COMP 445 - Midterm Notes

# Chapter 1

#### The Network Edge

packet transmission delay = time needed to transmit L-bit packet into link = L(bits)/R(bits/sec)

packets of length L

transmission rate R

#### The network core

hosts break application-layer message into packets

**Packet-Switching: store and forward**

takes L/R seconds to transmit (push-out)

entire packet must arrive at router before it can be transmitted

end-to-end delay = 2L/R assuming no propagation delay

**Packet-Switching: queueing delay, loss**

packets will queue, wait to be transmitted

packets can be dropped if memory buffer fills up

routing: determines source-destination route taken by packets

forwarding: move packets from router’s input to appropriate router output

**Circuit-switching**

dedicated end-to-end resources between source and destination

no sharing

common in traditional phone networks

FDM: Frequency

TDM: Time

#### Delay, Loss, Throughput

loss and delay occur because of packet queues in router buffers

arrival rate to link temporarily exceeds output link capacity

packets queue, wait for turn

**Four sources of packet delay**

nodal processing(dproc)

typically < msec

queueing delay(dqueue)

time waiting at output link for transmission

transmission delay(dtrans)

L: packet length (bits)

R: link bandwidth (bps)

L/R

propagation delay(dprop)

d: length of physical link

s: propagation speed in medium (~2x10^8 m/sec)

d/s

**Throughput**

rate(bits/time) at which bits transferred between sender/receiver

#### Protocol Layers, Service Models

Layering provides structure and organization of complex systems

Provides relationships of complex system’s pieces

modularization

**Internet Protocol Stack**

Application: FTP, SMTP, HTTP

Transport: TCP, UDP

Network: routing of datagrams from source to dest. IP, routing protocols

Link: data transfer between neighboring network elements. Ethernet, 802.11(WiFi), PPP

Physical: bits “on the wire”

#### Security

Internet originally not designed with security in mind

Types of malware

virus: self-replicating by receiving/executing object (email attachment)

worm: self-replicating by passively receiving object that gets itself executed

spyware: records keystrokes, other info and uploads

botnet: used for spam and/or DDoS attacks

DDoS: Dedicated Denial of Service

# Chapter 2 – Application Layer

#### Principles of Network Applications

**Application** structures

Client-server

Peer-to-peer

**Sockets**

Processes send/receive messages to/from its socket

Like a door

**Process identifier/address**

Machines 32-bit IP address and port #

**Application Layer protocol defines**

the type of message exchanged: request/response

message syntax

message semantics

rules for when and how processes send and respond to messages

App-layer can be used to secure Transport Layer connection

SSL -> encrypted data over TCP

#### Web and HTTP

**HTTP Overview (stuff to keep in mind)**

HTTP uses TCP

TCP connection (socket) to port 80

HTTP is stateless: server maintains no info about past client requests

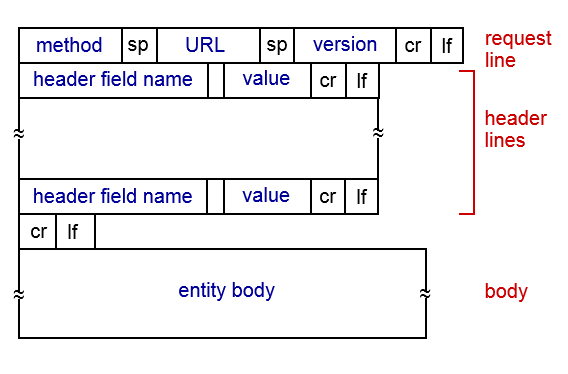
**HTTP Connections**

Non-persistent HTTP (one object at a time) vs persistent HTTP (multiple objects over single TCP connection)

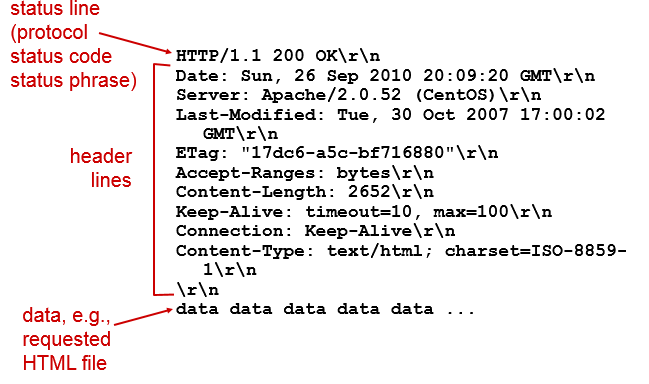
Non-persistent HTTP response time: 2RTT + file transmission time/object

Persistent HTTP response time: >= RTT

**HTTP Request message**



**HTTP Response message**



#### FTP

Connection over TCP

Client browses remote directory

Port 21: control connection

Port 20: data connection(non-persistent)

#### E-mail

3 major components: user agents, mail servers, SMTP

User agent: mail reader (Outlook, Thunderbird)

Mail servers

Mailbox: contains incoming and outgoing mail

Message queue: of outgoing messages

SMTP protocol: between mail servers

SMTP

Uses TCP on Port 25

3 phases:

Handshaking

Transfer of message

Closure

Uses persistent connection

Mail access protocols:

POP: post office protocol: authorization download

IMAP: internet mail access protocol

HTTP: gmail, Hotmail, Yahoo, etc…

#### DNS

Domain name system

Core Internet function implemented as application layer protocol

Service, Structure

Hostname to IP address translation

Not centralized: doesn’t scale

DNS records

A

Name: hostname

Value: IP address

NS

Name: domain

Value: hostname of authoritative name server for this domain

CNAME

Name: alias

Value: canonical name

MX

Value: name of mail server associated with name

File distribution: client server vs P2P

Time to distribute F to N clients

DC-S >= max{ NF/uS , F/dmin }

Dp2p >= max{ F/uS , F/dmin, NF/(uS + sum(ui)) } each peer briongs service capacity

#### Socket Programming with UDP and TCP

UDP

No handshaking between client and server

Transmitted data can be lost or out of order

Client socket (python): socket(socket.AF\_INET, socket.SOCK\_DGRAM)

TCP

Client must contact server by establishing TCP connection with IP and port

Provides reliable, in-order byte-stream transfer (“pipe”) between client and server

Client socket (python): socket(AR\_INET, SOCK\_STREAM)

# Chapter 3 – Transport Layer

#### Transport layer services

Logical communication between app processes

Run in end systems

TCP: reliable, in-order

UDP: unreliable, unordered, best effort

#### Multiplexing/Demultiplexing

Multiplexing: add a transport header

Demultiplexing: use header to deliver received segments to correct socket

UDP segments have dest. IP and port

TCP segments have 4 tuple identifier: source IP port, dest. IP port

Demux uses the four to direct segment to appropriate socket

Allows multiple sources to send to same dest/port but different sockets

#### Connectionless Transport: UDP

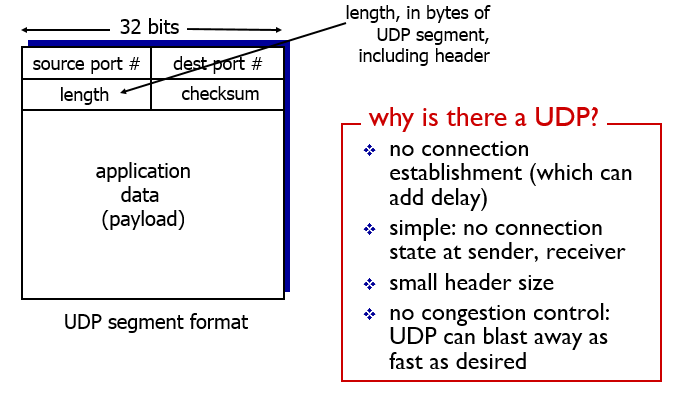
No handshaking

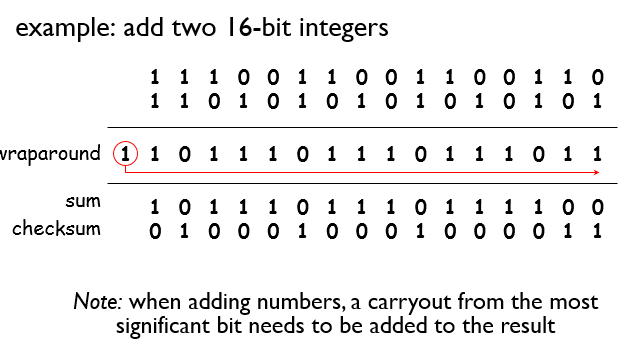
Use: streaming multimedia (loss tolerant, rate sensitive), DNS, SNMP

Reliability is added in application layer

Sacrificing reliability for speed

Checksum: detect errors (flipped bits) in transmitted segment



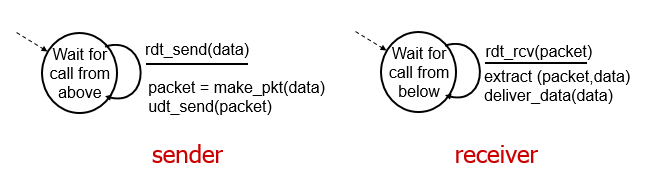


#### Principles of reliable data transfer

**Reliable data transfer RDT**

Rdt 1.0: reliable transfer over reliable channel

Underlying channel perfectly reliable



RDT 2.0: channel with bit errors

Checksum to detect bit errors

ACKS: receiver explicitly tells sender that packet received OK

NAKs: receiver explicitly tells sender that packet has errors

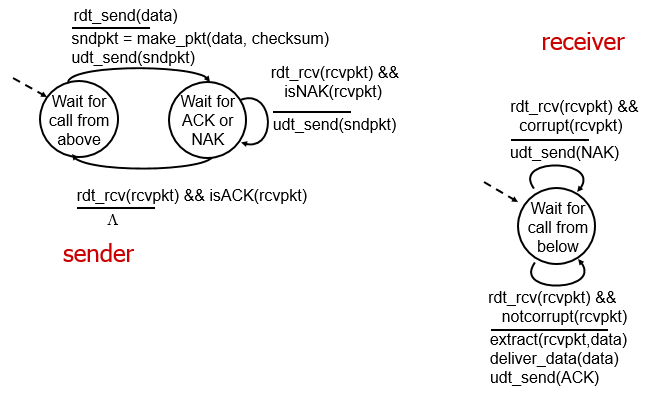
Sender re-transmits

New mechanisms:

Error detection

Feedback: control msgs(ACK,NACK) from rcvr top sndr

Flaw: ACK/NAK and be corrupted so 2.1

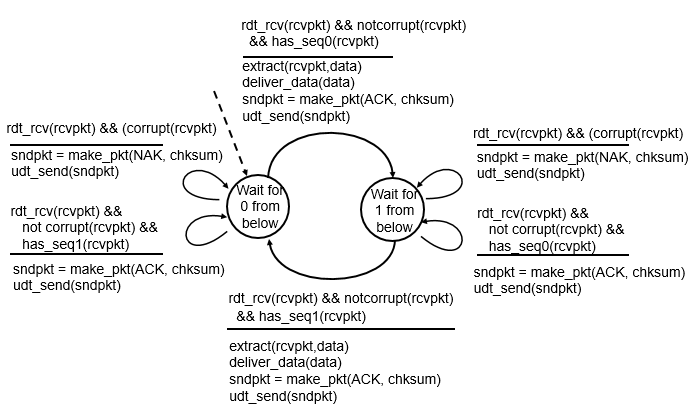
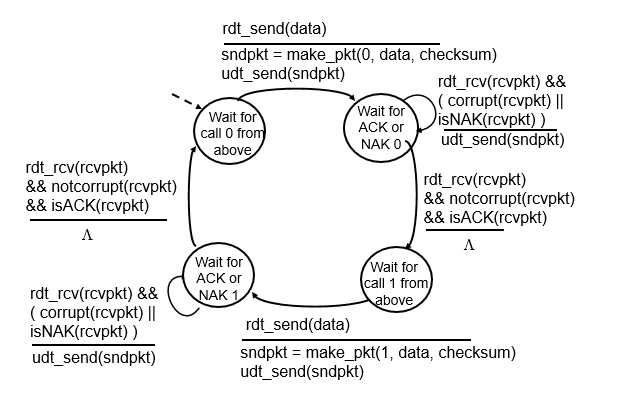


RDT 2.1: handling garbled ACK/NACK

Sender adds sequence number to packet, only 2 needed

State must remember whether expected packet has seq 1 or 0

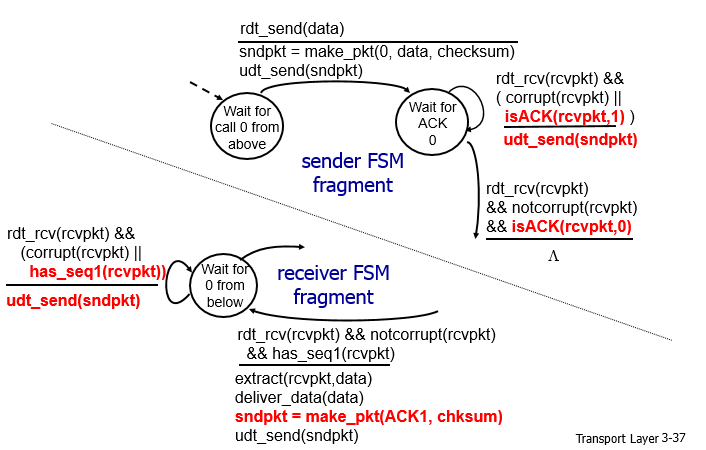
Receiver checks for duplicate



RDT 2.2: NAK free protocol

Same functionality as 2.1 accept using ACKs only

Duplicate ACK at sender results in same action as NAK: retransmit current packet

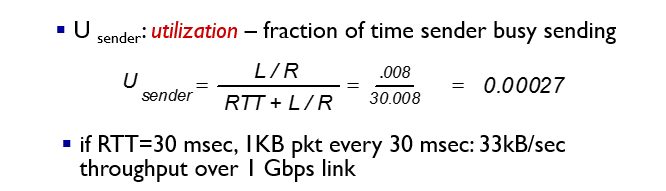


RDT 3.0: Channels with error and loss

Sender waits reasonable amount of time for ACK, if too long retransmit

Sequence number takes care of duplicates

Good protocol but performance is BAD



Pipelining: multiple in-flight yet to be acknowledged packets

2 types:

Go-back-N

Sender can have up to N unacked packets in pipeline

Receiver only sends cumulative ack

Sender has timer for oldest unacked packet

When timer expires, resend all unacked packets

Selective repeat

Sender can have up to N unacked packets in pipeline

Receiver sends ack for each packet

Timer for each individual packet

Receiver can’t see sender side, duplicate data can be received as new data if acks are not received

#### TCP

Point to point

Reliable, in-order

Pipelined: congestion and flow control set window size

Handshaking to initialize data exchange

Flow controlled: sender will not overwhelm receiver

Getting a proper timeout interval

EstmatedRTT = (1 – a) \* EstimaedRTT + a\*SampleRTT

Typically a = 0.125

DevRTT = (1 – b)\*DevRTT + b\* |SampleRTT – EstimatedRTT| (safety margin)

Typically b = 0.25

Timeout interval = EstimatedRTT + 4\* DevRTT

**Flow control:** receiver controls sender so sender won’t overflow receiver’s buffer by transmitting too much too fast

Receiver advertises free buffer space by including rwnd value in TCP header

#### Principles of congestion control

Congestion: too many sources sending too much data

Manifests:

Lost packets (buffer overflow at routers)

Long delays (queueing in router buffers)

2 approaches:

End-end congestion control

No explicit feedback from network

Congestion inferred from end system observed loss, delay

Approach taken by TCP

Network assisted congestion control

Routers provide feedback to end systems

#### TCP Congestion Control

Increase transmission rate until loss occurs

Additive increase: increase cwnd by 1 MS every RTT until loss detected

Multiplicative decrease: cut cwnd in half after loss

Slow Start: increase rate exponentially until loss event

Initially cwnd = 1 MSS

Double cwnd every RTT (every ACK received)

Detecting/Reacting to loss

Loss indicated by timeout

Cwnd set to 1 MSS

Rate grows like slow start until threshold, then linearly

Loss indicated by 3 duplicate ACKs: **TCP RENO**

Cwnd is cut in half, window then grows linearly

**TCP Tahoe**: always set cwnd to 1 (timeout or 3 dup ACKs)

