines fields in datagram as well as how end systems and routers act on these Routing protocol. Determines the route that datagrams take between source and destination. Link=Moves packet from one node to another
! network layer Passes datagram down to link layer ! Link layer moves to next node
! Link layer passes datagram up to network layer
! Link layer protocol (wifi, eithernet)
! Link layer packets known as frames.Physical=Moves individual bits within the frame from one node to the other.

Application architecture-Designed by application developer .Dictates how application is structured over various end systems Two predominant architecture paradigms .Client-server architecture Always-on host Server -Services request from other hosts called clients.Clients do not directly communicate with each other
! Server has a fixed IP address
! Datacenter
! ! Houses large number of host used to create a powerful virtual center Peer to peer architecture.Minimal to no reliance on datacenter
Exploits direct communication between pairs of connected hosts called peers
Peers
! Not owned by service providers
! Normal users
3 major challenges
! ISP friendly
=Residential ISP asymmetrical, more down stream than upstream traffic ! Security
= Highly distributes and open nature, thus difficult to secure
! Incentives
=Convincing users to volunteer bandwidth, storage, and computation
! ! resources to application
This is bit torrent, skype ect , Hybrids include aim, uses client-server to find IP and then uses peer to peer for rest

Processes communicating=1 Processes on two different end systems communicate by exchanging messages across computer network Sending process creates and sends messages into the network.Receiving process receives these messages and possibly responds by sending messages back. One labeled client process / downloading and other server process / uploading .Client server processes Exchange messages with a web server process .P2P =File is transferred in process in one peer to process in another peer

Socket interface between application layer and transport layer within a host. Referred to as application programming interface (API). Between application and network layer interface with which networks are built .Application developer has control on everything on the application side of the socket but little control. The transport side. 1 Choice of transport protocol (TCP or UDP) 2Ability to fix a few transport layer parameters such as max buffer and max segment size

Addressing processes- Two pieces of info needed for connection . address of the host
! IP address
! ! 32 bit number, uniquely identifies host
Identifier needed that specifies the receiving process int he destinations host ! Port number. Specific port numbers
! Webserver
! ! Port80
! Mail server process (SMTP)

Important things -Reliable data transfer - Guarantees that message will reach destination -Loss-tolerant applications - Not all the data may reach destinations -Bandwidth-sensitive applications - Force transport layer to secure a certain amount of bandwidth/ throughput-Elastic application - Uses whatever bandwidth/ throughput is available -Timing guarantees -Used for real time application -Security -Transport layer can provide security services

Internet does -Gives two transport protocols available TCP-Connection oriented and reliable data transfer service ! Connection oriented service.Has client and server exchanger transport layer control information with each other before the application level messages begin to flow .Prepares the client and server for onslaught of packets.After "handshake" TCP connection is said to exis between the sockets of two.processes -Full duplex connection .Can send messages at the same time . application finishes sending messages connection torn down.Reliable data transfer service.Can rely on TCP to deliver all data sent without error and in proper order when one application passes a stream of bytes into a socket it can count on TCP to deliver the same stream of bytes to the receiving socket with no missing or duplicate bytes .Congestion controlUDP .Lightweight ,Connectionless ,Unreliable data transfer ,No congestion control .

Application layer protocol ,How application processes on different end systems pass messages to each other

Defines ,The type of messages exchanged, for example request messages and response ,messages ,The syntax of the various message types such as the fields in the message and how the ,fields are delineated ,The semantics of the fields that is the meaning of the information in the fields ,Rules for determining when and how a process sends messages and responds to messages .

Hypertext transfer protocol(HTTP) -RFC 1945 RFC 2616 -Implemented in client and server -Web page -Consist of objects -Most consist of a base HTML file -If we have a page with HTML text and five JPEG -Base HTML file references the other objects in the page with the objects url -Each Url has two components -Host name of the server -Objects path name-http://www.someschool.edu/somedepartment/picture.gif -Someschool.edu is host name -/somedepartment/picture.gif is path name -Uses TCP -Stateless protocol -HTTP servers maintain no information about clients -Non-persistent connection -Each request/response pair is sent over a separate TCP connection -Process -1 HTTP client process initiates TCP connection to server someschool.edu on port

80 2 HTTP client sends an HTTP request message to the server via its socket -includes path name -3 HTTP server process receives the request message via socket retrieves -object /somedepartment/home.indx from storage encapsulates the object in an -HTTP response message and sends the response message to client -4 HTTP server process tells TCP to close connection -TCP doesnt close connection until it knows for sure client received info -5 HTTP client receives the response message TCP connection terminated message indicates the encapsulated object is an HTML file the client extracts -the file from the response message examines the HTML file and finds the references to the 10 jpegs -The first four steps are repeated for each of the referenced JPEG objects -Persistent connection

Each all request / response is sent over same TCP connection (RTT) Roundtrip time -Time it takes for a small packet to to travel from client to server and then back to client -Packet propagation delays -Queuing delays -Packet processing delays -HTTP uses persistent connections with pipelining

(HTTP)Hypertext transfer protocol! HTTP message format ,Two types of messages -Request message -Get/ somedir/page.html HTTP/1.1 -Request line -Three fields -Method field -Get -Majority of HTTP request use this -Used when browser request an object -Post -Uses entity body -Forms and search -Head-Debugging -Leaves out object -Put -Used in conjunction with web publishing tools -Uploads object to specific path on server -Delete -Allows to delete an object on a web server -Url field -Insert url here -HTTP version field

Host: www.someschool.edu -Header lines -Specifies host on which objects reside -Connection: close -Header lines -Doesnt bother with persistent connection--User-agent: mozilla/5.0 -Header lines -Browser type -Accept-language: fr -Header lines -Preferred language

HTTP)Hypertext transfer protocol! -HTTP message format - Two types of messages -Response message -HTTP/1.1 200 ok -Status line

Protocol version -Status code -200 ok -Request succeeded and information is returned -301 moved permanently -Requested object has been permanently moved -New url specified in location: of response message -400 bad request -Generic error code indicating that the request could -not be understood by the server -404 not found -Requested document does not exist on this server -505 http version not supported -Requested HTTP protocol version is not supported by -the server -Corresponding status message -Connection: close

Header line -Date: tue, 09, aug, 2011 15:44:04 GMT -Header line -Server: Apache/2.2.3 (centOS) -Header line -Last modified: tue, 09, aug, 2011 15:44:04 GMT -Header line -Content length: 6821! -Header line -Number of bytes -Content type: text/html -Header line -What entity body contains -(Data, data, data) -Entity Body -Cookies -Four components -1 Cookie header line in the HTTP response message 2 a cookie header line in the HTTP request message -3 a cookie file kept on the users end system and managed by the users browser -4 a back end database at the website Web cache -1 browser establishes TCP connection to web cache and sends in HTTP request for the object to the -web cache -2 web cache checks to see if it has a copy of the object stored locally if so web cache returs object -within HTTP response message to client browser -3 web cache does not have object web cache opens a TCP connection to the origin server, then -sends HATTP request for object into the cache to server TCP connection after receiving this request -the origin server sends the object to the web cache -4 web cache receives HTTP response stores a copy in its local storage sends a copy to client ! -browser -Total response time -Sum of -LAN delay -Access delay -Delay between two routers -Internet delay -RTT that the internet side of access link -Hit rates fraction of request satisfied by cache -Content distribution networks -Make caches -Web caches-FTPConditional GET HTTP request must use GET method HTTP request must include if-modified-since header -User must provide a user identification and password to access remote account -After the user can transfer from local system to remote system -User interacts through FTP user agent -User first provides hostname of remote host -Causes FTP client process in the local host to establish a TCP connection with FTP server -process in the remote host -User provides id and password, sent over TCP as FTP commands -Server authorizes user, user free to manipulate dataUses two TCP connection to transfer files -Control connection -Used for sending control information between two host !-Identification-Password -Commands to change remote directory -Commands to put and get files -Set up first –Persistent-Data connection -Used to actually send and transfer files -Non persistent -Out of band -Sends control information in separate connection -HTTP is in band

Server must maintain state Must associate control connection with specific user account and keep track of current users-directory as the user wanders the remote directory tree Commands USER username: -Used to send the user identification to server -PASS password: -Used to send the user password to the server -LIST: -Used to ask the sever to send back a list of all the files in the current remote directory -the list of files is sent over data connection -RETR filename: -Used to retrieve (GET) a file from the current directory of the remote host this command -causes the remote host to initiate a data connection and send data -STOR filename: -Used to store (PUT) a file into the current directory of remote host Reply -331 Username ok, password required -125 data connection already open; transfer starting =425 can't open data connection

452 error writing file

Electronic mail in the internet -Internet mail syst- SMTP -User agent -To manipulate messages -Mail servers -Core of email infrastructure -Every user has a mailbox assigned to them on the server -Authenticated with user and pass -Simple mail transfer protocol -Principle application layer protocol for internet electronic mail -Uses TCP -Client side -Executes senders mail server! -Server side-Executes on recipients mail server -Restricts body to 7-bit ASCII -Process -Invokes user agent -Enters information -Composes message -Sends message -Placed in message queue -Client side sees message in message queue -Opens TCP connection to SMTP server running server side mail server-SMTP handshaking -Client sends message through TCP connection -Server side receives message -Places it in server side mailbox -Server side receives it in user agent -Direct TCP connection between mail servers

Establishes TCP connection between mail servers and passes recipients address in the process -HTTP V SMTP -HTTP -Pull protocol -Loads info on a web server -Info pulled from web -TCP connection initiated by receiver of info -Does not have to be in 7 bit ASCII -Encapsulates each object in own HTTP response -SMTP -Push protocol -Sending mail server pushes file to receiving server -TCP connection initiated by sender -Must be in 7-bit ASCII -All message objects in one message -SMTP mail message formats -Header and body separated by blank line -Keyword followed by : followed by value -Required key words -From: -To:-Mail access protocol -POP3 post office protocol version 3 -Simple -Begins when TCP connection is created between user agent and mail server -Port 110 -Three phases -Authorization -User agent sends user name and pass -Transaction -User agent retrieves messages -Marks messages for deletion -Removes deletion marks -Obtains mail statistics -update -Occurs after client has issued quit command ending POP3 session -Server deletes messages marked for deletion -User agent issues command -Two possible responses +ok-Previous command was fine --ERR -Used by server to indicate that something was wrong with

the previous command -Authorization phase -User<username> -Command for inputing user -Pass<password> -Command for inputing pass -Transaction phase -Download and delete mode -Messages deleted of server as manipulated -List -Shows messages -Retr -Retrieves messages - Dele -Deletes messages -Download and keep -Message kept on server

IMAP internet mail access protocol -Associates each message with a folder -Keeps user state information -HTTP -Web based email -The user agent and mail server interaction is HTTP -2.5 -DNS the internets directory service -Hostname -Website human recognition -IP address -Four bytes -Each period separates on of the bytes expressed in decimal notation -DNS domain name system -Distributed database implements in a hierarchy of DNS servers -Application layer protocol that allows hosts to query the distributed database -UDP -!Port 53 -Process -Client side of DNS application -Browser extracts host names from url passes to client side DNS -DNS client sends query containing host name to DNS server -DNS client receives reply which includes IP address for host name -Browser receives IP address from DNS it initiates TCP connection to HTTP server on -port 80 of that address -!Host aliasing -Canonical hostname -Not userfriendly host name -Alias host name -User friendly -DNS can retrieve both -Mail server aliasing-Load distribution -Set of IP addresses is associated with host name -How DNS works -Invokes client side and indicates which hostname needs to be translated -DNS user host sends query to networks -UDP port 53 -!-Receives reply with desired mapping -Passed to invoking application -Problems with centralized design -A single point of failure -If DNS server crashes so does internet -Traffic volume -Single DNS server would have to handle all DNS queries -Distant centralized database -A single DNS cannot be close to all querying clients -Maintenance -Distributed hierarchical database -Three classes -Root DNS servers -Sends it to TLD -13 root DNS servers-A-M-Most in north america -(TLD)Top-level domain DNS-servers -!Sends it to authoritative -Responsible for com edu org ect -Authoritative DNS servers -Sends it to correct ip address -Houses DNS records that map IP and host name -Local DNS servers -Sends to root -DNS caching -Cache every interaction -Discarded after short interval 2 days -DNS records and messages -RRs resource records -Host name to IP -Each DNS reply message carries one or more sreource records-RRs four tuple -Name , value -Type-Type = A! -Name ! -Host name -Value P-Type = NS -Name -Domain -Value -Host name of authoritative DNS server that knows how to obtain IP -Type = cname -Name -Domain -Value -Canonical hostname ! -Type= MX -Name -domain -Value -Canonical name of mail server ! -TTL-Time to live of the resource record -When should be discarded -Only two kinds of DNS messages p-query-reply -Semantics of DNS- First 12 bytes Header section -Identification ! -16 bit number -flags-reply/ query flag -0 = query - 1= reply -Authoritative Recursion desired-4 number fields -Keeps track of flags -Questions -Info about the query being made -Name field of query -Type field of query -Answers -Contains resource codes for originally queried name -Authority section -Records of other authoritative servers -Additional section -Contains other helpful records

Peer to peer architecture -Scalability -Distribution time -Time it takes to get a copy of the file to all Npeers -For client server architecture -Dcs= max{(NF)\Us,F\dmin} N=peers -F=file size -Servers upload rate Us -Dmin = peer with lowes download rate -For P2P -Dp2p -max{(F\Us),(F\Dmin),(NF\(Us+nsigmaUi(i=1)))} -Bit torrent -Tracker Infrastructure node -Keeps track of all the peers participating in the torrent -New person gets persistent connected to x peers -Must choose which chunks to download and which to upload -Rarest first -Determines the chunk that are rarest among her neighbors -Request those chunks first -Trading -Gives priority to peers that supply her at highest rate -Sends chunks to 4 highest supply rate -Recalculates every 10seconds -These 4 are unchoked -Every 30s picks one more neighbor at random and sends it chunks -This one is optimistically unchoked -DHTs Distributed hash tables -Puts small bits of info on many locations

Socket programming -UDP -When socket is created identifier called port number is assigned to it

Sends Destination IP and destination Port number -Were creating a programs -Client reads a line of characters from keyboard and sends data to server -Server receives data and converts the characters to uppercase -The server sends the modified data to the client -Client receives the modified data and displays the line on its screen -Client program -UDPClient.py -Server program -UDPClient.py -ProcessClient creates socket -clientsocket= socket(AF_INET, SOCK_DGRAM) -Server creates socket, port = x serversocket= socket(AF_INET,SOCK_DGRAM) -Client Create datagram with server ip and port=x; sent datagram vis client socket -Server read UDPsegment from server socket -Write reply to server socket, specifying client address, portnumber -Client reads datagram from client socket -Closes client socket UDPClient.py -From socket import * -Enables us to create a socket -Servername='hostname'-Serverport=12000 -Connects us to server, provide either ip or hostname for server name -Also put it correct port so UDP socket knows where to go -Clientsocket=socket(socket.AF_INET,socket.SOCK_DGRAM) -Creates the actual clients socket -First parameter indicates the address family -Second parameter indicates that the socket is of type SOCK_DGRAM (UDP socket)-Message = raw_input('Input lowercase sentence:') -Raw_input() is built in function in python -When executed client prompted with words input lowercase sentence : Data from keyboard put in the variable message -Clientsocket.sendto(message, (servername,serverport)) -Sendto() attaches the destination address(servername, serverport) to the message -Sends resulting packet into the client socket -Goest to server -Modifiedmessage, serveraddress= clientsocket.recvfrom(2048) -Received data put into variable modifiedmessage packets source address put into serveraddre -Serveraddress includes server name and port-Print modifiedmessage -Prints modified message to users display -Clientsocket.close() Closses socket and then terminates

-UDPServer.py -From socket import * Serverport = 12000 Serversocket = socket(AF_INET,SOCK_DGRAM)

Serversocket.bind(('',serverport)) -Assigns the port number to server socket-Print "the server is ready to receive" While 1: Message, clientaddress = serversocket.recvfrom(2048) -When packet arrives data is put in variable message source address is put in -Variable clientaddress Modifiedmessage = message.upper() -Takes line sent by client puts it through function upper and saves it to modifiedmessage -Serversocket.sendto(modifiedmessage, clientaddress) -Attaches the client address to new message and sends

Connection oriented-Process -Server –Create-socket, port=x, for incoming request: -Serversocket= socket() -Close connectionsocket -Wait for incoming connection request: -Connectionsocket= serversocket.accept() Client Create socket, connect to server ip, port=x: Clientsocket=socket()

Connects to welcoming socket Connectionsocket created in server Sends request using client socket

Request read in server connectionsocket Write reply in connection socket Read reply in clientsocket Close client socket -Close connectionsocket TCPclient.py From socket import * Servername='servername'

Serverport= 12000 -Clientsocket = (AF_INET,SOCK_STREAM) Creates the client socket ! Firstparameter indicates the underlying network -second parameter indicates that the socket is of type sock_stream (Tcp) Clientsocket.connect((servername,serverport)) Initiates TCP connection between client and server -Connect() is address of the server side of the connection after execution connection is established Sentence=raw_input('input lowercase sentence:') Obtains sentence from user

Clientsocket.send(sentence) Sends the string sentence through client socket into TCP connection client waits for response Modifiedsentence= clientsocket.recv(1024)

3.5
Connection oriented transport TCP
TCP
- Full duplex service
  - Data can flow both ways
- Point-to-point
  - Between single sender and single receiver
- clientSocket.connect((severName,serverPort))
  - Establishes TCP connection
  - Three way handshake
- Data goes to client send buffer
  - TCP will grab chunks and send
- (MSS) maximum segment size
  - The maximum amount of data that can be put in one segment
  - Determined by the length of the (MTU) maximum transmission unit
    - (MTU) maximum transmission unit
      - Largest link layer frame that can be sent by local host
  - MSS = MTU plus TCP header bytes

TCP segment structure
- Source port number
  - 16 bits
- Destination port number
  - 16 bits
- Sequence number field
  - 32 bits
  - Used with acknowlegment number field for relaible data transfer
- Acknowledgment number field
  - 32 bits
  - Used with sequence number field for reliable data transfer
- Receive window
  - 16 bits
  - Flow control
  - Number of bytes receiver is willing to accept
- Header length
  - 4 bits
  - Specifies length of header in 32 bit words
- Options field
  - Variable length
  - Used when sender and receiver negotiate MSS or as window scaling factor in high speed networks

- Flag field
  - 6 bits
    - Ack bit
      - Indicates that value carried in acknowledgment field is valid
    - RST
    - SYN
    - FIN
      - Are used for tear down an set up of connection
    - PSH
      - Receiver should pass data directly to upper layer immediatly
    - URG
      - Urgent

Sequence numbers and acknowledgment numbers

- TCP views data as a byte stream
- Thus each sequence number corresponds to the where in the byte stream the first byte of the segment actually is
- The ack number is the next expected sequence number by the server thus if client A sends a data segment with seq num 500-699 the ack num Server A would send to client A is 700
- TCP does culminitive acknowledgment, acknowledging in sequence order
  - Thus if client A sends three segments 0-199, 200-399, 400-599 and server A drops the second segment and only receives segments 0-199 and 400-599 it will send to client A a segment with ack number 200
- Remember each segment regardless if its from server or client has an ack and sequence field

Round trip time estimation and timeout

- SampleRTT
  - Time between when the segment is sent the acknowledgment is received
  - New value once every RTT
    - Never computed for a value that has been retransmitted
- EstimatedRTT
  - Averages of SampleRTT
  - EstimatedRTT = (1- a) x EstimatedRTT + a x SampleRTT
    - Recommended a = .125
    - Weighted combination of previous value of EstimatedRTT and SampleRTT
    - (EWMA) exponential weighted moving average
      - New SampleRTT given more weight
- DevRTT
  - Estimate of how much Sample RTT typically deviates from EstimatedRTT
  - DevRTT = (1-b) x DevRTT + b x |SampleRTT - EstimatedRTT|
    - Recommended b = .25

Setting and managing the retransmission timeout interval

- TimeoutInterval
  - Length of time until retransmission
  - TimeoutInterval = EstimatedRTT + 4 x DevRTT
  - Initial TimeoutInterval of one second is recommended
    - If time out occurs on first instance than TimeoutInterval is doubled, this goes until a ack is received, then the formulas kick in
  - When a timeout occurs it sets TimeoutInterval to double the previous one

Reliable data transfer
- Timer starts on sendbase sequence number
- If time out it sends the sendbase sequence number packet again
- When it receives an ack it checks the ack number
  - If the number = sendbase it increments sendbase by one and restarts timer thus the segment with the smallest sequence that has not been confirmed by an ack is now the sendbase
  - If the number > sendbase it makes sendbase = acknumber+1 and restarts the timer thus making the segment with the smallest sequence that has not been confirmed by an ack the sendbase

Fast retransmit
- When receiver detects a gap in data stream it sends a duplicate ack that is the same as the last ack it sent, it does so for each segment it has received
- When the sender receives three duplicate acks it understands that this segment has been lost and retransmits it before the timer runs out

Flow control
- Buffer has limited size, thus to prevent buffer overflow this is used
- Host B (receiver)
  - *RcvBuffer*
    - Receiver Buffer in a host
  - *Rwnd*
    - Receiver window
      - Room left in buffer
  - *LastByteRead*
    - Number of the last byte in the data stream read from the buffer by the application process (of receiver of files)
  - *LastByteRcvd*
    - Number of the last byte in the data stream that has arrived from the network and has been placed in the receiver buffer at receiver of data
  - Because TCP is not permitted to overflow
    - *LastByteRcvd - LastByteRead <= RcvBuffer*
  - Receive window is set to spare room in buffer
    - *Rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]*
  - Current value of *Rwnd* is placed in receiver window field of every segment
    - Initially *Rwnd = RcvBuffer*
- Host A (sender)
  - Keeps track of
    - *LastByteSent*
      - Number of the last byte in the data stream sent by the sender
    - *LastByteAcked*
      - Number of the last byte acked by the receiver that the sender has received
    - *LastByteSent - LastByteAcked* = amount of unacknowledged data that is in connection
    - To not overflow the *RcvBuffer* in host B host A makes sure that
      - *LastByteSent - LastByteAcked<= Rwnd*
    - When host B Rwnd = 0 then the sender sends one byte segments to update itself on the condition of the receiver buffer

TCP connection management
- Step 1
  - Client side sends special TCP segment to server side
    - Just a header
      - SYN bit set to 1
      - Client randomly chooses an isn number
- Step 2
  - Sever side receives TCP SYN segment and extracts it
    - Allocates necessary TCP buffers
    - Sends back a TCP segment with no app data just header
    - TCP SYNACK
      - SYN bit set to 1
      - Ack field set to clien_isn+1
      - Random sequence number chosen
- Step 3
  - Client receives SYNACK segment
    - Allocates TCP buffers
    - Sends back TCP segment
      - May carry app data
      - SYN set to zero
      - ACK field set to server_isn +1

TCP states of existence
- Client side
    - Closed
        - Client application initiates TCP connection
        - Sends SYN
    - SYN_start
        - Receives SYNACK
        - Sends ACK
    - Established
        - Send FIN
        - Client application initiates close connection
    - FIN_WAIT_1
        - Receive ACK
        - Send nothing
    - FIN_WAIT_2
        - Receive FIN
        - Send ACK
    - TIME_WAIT
        - Wait 30 seconds
    - Closed
- Server side
    - Closed
        - Server application creates a listen socket (welcoming)
    - Listen
        - Receives SYN
        - Sends SYNACK
    - SYN_RCVD
        - Receive ACK
        - Send nothing
    - Established
        - Receive FIN
        - Send ACK
    - Close_wait
        - Send fin
    - Last_ack
        - Receive ACK
        - Send nothing
    - Closed

Congestion control
- Scenario 1
    - Two senders a router with infinite buffers
        - No retransmission
        - Because tcp is full duplex the throughput it R/2
        - As sending rate approaches R/2 the average delay increases exponentially
            - Queuing delays
- Scenario 2
    - Two senders and a router with finite delay
        - Packets dropped when arriving at an already full buffer
        - Sending rate

- Original data into the network
- Offered load
  - The original data and retransmission into the network
- When offered load = R/2
  - At this offered load R/3 = original data
  - And R/2- R/3 = retransmission load
    - Retransmission delays
  - The router will forward each packet twice thus
    - The throughput = R/4

Scenario 3
- Four senders, routers with finite buffers and multihop paths
  - Two hop paths
  - As traffic increases throughput eventually goes to 0
  - When a packet is dropped along a path the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it is dropped ends up having beed wasted

Methods
- End-to-end congestion control
  - Network layer gives no support
  - Based on observed network behavior
    - TCP segment loss indication of network congestion
    - TCP decreases its window size accordingly
- Network assisted congestion control
  - Routers provide feedback to sender regarding congestion state in network
    - Can be single bit with yes/no congestion
  - Used in Asynchronous transfer mode (ATM) available bit rate (ABR)
    - Routers tell sender the transmission rate it can support
  - Two ways of network informing the sender
    - Choke packet
      - Says in single bit I am congested
    - Modification
      - Router marks a bit in packet header from sender to receiver
        Receiver receives the modified packet and informs the sender that network is congested

TCP congestion control
- (Cwnd) Congestion window
  - Restraints the amount of packets that the sender sends
  - Amount of unacknowledged data at the sender may not exceed minimum of cwnd and rwnd
  - Usually the rwnd is so large it is ignored thus the amount of unacknowledged data at sender cannot exceed min cwnd or
    - Lastbytesent-Lastbyteacked<= min cwnd
- Sender rate
  - Cwnd/RTT
- Selfclocking
  - Uses acknowledgments to increases its cwnd

- TCP congestion-control algorithm
  - Three major components
    - (essential) Slow start
      - When TCP connection begins cwnd is set to MSS
      - The initial sending rate is thus MSS/RTT
      - For every consecutive ACK message the sender receives it increments the cwnd by one MSS
        - Thus it doubles every RTT because every additional MSS it sends will receives an additional ACK which will in turn increases the cwnd by an additional MSS
      - If there is a loss event(which indicates congestion) by a timeout the TCP sender sets the value of ssthresh(slow start threshold) to half the cwnd at the loss event
      - When the cwnd = sstresh slow start ends and congestion avoidance starts
      - If three duplicate ACKs are received by sender TCP performs fast retransmit and enters fast recovery state
    - (essential) Congestion avoidance
      - When this start cwnd is half of what it was before the last loss event In this state the cwnd is incremented by MSS every RTT, slowly increasing at a linear pace
      - If a timeout occurs the cwnd goes to 1MSS and the ssthresh is set to half cwnd
      - If a a triple ACK is received then the cwnd is set to half the cwnd + 3 (because of the 3 acks) and the ssthresh is set to half the cwnd and then it enters fast recovery state
    - (optional) fast recovery
      - The value of cwnd is increased by 1MSS for every duplicate ACK received for a missing segment that caused TCP to enter fast recovery state
      - When the ack for the missing segment is received TCP enters congestion avoidance mode after deflating the cwnd
      - If a time out occurs fast recovery transitions to slow start the value of cwnd is set to 1mss and the ssthresh is set to half cwnd
    - TCP tahoe
      - Sends the cwnd back to 1mss for every loss event including triple ack and timeout
    - TCP Reno
      - Incorporates fast recovery
        - Cutting the cwnd in half plus 3
    - TCP congestion control is often refereed to as additive-increase, multiplicative decrease (AIMD)
    - Average throughput of a TCP connection
      - .75Cwnd/RTT = average throughput

Chapter 4 network layer
- Network layer has 3 components
  - IP protocol
  - Routing protocol
  - ICMP

4.1.1 forwarding and routing
- Forwarding
  - Router-local action of transferring a packet from an input link interface to the appropriate output link interface
- Routing
  - Network wide process that determines the end to end paths that packets take from source to destination
- Every router has a forwarding table
  - Router forwards packet by examining the value of a field in the arriving packet header
    - Uses this value to index into the routers forwarding table
    - The value stored in the forwarding table indicates the routers outfoing link interface to which the packet is to be forwarded
- Packet switch
  - General packet-switching device that transfers a packet from input link interface to output link interface
    - Link-layer switches
      - Base their forwarding decision on values in the fields of link layer frame
        - Link layer (layer 2) devices
    - Routers
      - Base their forwarding decisions on the value of network layer field
        - Network layer (layer 3 (devices
    - Connection set up
      - ATM, frame relay, MPLS

4.1.2 network service models
- Network service model
  - Defines characteristics of an end to end transport of packets
  - Possible Services included
    - Guaranteed delivery
      - This service guarantees that the packet will eventually arrive at its destination
    - Guaranteed delivery with bounded delay
      - this service not only guarantees delivery of every packet but delivery within a specified host to host delay bound (like less than this much time)
  - Possible services provided to a flow of packets between destination
    - In-order packet delivery
      - Guarantees that packets arrive at the destination in the order they were sent
    - Guaranteed minimal bandwidth
      - Sets up restriction where it will not lose any packets and they will arrive within a pre-specified host to host delay if the bit rate stays below a specified bit rate
    - Guaranteed maximum jitter
      - Guarantees that amount of time between transmission of two successive packets is equal to amount of time between their receipt at destination , or that this amount of time changes by a pre-specified amount

- Security services
  - Encrypts payloads of datagrams between source and destination

- Internet provides a single service
  - Best-effort service
    - No service

4.4 the internet protocol (IP) : forwarding and addressing in the internet

- Two versions of IP
  - Ipv4
    - Most used today
  - Ipv6

4.4.1 Datagram format

- Network layer is a datagram
- Setup ----- 20 bytes in header with no options
  - Version number
    - 4bits
  - Header length
    - 4 bits
    - Contains the length of the header since multiple options are available
    - Most common is 20 byte headers
  - Type of service
    - 8 bits
    - Allows to distinguish between different types of datagrams
  - Datagram length
    - 16 bits
    - Total length of the datagram in bytes
    - Typical length 1500 bytes
  - Identifier, flags, fragmentation offset
    - These fields have to do with fragmentation
    - Ipv6 has no fragmentation
  - Time-to-live
    - Insures that datagram does not circulate network indefinitely
    - At each router it is decremented by one until it reaches 0 and then is dropped
  - Protocol
    - Use only when datagram reaches its final destination
    - Indicates the specific transport layer protocol this datagram should be passed to
    - Value =6 passed to TCP
    - Value = 17 passed to UDP
  - Header checksum
    - Aids router in detecting errors in received IP datagram
    - ONLY CHEAKS HEADER
    - Computed by treating each 2 bytes in header as a number and summing these numbers using 1's compliments
    - Check sum is stored, if computed checksum and stored checksum do not match Router discards the datagram
    - Checksum is changed at every router
  - Source and destination IP addresses
    - When created datagram has the source IP address and destination IP address
  - Options

- Allows IP header to be extended
- Not in Ipv6

- Data ---- self explanatory

- IP data fragmentation
    - Since the MTU between routers differs when sending a large file to a link with a smaller MTU the datagram is fragmented into smaller datagrams called fragments
    - Must be reassembled before they reach transport layer at destination
    - Reassembled at host
    - All info put in the Identifier, flags, fragmentation offset field of header
    - Must be fragmented all but last fragment into datagrams with data size divisible by 8
        - Identifier
            - the datagram identification number
        - Flag
            - Set to 0 if last fragment
            - Set to 1 otherwise
        - Fragmentation offset
            - Where the fragment fits in the datagram
            - For the first one it is 0 then it is the total amount of data without header divided by 8
    - If one or more fragments not received entire datagram is discarded

4.4.2 IPv4 addressing

- Interface
    - Boundary between host and physical link
- Host has one interface
- Router has multiple interfaces
- Each interface has its own IP address
- Each IP address is 32 bits long, 4 bytes
    - 2^32 possible addresses
- Dotted decimal notation
    - Each byte is written in decimal form and separated by period
- Subnet
    - A group of interfaces that share a portion of their left most IP address
        - The degree of subnet is denoted by how many bits they share
            - Written at the end with a backslash followed by the amount of bits they share
- Classless interdomain routing (CIDR)
    - Divided into two parts
        - A.b.c.d/x
        - The x most significant bits on an address constitute the network portion of IP
            - Prefix (network prefix)
                - The left most bit they share in common
        - When routing
            - When routing only the prefix is taken into account before they reach the subnet
            - Within the subnet the rest of the numbers is what is take into account
- Classful addressing
    - Constrained using 8, 16, 24 bit length
- When host in subnet sends datagram to 255.255.255.255 everyone in subnet receives

- Dynamic host configuration protocol (DHCP)
  - Allows a host to be allocated to a IP address automatically
  - Four steps for newly arriving host
    - DHCP server discovery
      - Host sends a DHCP discovery message
        - UDP packet to port 67
        - Creates an IP datagram with dest 255.255.255.255 and source 0.0.0.0
        - This is passed to link layer and sent to all nodes attached to subnet
    - DHCP server offer(s)
      - Server responds with DHCP offer message
        - Broadcasted to all nodes on subnet dest 255.255.255.255
        - Multiple servers can send this simultaneously thus the host can choose
        - Offer message contains the transaction Id of the received discovery message, proposed Ip address, network mask and IP address lease time
          - IP Address lease time
            - Length of time ip address with be valid
    - DHCP request
      - Newly arriving client will choose from among one or mores server offers and respond to its selected offer with a DHCP request message
        - DHCP request message
          - Echoes back configuration parameters
    - DHCP ACK
      - Server responds with
        - DHCP ACK message
          - Confirms parameters
- Network address translation (NAT)
  - Creates private realm
    - Address only have meaning within the realm
    - NAT looks like a single IP address to outside world
    - Uses NAT translation table to differentiate traffic between different users
      - NAT translation tables
        - Contain ip address and port numbers
- UPnP
  - Universal plug and play
    - Allows host to discover and configure nearby NAT allows TCP and UDP connections between direct host circumventing the NAT
    - NAT traversal

4.4.3 internet control message protocol (ICMP)

- Used by host and routers to communicate network layer information to each other
  - ICMP messages
    - have a type and a code field
    - Contain header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated

ICMP TYPE	CODE	description
0	0	echo reply (ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

4.4.4

- Ipv6
  - Format
    - Expanded addressing capabilities
      - IP address size 128 bits
      - Anycast address
        - Allows datagram to be delivered to any one of a group of hosts
    - Streamlined 40 byte header
      - Fixed length
    - Flow labeling and priority
      - Labeling of packets belonging to particular flows which the sender requests special handling such as non-default quality of service or real-time service

- ipv6
  - Structure
    - Version
      - 4 bits
    - Traffic class
      - 8 bit
      - Same as TOS field in ipv4
        - Allows to distinguish between different types of datagrams
    - Flow label
      - 20 bit field
        - Identify a flow of datagrams
    - Payload length
      - 16 bit
      - Treated as unsigned integer gives total amount of bytes in datagram
    - Next header
      - Provides protocol its going to in transport layer (UDP TCP)
    - Hop limit
      - Decremented by one each transmission once 0 reached datagram dropped
    - Source and destination addresses
      - 128 bit address
    - Data
      - Obvious
- Tunneling
  - In transition where two IPv6 routers are separated by ipv4 they take the ipv6 including header and put it into an ipv4 datagram
  - The destination for the ipv4 is set to the next router ipv6 compatible

$d_{end-to-end}=N(L/R)$the probability that there are i active users (and 35-i) inactive users is: $P_i = C(35,i)*0.1^i*0.9^{35-i}$
$d_{nodal}=d_{proc}+d_{queue}+d_{trans}+d_{prop}$//Suppose there are n users, and each user is active with probability p. The probability that at any
$d_{end-to-end}=N(d_{proc}+d_{trans}+d_{prop})$//time instance, the number of active users is less than or equal to c is given by:
$D_{prop}=d/s$/ $Utilization_{sender}=(nL/R)/(RTT+(L/R))$/ P1+P2+P3+…+Pc = C(n,1)p(1-p)n-1 + C(n,2)p2(1-p)n-2+C(n,3)p3(1-p)n-3+...C(n,c)pc(1-p)n-c
distribute F to N clients- client-server approach- Dc-s > max{NF/us,,F/dmin}// p2p approach-DP2P > max{F/us,,F/dmin,,NF/(us + Σui)}
$D_x(y)= \min_v\{c(x,v)+D_v(y)\}$