6.02 Recitation

4/2/06

(10 min late)

Link-State vs Distance Vector

Link-state - each node learns entire topology

distance-vector - each node maintained little state
- next hop
- cost

) for any destination node

~~Have~~ Want a RAM-style compressor/decompressor

typical qv (or something like it)

~~Then~~

After each node has learned network topology - what's next?
- each node runs Dijkstara's algorithm
- then each node makes a routing profile
- which is same in both ways in static case

~~Dija~~ Dijkstara's Algorthm

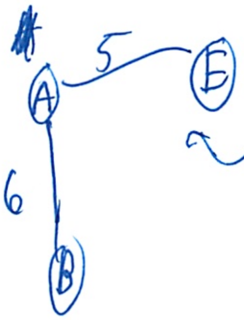
~~Say you are at A, want to go to A~~

1. You are given entire network w/ link cost
2. Shortest path tree given (say) dest (A)

② Shortest path is single path

Superimpose the shortest paths - become a tree

Edge w/ shortest path in must be used



A knows that
E must use link w/ cost 5 to talk to A

Repeat

This is say you were D wanted to go to A
But all nodes are running this the same
(I don't really get why we are doing this)

Code

initalize

V = set of all nodes

Spcost = lowest possible costs to reach A
= {(A,0), (*, ∞)}

routes = {(A,A), *}

3

Find $v \in V$ such that sp cost is smallest

- initally $v = A$

Once you find it, remove it

- remove A from V

Take every neighbor of ~~u~~ v

~~for~~ - B, E in our example

For every neighbor compute the cost to the destination

$d(n) = \text{sp cost}(v) + \text{link cost}(u, v)$

- $v = B$

$N(B) = 0 + 16 = 16$

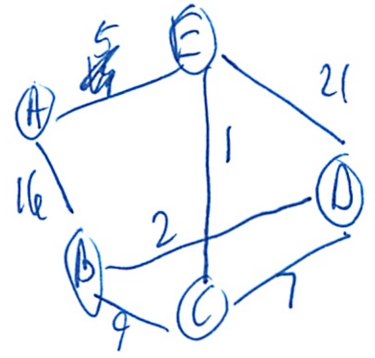
Compare w/ current Sp cost of B

If < sp cost (v), then set sp cost $(v) = d(v)$

So set {B, 16} in sp cost since $16 < \infty$

Set route $(v) \to v$

So set ~~rt~~ (B, A)



(4)

$V = \{A, B, C, D, E\}$

0 ∞ ∞ ∞ ∞

A . . . .

Pop A $V = \{B, C, D, E\}$

16 ∞ ∞ 5

B→A E→A

Then smallest pop E

$V = \{B, C, D\}$

16 6 26

B→A C→E D→E — yeah new link or are we supposed to track initial link (next step)

↑ oh from A ~~not D from B~~ since not replaced

Pop C

$V = \{B, D\}$

15 13

B→C D→C

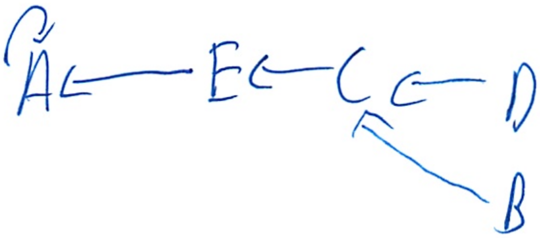
Pop D

$V = \{B\}$

. . .

(5)

When you pop something out its connected forever/frozen



Ignores unused connections

This is done identically by every node in the network for getting to A

Then ~~do~~ each ~~network~~ node does seperatly ~~the destination for~~ something similar for every other ~~node~~ destination node in the network

Optimization

Lagrange parameters?

Like max $x_1^2 + x_2^2$

Such that $3x_1 + x_2 = 4$

Could sub in, deriv, set $= 0$, solve

or

$$F(x_1, x_2, \lambda) = x_1^2 + x_2^2 + \lambda(4 - 3x_1 - x_2)$$

take deriv w/ respect to each

set each $= 0$

$$\frac{dF}{dx_1} = 0 \qquad \frac{dF}{dx_2} = 0 \qquad \frac{dF}{d\lambda} = 0$$

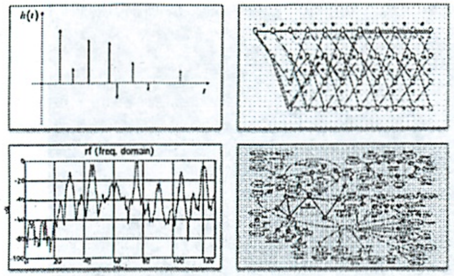
6

Get 3 eq w/ 3 unknown

Solve

The λ is like telling you what to prioritize w/ cost / constraint

4/27



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

HW Eval online now

6.02 Spring 2011
Lecture #21

- Redundancy via careful retransmission
- Sequence numbers & acks
- RTT estimation and timeouts
- Stop-and-wait protocol

The Problem

rough world out there

- Given: Best-effort network in which
  - Packets may be lost arbitrarily
  - Packets may be reordered arbitrarily
  - Packet delays are variable (queueing)
  - Packets may even be duplicated

Noise
queues full

- Sender S and receiver R want to communicate reliably
  - Application at R wants *all* data bytes in exactly the same order that S sent them
  - Each byte must be delivered exactly once
- These functions are provided by a *reliable transport protocol*
  - Application "layered above" transport protocol

for the app —

transport protocal cares about this so not written in each app

Proposed Plan

1960-s like card punch cards

- Transmitter
  - Each packet includes a sequentially increasing sequence number
  - When transmitting, save (xmit time,packet) on un-ACKed list
  - When acknowledgement (ACK) is received from the destination for a particular sequence number, remove the corresponding entry from un-ACKed list w/ seq #
  - Periodically check un-ACKed list for packets sent awhile ago
    - Retransmit, update xmit time in case we have to do it again!
    - "awhile ago": xmit time < now – timeout

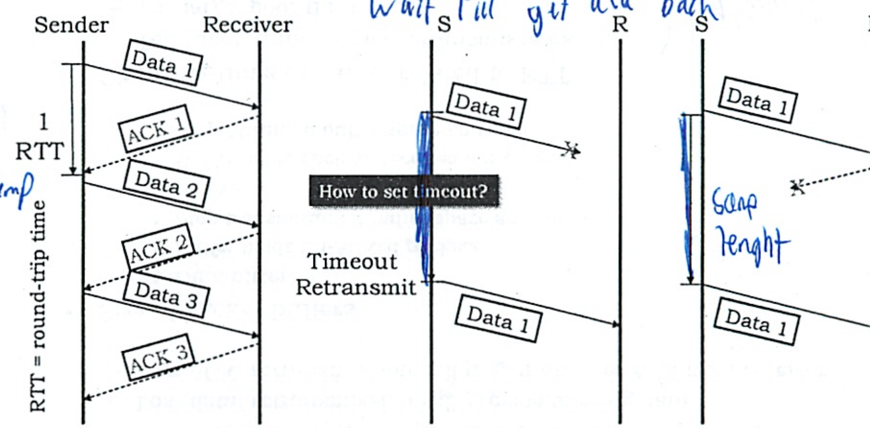
packets sent, but not confirmed rec.
- seq #
- timestamp
- data

how long: round trip time, etc — will talk about

- Receiver
  - Send ACK for each received packet, reference sequence number
  - Deliver packet payload to application

Stop and Wait Protocol

wait till get ack back



Normal behavior (no losses)

Data loss + retransmission

Duplicate packet reception

Can't deliver to app 2x

Wanted "exactly once", got "at least once"

4/27

Revised Plan

- Transmitter
  - Each packet includes a sequentially increasing sequence number
  - When transmitting, save (xmit time,packet) on un-ACKed list
  - When acknowledgement (ACK) is received from the destination for a particular sequence number, remove the corresponding entry from un-ACKed list
  - Periodically check un-ACKed list for packets sent awhile ago
    - Retransmit, update xmit time in case we have to do it again!
    - "awhile ago": xmit time < now − timeout
- Receiver
  - Send ACK for each received packet, reference sequence number
  - Deliver packet payload to application in sequence number order
    - By keeping track of next sequence number to be delivered to app, it's easy to recognize duplicate packets and not deliver them a second time.
    - Remember packets received out-of-order so we can deliver them once missing packet is finally received

make more sophisticated

but this is stop + wait

How long to wait for timeout?
- if network slow (congestion) don't dump more duplicate packets on

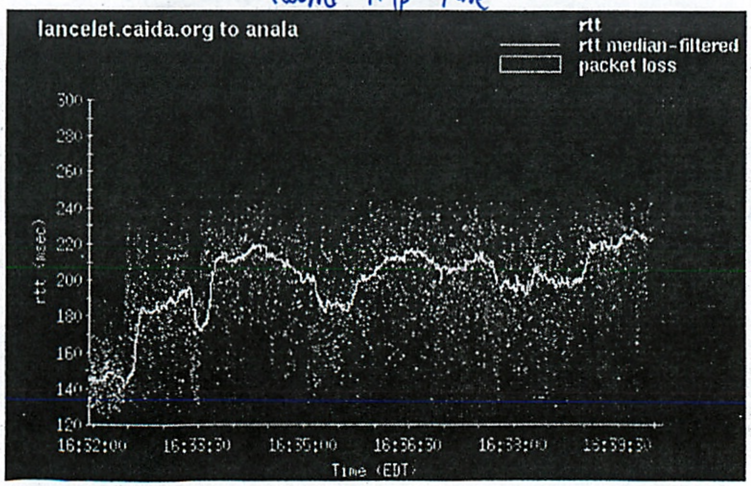
Issues

- Protocol must handle lost packets correctly
  - Lost data: retransmission will provide missing data
  - Lost ACK: retransmission will trigger another ACK from receiver
- Size of packet buffers
  - At transmitter
    - Buffer holds un-ACKed packets
    - Stop transmitting if buffer space an issue
  - At receiver
    - Buffer holds packets received out-of-order
    - Stop ACKing if buffer space an issue
- Choosing timeout value: related to RTT
  - Too small: unnecessary retransmissions
  - Too large: poor throughput
    - Delivery stalled while waiting for missing packets

balance!

RTT Measurements

Round Trip time

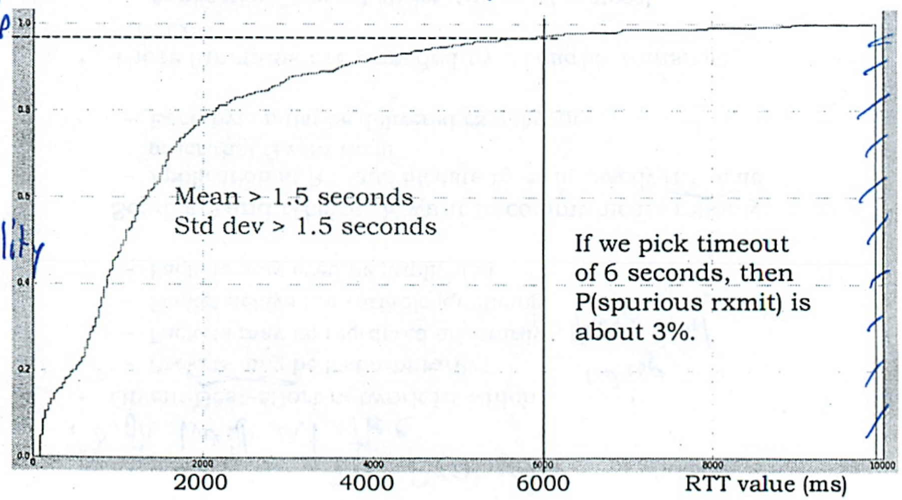


large variability

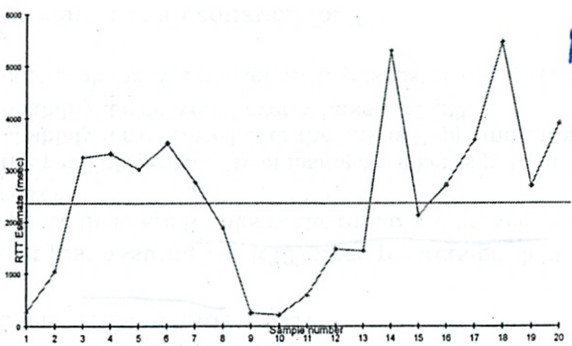
CDF of RTT over Verizon Wireless 3G Network

Slow!

97%



RTT Can Be Highly Variable



Pinged yahoo in class

know RTT from when ack comes back

Example from a TCP connection over a wide-area wireless link
Mean RTT = 2.4 seconds; Std deviation = 1.5 seconds!

Estimating RTT from Data Distribution

- Gather samples of RTT by comparing time when ACK arrives with time corresponding packet was transmitted
  - Sample of random variable with some unknown distribution (not Gaussian!)
- Chebyshev's Inequatility tells us that for a random variable X with mean μ and finite variance σ^2:

$$prob(|X-\mu| \ge k\sigma) \le \frac{1}{k^2}$$

estimation well bound (upper)

  - To minimize the chance of unnecessary retransmissions – packet wasn't lost, just the round trip time for packet/ACK was long – we want our timeout to be greater than most observed RTTs.
  - So choose a k that makes the chances small...
  - We need an estimate for μ and σ

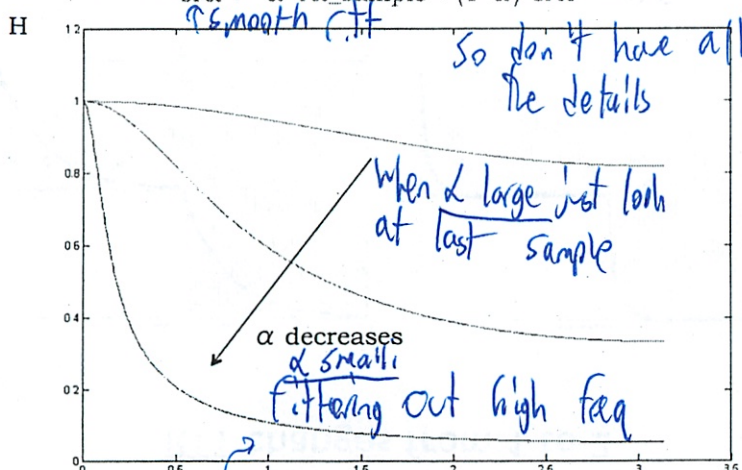
expand by some # of σ = k (just like 15.761)

Exponential Weighted Moving Average (EWMA) LPF Frequency Response

srtt ← α*rtt_sample + (1-α)*srtt

↑ smooth rtt

So don't have all the details



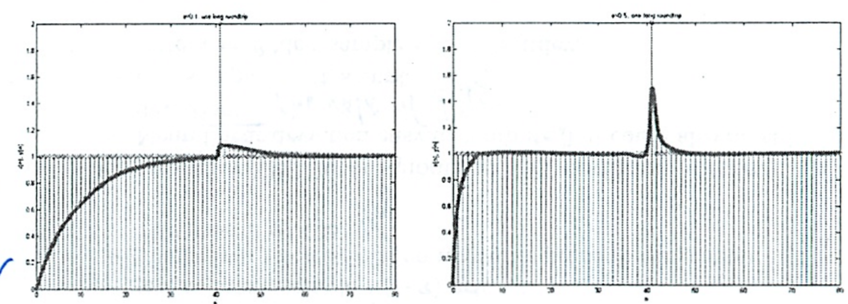
When α large just look at last sample

α small: filtering out high freq

exponential weighted moving avg – like a filter

if see big change – less likly to react

Response to One Long RTT Sample



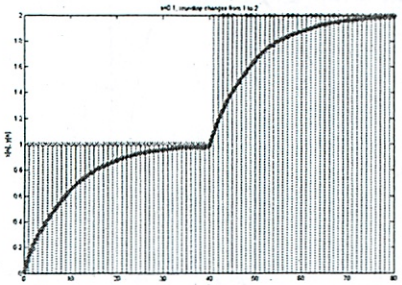
$\alpha = 0.1$

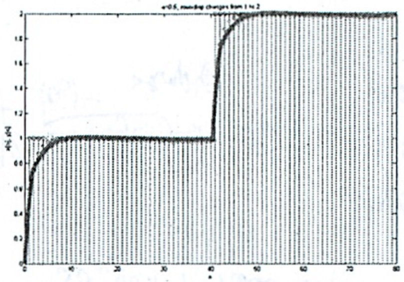
$\alpha = 0.5$

Responds too quickly?

Very sensitive

RTT changes from 1 to 2





$\alpha = 0.1$ $\alpha = 0.5$

What is the best? b/w rock + hard place

Doesn't respond quickly enough?

Timeout Algorithm

- EWMA for smoothed RTT (srtt)
  - srtt ← α*rtt_sample + (1-α)*srtt
  - Typically 0.1 ≤ α ≤ 0.25 on networks prone to congestion. TCP uses α=0.125. 1/8
- Use another EWMA for smoothed RTT deviation (srttdev)
  - Mean linear deviation easy to compute (but could also do std deviation) abs value of difference
  - dev_sample = |rtt_sample – srtt|
  - srttdev ← β*dev_sample + (1-β)*srttdev,
- Retransmit Timeout
  - timeout = srtt + k·srttdev Straight out of [5.26]
  - k = 4 for TCP
  - Makes the "tail probability" of a spurious retransmission low

conservative - not retransmit before needs to

So P(spurious retransmit) = about $\frac{1}{16}$

Throughput of Stop-and-Wait

- We want to calculate the time T between successful deliveries of packets. Throughput = 1/T.
- We can't just assume T = RTT since packets get lost
  - Suppose there are N links in the round trip between sender and receiver
  - If the per-link probability of losing a packet is p, then the probability it's delivered over the link is (1-p), and thus the probability it's delivered over N links is $(1-p)^N$.
  - So the probability a packet/ACK gets lost is $L = 1 - (1-p)^N$.

estimate likely hood packet is lost

Empirical 2%

- Now we can write an equation for T:

$$T = (1-L)\cdot RTT + L\cdot(timeout + T)$$

$$= RTT + \frac{L}{1-L}timeout$$

Some fraction of timeout

Data Ack(N) A (1-p) N=8

S P(loss) = p

$(1-p)^N$ Prob of success for all 8 hops

Bottom Line

- Suppose RTT is the same for every packet, so timeout = RTT

if every packet takes exactly the same time

$$T = RTT + \frac{L}{1-L}RTT = \frac{1}{1-L}RTT$$

$$\text{Throughput} = \frac{1-L}{RTT} = \frac{(1-p)^N}{RTT}$$

- If we can transmit 100 packets/sec and the RTT is 100 ms, then, using stop-and-wait, the maximum throughput is 10 packets/sec.
  - Urk! Only 10% of the capacity of the channel.
  - We need a better reliable transmission protocol... next time: sliding window protocol

6.02 Recitation

4/28

(5 min late)

(substitute instructor)

Reliable Data Transmit

~~what~~ what if packets get lost/damaged?

Need to send an ack

Do Stop+Wait protocol

- waits to get confirm

~~Normally~~ If no ~~errors~~ noise (perfect), round trip time (RTT) will always be the same

Could set retransmission time out (RTO) to = RTT

But RTT changes/varies

Don't want to set RTO too low - will flood network w/ duplicate packets!

(I know all this stuff - remember from lecture!)

Say have n hops

data 5 bits

ack k bits

(2)

oh he is doing

a) What is RTT?

$$RTT = \underbrace{n \frac{S}{R}}_{data} + \underbrace{n \frac{k}{R}}_{ack}$$

$\uparrow$ in real life

$\sum \frac{S}{R}$ since each link different

$+ T_p$ add processing time

$+$ propagation delays

- since queues are full

$+$ speed of light is not ∞

b) When will packets get lost?

- need model for how packets get lost
- say packets get lost on single hop w/ prob $= p$
  - very simple model
- So what is P(loss) of entire ~~the~~ data paths
  - either data not recieved
  - ack lost

or

- P(sucess) = 1 - P(loss)

③

N = total # hops

- ~~start~~ ~~or~~
- data and ack

~~Need to talk about throughput~~

$1 - P(loss) = P(success) = (1-p)^N$

c) Throughput

- Need time T for data to go ~~RTT~~ proper 1 round trip
  - Not exactly RTT
  - Since this must also include needed retransmissions
  - $T \neq RTT$ if losses occur
  - $T = RTT$ if no losses (data or ack)
  - $T = P(success) \cdot RTT + P(loss)(\ldots)$

what goes here?

$(RTO + T)$

another time T

* recursive argument

bring T on both sides + solve

$$T = RTT + \frac{P(loss)}{P(success)} RTO$$

④

Throughput $= \frac{1}{T}$

Util $= \frac{\text{Throughput}}{\text{Max Throughput}}$ = bit rate

↑ debate in last recitation about what last throughput should be

S bits

R bits/sec

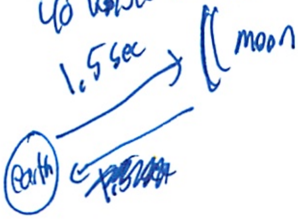
max throughput $= \frac{1}{\text{Time } 1 \text{ packet}}$

$= \frac{1}{S/R} = R/S$

4)

1 kb packet ← byte

40 kb/sec ← bit



a) What is data transfer rate?

Note: diff link speed and time taken by 1 packet

(5)

Time taken by 1 packet

Rate $= \frac{1\text{kb}}{1.5 \cdot 2} = 333$ bits/sec = ~~2.6~~ kb/sec

$\boxed{2.6}$

b) Utilization

$\frac{\text{data rate}}{\text{link rate}} = \frac{2.6 \text{ kb/sec}}{48 \text{ kbits/sec}} = 6.5\%$

Ack problems

5. Reciever

Packets have a seq #

Acks use that #

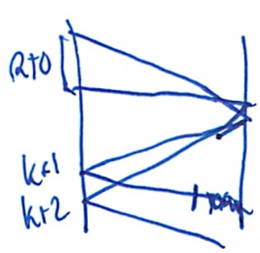
But what if reciever does not send packet #?

Even in stop + wait packets can be reordered

- if ~~pa~~ ack gets delayed, a new one is sent, and recieved, then orignal ack gets delivered, then ... do more

- or

only broken if no reorders



this will have k+1 missing

Checkoff 8

4/29

Don't have to do avg

- Could do $\frac{max - min}{2}$

↑ minimize the max error

Could min error^2

Etc

Quiz is during final - offical time

Only last third

2hrs

Checkoff 4

4/29

Small error in 2a

So 4x not 2x

Chain net made error w/ later nodes powering on not sending everywhere

Then the bounce back one

A, B do offer route to D

Almost Always have count to ∞ problem

Don't consider packet, consider table

B still has route!

---

t_1 C-D breaks

B ad

C ad

Bounce

B↔C

OR

t_1 CD breaks

C ad

A ad

B ad

Bounce A↔B

2

Much of the P-set is wrong

Look at solutions!

Last one

- diameter of network • adv invert

Made too complex

~~[illegible]~~ takes 2 Hellos to realize link broken

~~[illegible]~~ add advert time

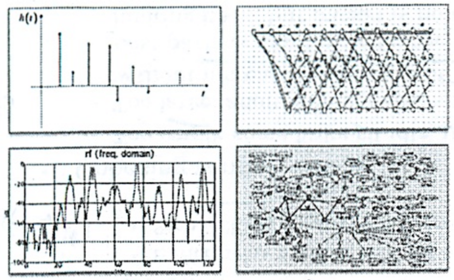
LS about same - do another flood

LS - each node just sends very local info - link info

each node then reconstructs entire network

DV - ~~[illegible]~~ sends its routing table

"Best-effort" - can drop packets



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

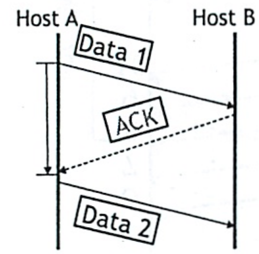
6.02 Spring 2011 Lecture #22

- Sliding-window protocol
- Sizing the window

Quiz 3 5/17 1:30-3:30 Johnson
- not cumulative

5/12 is pset deadline
no class next wed

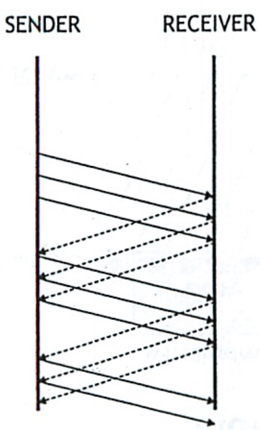
Improving Performance



- Stop-and-wait protocol too slow
- Throughput = 1 packet per RTT
- 1500 byte pkt, 100 ms RTT throughput pegged at 15 KBytes/s
- With packet loss & timeout, throughput even lower
- Solution: Use a *window*
  - Allow W packets outstanding in the network at once (W is called the window size).
  - Overlap transmissions with ACKs

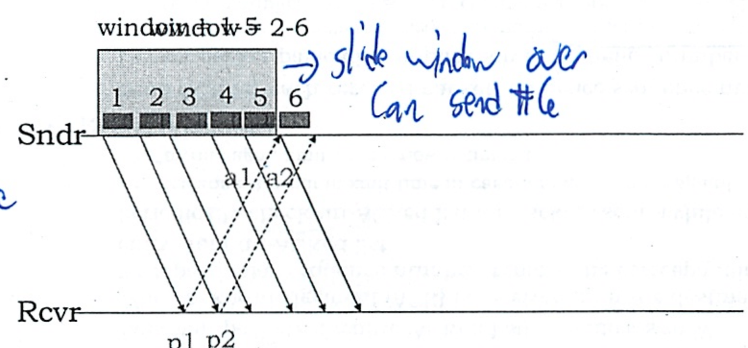
what should W be? for max throughput

Solution: Use a Sliding Window



- Senders advances the window by 1 for each in-sequence ack it receives
  - I.e., window *slides*
  - So, idle period reduces
  - **Pipelining** doing more than 1 thing at once
- Assume that the window size, W, is fixed and known
  - Later, we will discuss how one might set it
  - W = 3 in the example on the left

Sliding Window in Action



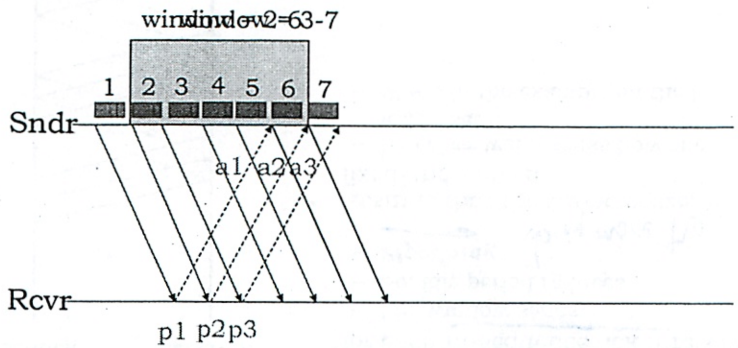
→ slide window over
Can send #6

W = 5 in this example

Have W packets
Outstanding

5/12

Sliding Window in Action



Window definition: If window is W, then max number of ~~*unacknowledged packets*~~ is W → packet is outstanding

This is a fixed-size sliding window

Sliding Window Implementation

- Transmitter
  - Each packet includes a sequentially increasing sequence number
  - When transmitting, save (xmit time,packet) on un-ACKed list
  - Transmit packets if len(un-ACKed list) ≤ window size W
  - When acknowledgement (ACK) is received from the destination for a particular sequence number, remove the corresponding entry from un-ACKed list
  - Periodically check un-ACKed list for packets sent awhile ago
    - Retransmit, update xmit time in case we have to do it again!
    - "awhile ago": xmit time < now − timeout
- Receiver
  - Send ACK for each received packet, reference sequence number
  - Deliver packet payload to application in sequence number order
    - Save delivered packets in sequence number order in local buffer (remove duplicates). Discard incoming packets which have already been delivered (caused by retransmission due to lost ACK).
    - Keep track of next packet application expects. After each reception, deliver as many in-order packets as possible.

It's not really a window - if one packet is stuck - it will send future rest - simplification for our purposes

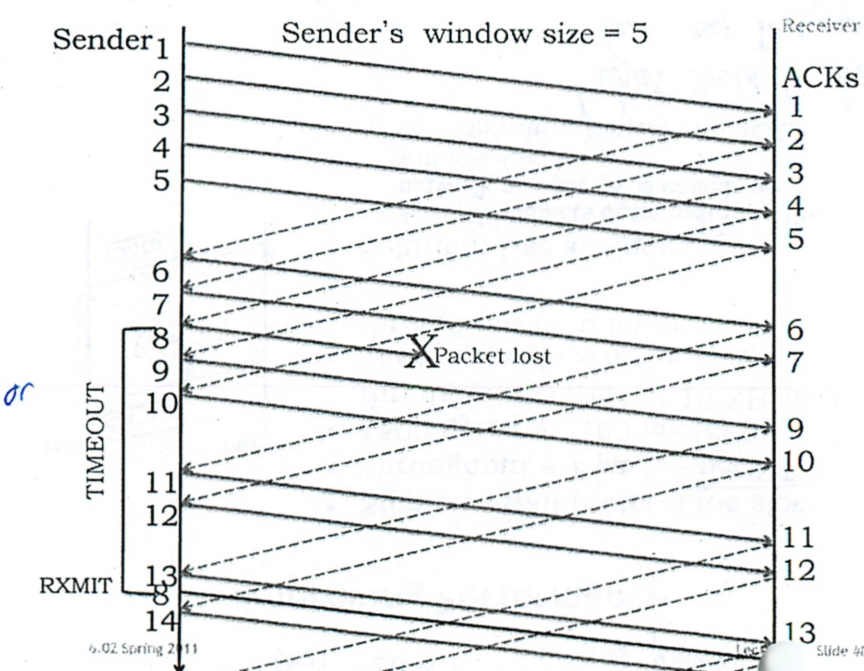
Issues

TCP = 4

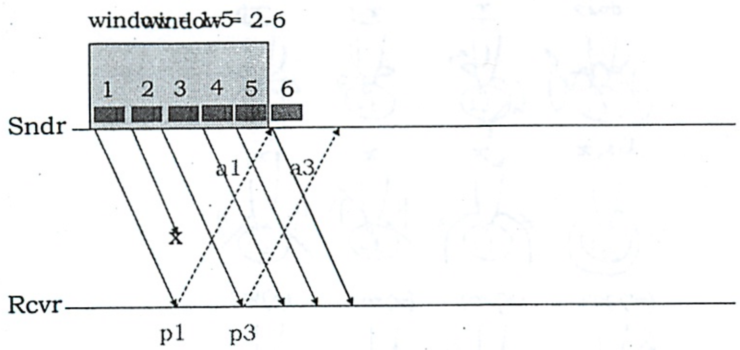
- Timeout chosen as before. └ Smooth round trip time + k * std dev
  avg case
  no spurious retransmit
- Size of packet buffers
  - At transmitter
    - Buffer holds un-ACKed packets
    - Stop ~~transmitting~~ if buffer space an issue
  - At receiver
    - Buffer holds packets received out-of-order
    - Stop ACKing if buffer space an issue

receiver can only send ACKs or not - not any special commands

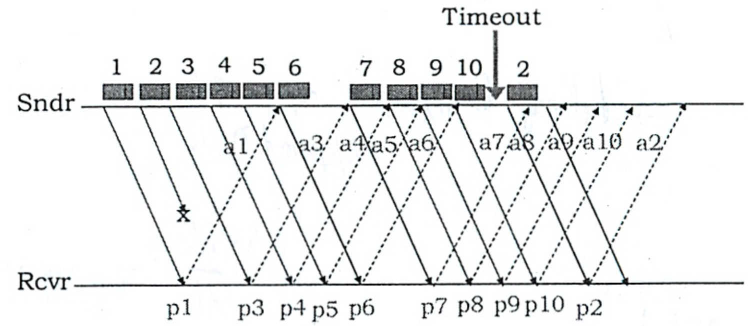
- Choosing window size W
  - Too small: some links ~~sit idle~~, throughput less than maximum
  - Too large: additional packets sit in queues, increasing latency without increasing throughput. ← Little's law
  - Lost packets diminish effective window size (transmitter is limiting itself, but packet is no longer in the network).



Sliding Window: Handling Packet Loss



Sliding Window: Handling Packet Loss

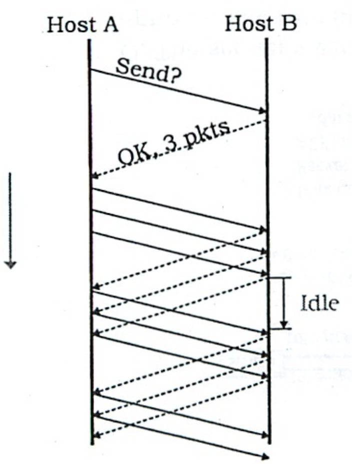


The receiver has to save packets 3 through 10 until packet 2 arrives, which will allow it to deliver packets 2 through 10 to the application. Note that with this definition of the window protocol, there's no limit to the number of packets that might arrive out of order.

Reciever - semi ∞ buffer - as long as it keeps ACking

How to set W?

Setting the Window Size: Apply Little's Law



- If we can get "Idle" to 0, will achieve goal

network should not sit idle!

- "N" = #packets in window
- B = rate of slowest (bottleneck) link
- RTT = avg delay
- If $W = B{\cdot}RTT$, path will be fully utilized
  - The **"bandwidth-delay product"**
  - Key concept in transport protocols

Throughput of Sliding Window Protocol

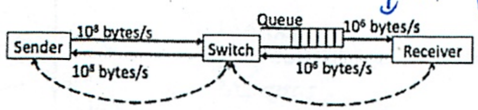
- Goal: select W so that (slowest) links are never idle due to lack of packets
  - Avoid overfilling queues since that increases packet latency and, if timeouts are triggered, possibility of spurious retransmissions.
    - Measured RTT includes queuing delay = $RTT_{min} + Q_{delay}$
    - As Q_{delay} increases, so does W, which increases Q_{delay}, …
    - Use B·RTTmin when calculating W
  - Slightly larger than $B{\cdot}RTT_{min}$ to ensure bottleneck link is busy even if there are packet losses
    - total # of transmissions for successful delivery $= 1 + L{\cdot}(1 + L{\cdot}(1+\ldots)) = 1 + L + L^2 + \ldots = 1/(1-L)$ where $L < 1$ is the round-trip loss rate.
    - Max utilization is 1/throughput = 1−L
- If there are no lost packets, protocol delivers W packets every RTT seconds, so throughput = W/RTT.
  - Maximum throughput is limited by rate of bottleneck link (B)

Set window to real RTT don't include queues!

actual throughput = $\min(B, \frac{W}{RTT})$ | will queue up in front of slow link but Queue length ↑, Queue time ↑ so ↑ RTT, ↑ w, etc

Example

slow link bottleneck



Propagation delay = 0 milliseconds

One-way propagation delay = 10 milliseconds

Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Initial sender window size = 10 packets

Q: The sender's window size is 10 packets. At what approximate rate (in packets per second) will the protocol deliver a multi-gigabyte file from the sender to the receiver? Assume that there is no other traffic in the network and packets can only be lost because the queues overflow.

B = 1e6 bytes/sec = 1000 packets/sec

W = 10

RTT = prop delay + xmit time

= 20 ms + 1 ms + ε ≈ 21 ms

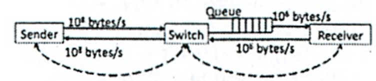
min(1000 packets/sec, $\frac{10}{21ms}$)

↓

476 packets/sec

↑ so window is controlling throughput

Example



Propagation delay = 0 milliseconds

One-way propagation delay = 10 milliseconds

Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Initial sender window size = 10 packets

Q: You would like to double the throughput of this sliding window transport protocol. To do so, you can apply one of the following techniques:

a. Double window size W 2W
b. Halve the propagation delay of the links RTT/2 21 ms → 11 ms
c. Double the speed of the link between the Switch and Receiver. 2·B

For each of the following sender window sizes, list which of the above technique(s), if any, can approximately double the throughput: W=10, W=50, W=30.

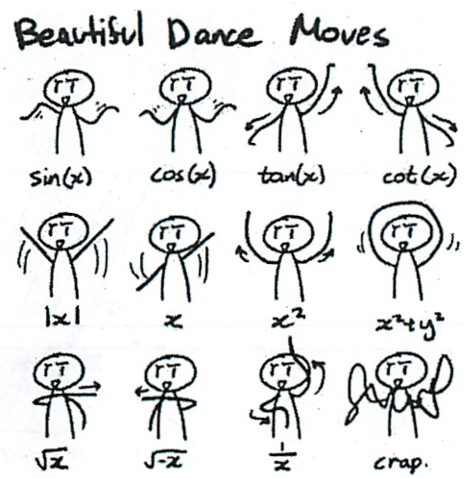
What happens?

	W=10	W=50	W=30
2W	952 ✓	X	X
RTT/2	~952 ✓	X	X
2B	476 ✗	✓	$\frac{30}{.021}$ = 1400

↑ not limiting factor

↑ now limited by B

↑ was limited by B, but when 2×, limited by W so does not ↑

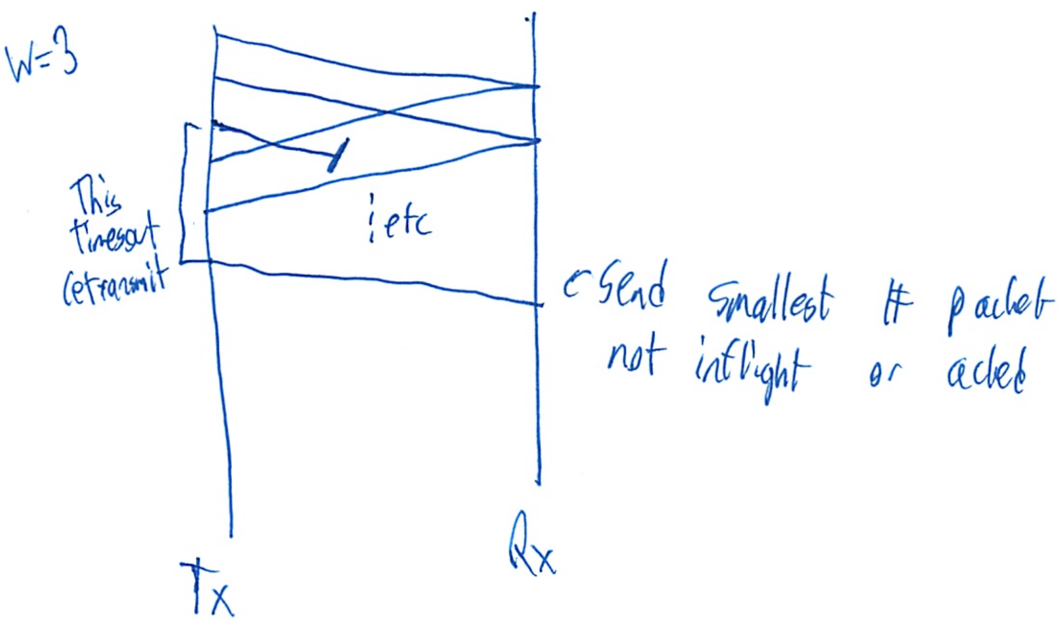


"Shouldn't that be –sin(x)?"

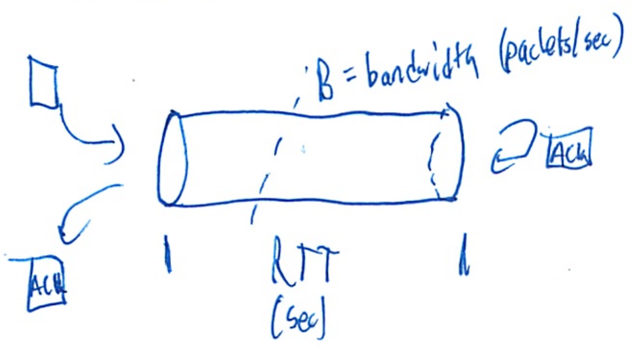
6.02 Recitation 5/3

Sliding Window Protocal

- W is window size
- At any pt in time can have W packets "in flight"
  - not acknogled
- If non ack packets < W, transmit next packet



Setting The Window Size



Look at Little's Law

How long should queue be?

(2)

Queue = rate · delay

$Q = B \cdot RTT$

# of unacked packets

Should $= W$

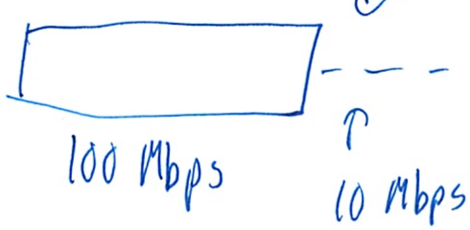
Rate could be $< B = x$

Then

$Q \approx x \cdot RTT$

$x \approx \frac{W}{RTT}$ effective departure rate

Effective Throughput $= \min\left(\frac{W}{RTT}, B\right)$



$U = \frac{10 \text{ Mbps}}{100 \text{ Mbps}} = \frac{1}{10} = .1 = 10\%$

TCP is a sliding window protocol

It keeps changing window size

It needs to sense its effective throughput

Think of as Media Access Problem

- where did not know how many users

③

How to find out your bandwidth?

If keep getting ACks, your good

↑ window size

Till packets start getting dropped

- Have reached the boundry

For network, how to share ~~band~~ bandwidth

- like Cake sharing
- TCP achieves
- Like Boltzman's ideal gas law
  - one parameter to describe state of system
  - model for how gas molecules behave
  - entropy maximization
  - Teleology
    - principles of systen
  - People found that it is fair
  - Solves optimization problem

(9)

Tutorial Problems

~~$\cancel{\#}$~~

a) A — B — C — D — E
$1s \quad 1s \quad 1s \quad 1 sec$

$RTT = 8$ sec

b) Stop + Wait. Throughput?

$$\frac{1}{RTT} = \frac{1}{8} \text{ packets/sec}$$

c) Sliding window. Best w?

Want $RTT \cdot B$

$\uparrow$ what is?
1 packet/sec

$W = RTT \cdot B$
$W = 8 \cdot 1 = 8$

Throughput $= B$

(5)

d) W=4. What is throughput?

$\frac{1}{2}$ packets/sec

Utilization → $\frac{\frac{1}{2}\text{ packet/sec}}{1\text{ packet/sec}} = .5 \sim 50\%$

e) W=8 again. Now every odd packet is dropped

timeout = 40sec unacked

A - - - - E

0 - - - - 35sec

What will it recieve?

Only even things

1
⋮ Rx 8 packet
8
9 Gets acked from 2
11 Tx
13
15
31 last sent

⋮
12 4 packets Rx
35

(getting confused w/ 0,1 indexed)

6 ~~But!~~

~~Window size ↓ !~~

1
2
3
4
5
6
7
8

10

12

14

16

20

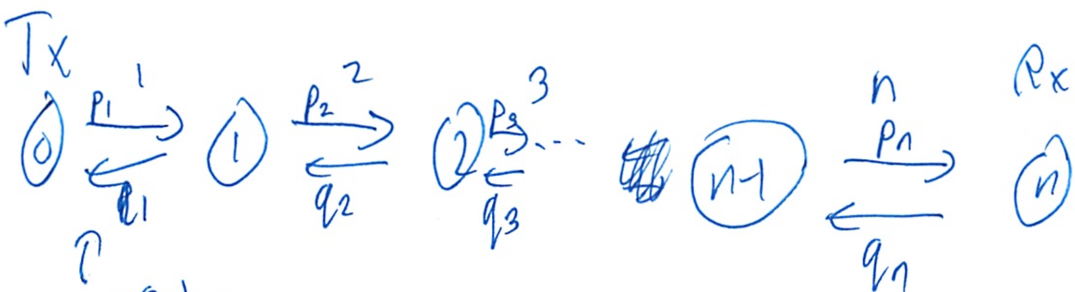
24 ← even packets up to 14 so 7

32

Window size is not changing!
Unacked packets still $\leq$ W
(So the every 2 packet pattern does to repeat!
I think I just did not think it through
(It trys tx 9 - which does not arrive)
(For some odd reason I thought it only Tx even after this)

7

New question



$p(\text{packet}_{tx} \text{ is dropped}) = p_1$

$p(\text{packet}_{ack} \text{ is dropped}) = q_1$

So what is RTT?

Need to find avg # of attempts

What is p(won't fail on first hop)

$$= (1-p_1)(1-q_1)$$

So # of attempts =

$$\frac{1}{P(\text{sucess 1st hop})} = \frac{1}{(1-p_1)(1-q_1)}$$

But what about the other nodes?

8

$P_{success} = (1-p_1) \ldots (1-p_n)(1-q_1) \ldots (1-q_n)$

Suppose $p_i = q_i = \varepsilon$

$P_{success} = (1-\varepsilon)^{2n}$

Suppose ε is really small

Approxi

$= \left((1-\varepsilon)^{1/\varepsilon}\right)^{2\varepsilon n}$

Suppose $\lim \varepsilon \to 0$

$\lim_{x \to 0} (1-x)^{1/x}$

$\lim_{k \to \infty} \left(1-\frac{1}{k}\right)^{k}$

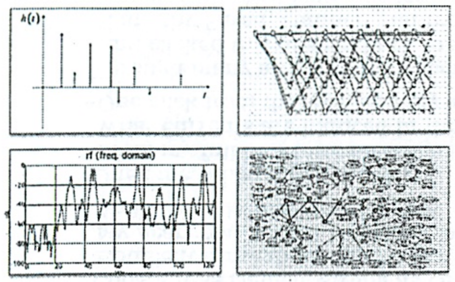
$\approx e^{-2\varepsilon n}$

Taylor expansion (only thing that matters

$\boxed{e^{-x} \approx 1-x} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

9

ㅈ|-26n



INTRODUCTION TO EECS II
DIGITAL COMMUNICATION SYSTEMS

Before

Sneaker net

HKN Eval!

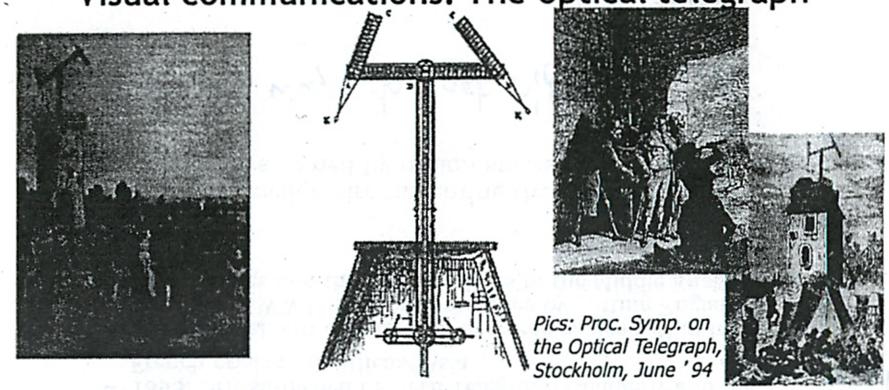
6.02 Spring 2011
Lecture #23

- Evolution of communication networks

P-Sets due 5/12 5PM
HKN

Quiz 3 5/17 (1:30-3:30) at
No lecture ~~[illegible]~~ Wed ~~[illegible]~~

Visual communications: The optical telegraph



Pics: Proc. Symp. on the Optical Telegraph, Stockholm, June '94

- Chappe (1763-1805), a "defense contractor"; 1st message successfully sent in 1794
- 1799: Napoleon seizes power; sends "*Paris is quiet, and the good citizens are content.*"
- 1814: Extends from Paris to Belgium & Italy
- 1840: 4000 miles, 556 stations, 8 main lines, 11 sublines, each hop ~10 km
- Many "advanced" techniques: switching, framing, codes, redundant relays, message acks, priority messages, error notification, primitive encryption!

Same day
advantage of Napolean
Various techniques used

modeled on ships

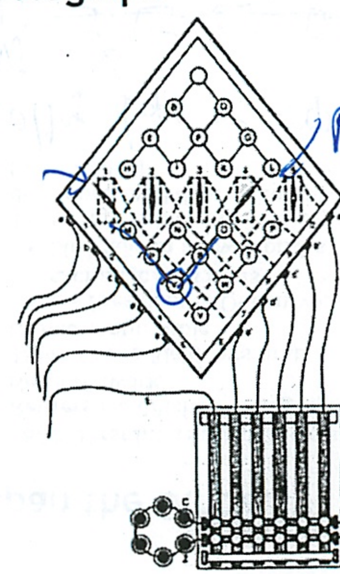
Advances in Electricity and Magnetism (Late 18th and 19th centuries)

- Oersted (Copenhagen): demonstrated electricity's ability to deflect a needle
- Sturgeon (London), 1825: electromagnet demo
- Joseph Henry, 1830: 1-mile demo: current through long wires, causing bell to ring!
- Faraday (London), 1831: EM induction experiments (induction ring), basis for motors

keyi action in a distance

The Electric Telegraph

- Cooke and Wheatstone, Railroad Telegraph, 1837
  - 1
  - 4



points at V

need 5 wires + grand

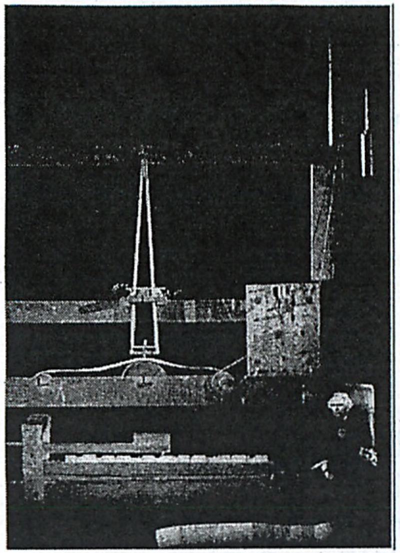
5/4

Are there better ways?

encoding

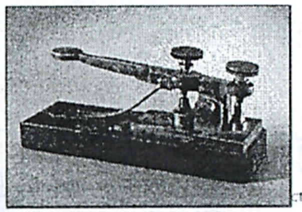
The Electric Telegraph (Samuel Morse)

paper tape

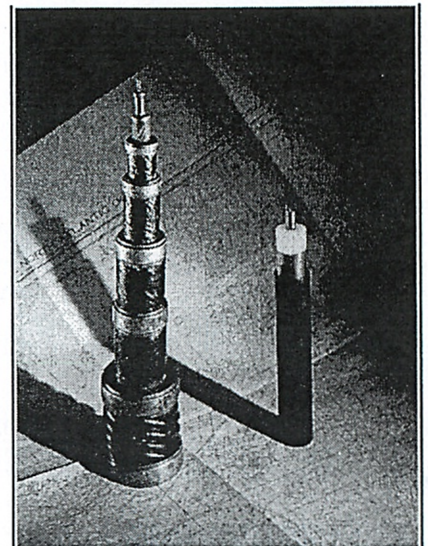


Morse Code (1835-1837)

- 1838: demo'd over 2 miles
- 1844: US- sponsored demonstration between Baltimore and Washington DC



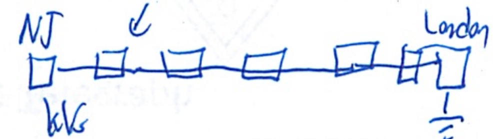
Dots and Dashes Span the Globe



big issues underwater

- 1852: First international telegram
- Reuters establishes "Telegraph News Network"
- 1858: Cyrus Field lays first transatlantic cable
  - US President & Queen Victoria exchange telegrams
  - Line fails in a few months
- 1866: New cable & technology developed by William Thompson (Lord Kelvin)

Voltage drops for repeaters



Early Uses (cf. IM today!)

Valentine by a Telegraph Clerk (male) to a Telegraph Clerk (female):

"The tendrils of my soul are twined
With thine, though many a mile apart,
And thine in close-coiled circuits wind
Around the needle of my heart.

"Constant as Daniell, strong as Grove,
Ebullient through its depths like Smee,
My heart pours forth its tide of love,
And all its circuits close in thee.

"O tell me, when along the line
From my full heart the message flows,
What currents are induced in thine?
One click from thee will end my woes."

Through many an Ohm the Weber flew,
And clicked this answer back to me, --
"I am thy Farad, staunch and true,
Charged to a Volt with love for thee."

Who or what are Daniell, Grove and Smee?! ☺

Dots and Dashes Span The Globe

- Communications arms race in the Imperial Age
  - No nation could trust its messages to a foreign power
  - 1893: British-owned Eastern Telegraph Company and the French crisis in Southeast Asia
  - 1914: British cut the German overseas cables within hours of the start of WW I; Germany retaliates by cutting England's Baltic cables and the overland lines to the Middle East through Turkey

cut everyones cables

- Strategic necessity: circumventing the tyranny of the telegraph lines owned by nation states

Need a way to get rid of wires

Wireless!

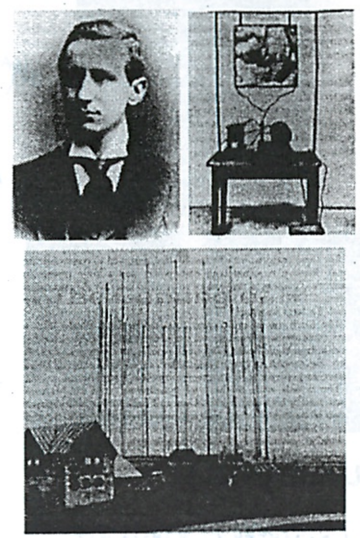
James Clerk Maxwell (1831-1879)

"... we have strong reason to conclude that light itself -- including radiant heat, and other radiations if any -- is an electromagnetic disturbance in the form of waves propagated through the electromagnetic field according to electromagnetic laws." *Dynamical Theory of the Electromagnetic Field*, 1864.

Heinrich Hertz (1857 - 1894)

– Mid-1880s: Demonstrated experimentally the wave character of electrical transmission in space

Wireless Telegraphy

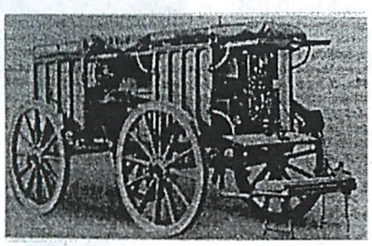


Guglielmo Marconi

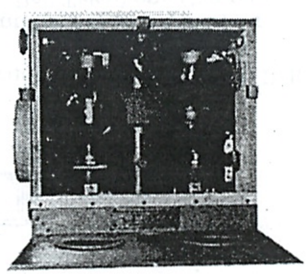
– 1895: 21 year-old demonstrates communication at distances much greater than thought possible
– Offers invention to Italian government, but they refuse
– 1897: Demonstrates system on Salisbury Plain to British Royal Navy, who becomes an early customer
– 1901: First wireless transmission across the Atlantic
– 1907: Regular commercial service commenced

Wireless in Warfare

Mobile



"Portable" radio, circa 1915



Airborne radio telephone, post WW I

In the Meantime, in the Wired World...

- The telegraph learns to talk
- Morse telegraph: no multiplexing
  – Only one message sent/received at a time
- Second half of 19th century: many researchers work on improving capacity
- Idea: send messages at different pitches
  – Graham Bell – harmonic telegraph
  – Develops way to send different source frequencies by adjusting current levels

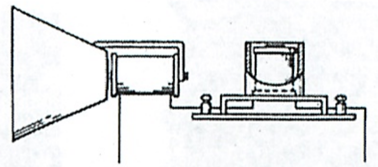
freq division multiplexing

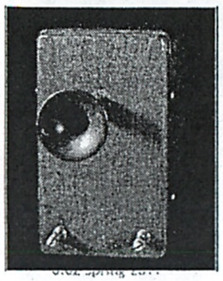
Speed.. was all driven by hand

Want most use of investment

The Telephone

Alexander Graham Bell
- 1876: Demonstrates the telephone at US Centenary Exhibition in Philadelphia





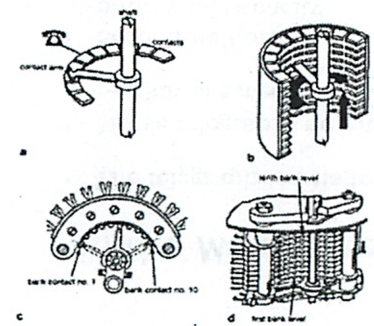
- Bell and Elisha Gray rush patents to USPTO, Bell first by a few hours
- Bell offers to sell patents to Western Union for $100,000, who refuse. Bell Telephone Company founded 9 July 1877.
- 1878: Western Union competes using rival system designed by Thomas Edison and Elisha Gray. Bell sues and wins.

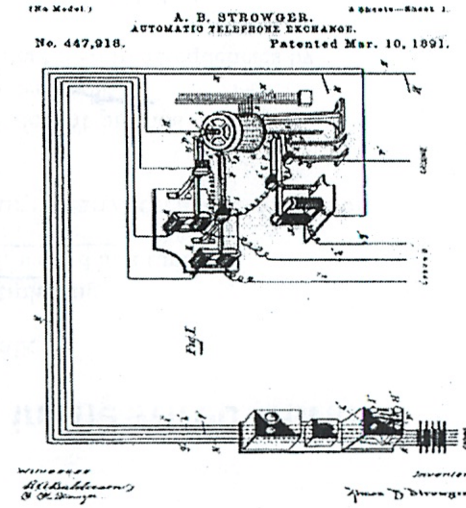
↑ where patent fight was actually worth it

Setting up a circuit

Mechanical Telephone Switch

Almon Brown Strowger (1839 - 1902)
- 1889: Invents the "girl-less, cuss-less" telephone system





↑ actually making a physical network connection

"Ma Bell" and the Telcos

- Bell's patents expire in 1890s; over 6000 independent operators spring up
  - 1910: Bell System controls 50% of local telephone market
  - 1913: AT&T & U. S. government reach Kingsbury Agreement: AT&T becomes regulated monopoly while promising "universal" telephone service
    - Long distance interconnection withheld as a competitive weapon
- 1950: Bell controls 84% of the local telephone access market
- 1984: Divesture of Ma Bell (Judge Greene)
- 1996: Trivestiture of AT&T Bell (AT&T, Lucent, NCR)
- **2000s: The death of the classic wired telephone network**

how to ensure service for everybody

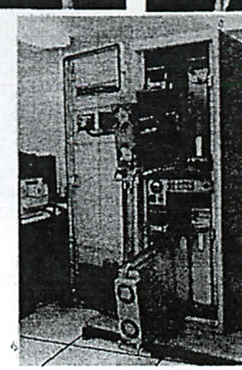
- how to pay for service in middle of nowhere

tech + long distance broke off

hard to sustain Ind.

The Dawn of Packet Switching

data communications



ARPA: 1957, in response to Sputnik

Paul Baran (RAND Corp)
- Early 1960s: New approaches for survivable comms systems; "hot potato routing" and decentralized architecture, paper on packet switching over **digital** comm links

Donald Davies (UK), early 1960s
- Coins the term "packet"

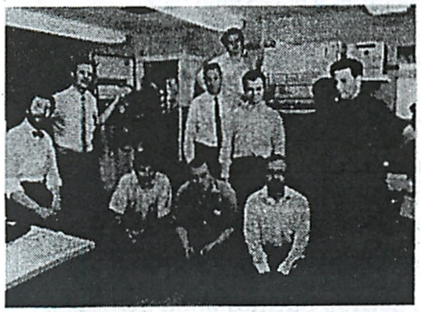
Len Kleinrock (MIT thesis): "Information flow in large communication nets", 1961

J. Licklider & W. Clark (MIT), On-line Man Computer Communication

L. Roberts (MIT then ARPA), first ARPANET plan for time-sharing remote computers

packet switched
- survive attacks
- connect computers

ARPANET



BBN team that implemented the interface message processor

Company that got contract

- 1967: Connect computers at key research sites across the US using telephone lines
- Interface Message Processors (IMP) ARPA contract to BBN
- Ted Kennedy telegram on BBN getting contract
  - Congratulations ... on **interfaith** message processor"

not interface

Initial Baby Steps

1 computer to each node

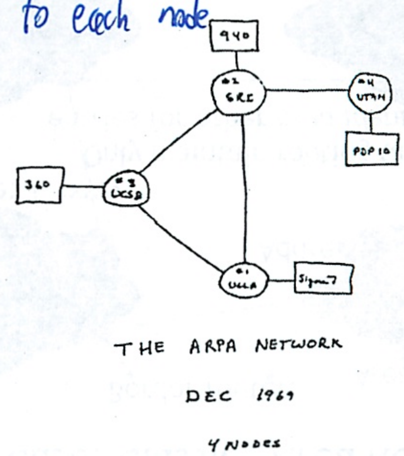


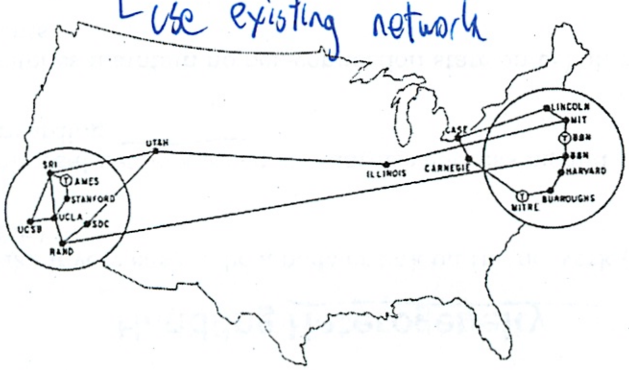
FIGURE 6.2 Drawing of 4 Node Network (Courtesy of Alex McKenzie)

September 1971

1970, ARPANET hosts start using NCP; first two cross-country lines (BBN-UCLA and MIT-Utah)

"Hostile overlay" atop telephone network

Use existing network



MAP 4 September 1971

1970s: Internetworking Develops

Everyone had their own packets

- 1972: modified ARPANET email program
- 1972: French CYCLADES network – developed **sliding window** protocol
- 1973: ARPANET becomes international
- 1973-75: Internetworking effort (Cerf, Kahn, et al.) → Internet
  - Developed TCP and IP (originally intertwined) – TCP uses **sliding window**

need to unify schemes

Competitors collaborate!

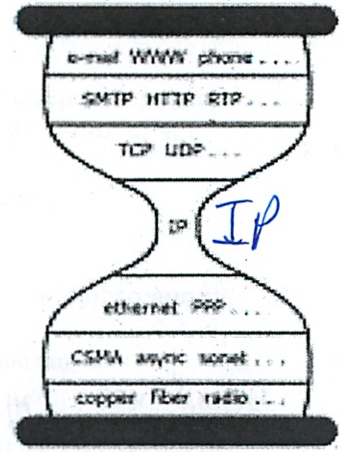
Handling Heterogeneity

- Make it very easy to be a node or link on the network (best-effort)
- Universal *network layer*: standardize addressing and forwarding
- Switches maintain no per-connection state on behalf of end points

robust
diff. equipment
intermediate nodes don't really know anything

1970s: Internetworking

- 1978: **Layering**! TCP and IP split; TCP at end points, IP in the network
- IP network layer: simple best-effort delivery
- In retrospect: Packet switching won because it is good enough for almost every application (though optimal for almost nothing!)



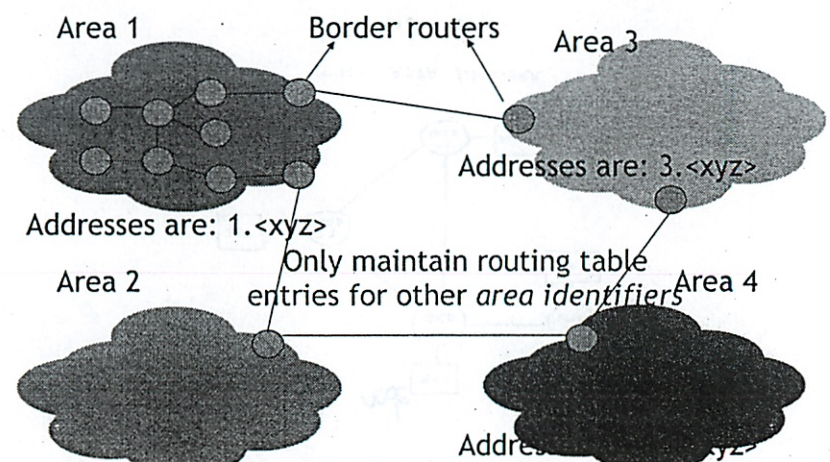
1980s: Handling Growth with Topological Addressing

How to address everyone

- Per-node routing entries don't scale well
- Solution: Organize network hierarchically
  - Into "areas" or "domains"
  - Similar to how the postal system works
  - Hide detailed information about remote areas
- For this approach to work, node addresses must be **topological**
  - Address should tell network *where* in the network the node is
  - I.e., address is a *location* in the network

like area code for phones

Ideal Case: Classic "Area Routing"



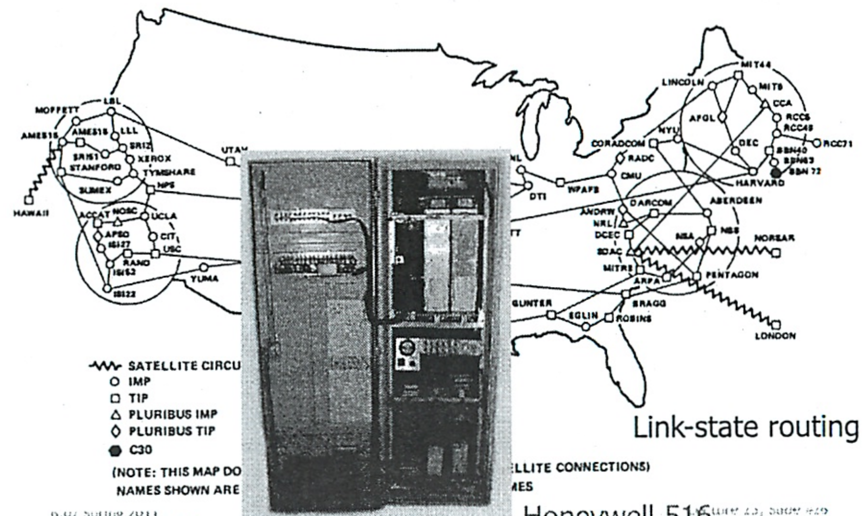
And one could have areas within areas, etc.

IPv4 Example: Addresses & Prefixes

- 18.31.0.82 is actually the 32 bit string 00010010 00111110 00000000 01010010
- Routers have forwarding table entries of the form Address/ Mask, which corresponds to a **prefix**
  - Range of addresses that use the route
- 18.0.0.0/8 stands for all IP addresses in the range 00010010 00…0 to 00010010 11…1
- Hence, "areas" may be of size 1, 2, 4, 8, … (maxing out at 2^{24} usually)
  size of area
- Forwarding uses **longest prefix match**

/8 Class A 18.*.*.*
/16 Class B 18.245.*.*
/24 Class C 18.245.2.*

ARPANET GEOGRAPHIC MAP, OCTOBER 1980



Link-state routing

Honeywell 516

Starting to get commercial interest

1980s: Rapid Growth

- 1982: US DoD standardizes on TCP/IP
- 1984: Domain Name System (DNS) introduced ← names hard to distribute
- 1986: Congestion collapse episodes
  - Problems with bad timeout settings
  - Adaptive timers, TCP congestion control solution
  - Athena network file system congestion problems (bad timeout settings)

  & became very important - network would just melt down
- Solution
  - **RTT estimation using EWMA, timeout method**
  - TCP congestion control

1990s

- 1990: no more ARPANET
- 1991: WWW released (Berners-Lee) LS inside LANs
- Mid-1990s: NSFNet gets out of backbone
  - Commercial ISPs take off
- BGP4: **Path vector protocol** between competing ISPs, who must yet cooperate
- 1996-2001: .com bubble starts and bursts
- 2000s: Internet now truly international; more non-PC devices than PCs
- Wireless and mobility take off…

Example Security Problem: Route Hijacks

- In Feb 2008, Pakistani government wanted Pakistan Telecom (PT) to block YouTube
  - PT advertised its own host as the destination for YouTube's IP address range ← cheaper than any other route
- Misconfiguration causes this advert to propagate to PT's ISP (PCCW, Hong Kong)
- PCCW sees that this advert is "more specific" than what it has, so accepts
  - Propagates to other ISPs, who also accept
- Soon, much of the Internet wasn't able to reach YouTube!

The whole network was built on trust
Need to stop trusting things
Authentication
Still in infancy ~~state~~ state

Some Big Challenges

- A largely mobile, wireless world
- Security: coping with errors and malice
- Availability and reliability improvements
- Flexibility and evolution of the network
- Large-scale video, collaboration, and "network neutrality"
  - 2010 factoid: Netflix consumes 21% of Internet bandwidth during prime time

-Entertainment world

5/4

To save your work, click the SAVE button at the bottom of this page. You can revisit this page, revise your answers and SAVE as often as you like.

To submit the assignment, click the SUBMIT button at the bottom of this page. YOU CAN SUBMIT ONLY ONCE. Once the assignment has been submitted, you can continue to view this page but will no longer be able to make any changes to your answers.

6.02 Spring 2011: Plasmeier,Michael E.

PSet PS10

Dates & Deadlines

issued:	Apr-27-2011 at 00:00
due:	May-05-2011 at 06:00
checkoff due:	May-10-2011 at 06:00

Help is available from the staff in the 6.02 lab (38-530) during lab hours -- for the staffing schedule please see the Lab Hours page on the course website. We recommend coming to the lab if you want help debugging your code.

For other questions, please try the 6.02 on-line Q&A forum at Piazzza.

Your answers will be graded by actual human beings, so your answers aren't limited to machine-gradable responses. Some of the questions ask for explanations and it's always good to provide a short explanation of your answer.

Problem 1.

The 802.11 (WiFi) link-layer uses a stop-and-wait protocol to improve link reliability. The protocol works as follows:

- The sender transmits data packet k+1 to the receiver as soon as it receives an ACK for data packet k. (Sequence numbers increment by 1 for each successive data packet sent.)
- After the receiver gets the entire data packet, it computes various

things, including a cyclic redundancy check (CRC). The total processing time at the receiver for a data packet is T_p and you may assume that it does not depend on the packet size.

- If the CRC is correct, the receiver sends a link-layer ACK to the sender. The ACK also incurs some processing time at the receiver.

The sender and receiver are near each other, so you can ignore the propagation delay. The bit rate is R = 54 Megabits/s, the smallest data packet size is 540 bits, and the largest data packet size is 5,400 bits. Assume that the size of an ACK is 108 bits. Assume also that the processing time required to process an ACK is one-half the time required to process a data packet (i.e., it is equal to $T_p/2$). (These numbers aren't the same as in the WiFi standard, but have been picked to make the calculations easier.)

What is the maximum processing time T_p that ensures that the protocol will achieve a throughput of at least two-thirds of the bit rate of the link in the absence of packet and ACK losses, for any packet size in the given range?

math qv

1

(points: 1)

Problem 2. Sliding window utilization

The bottleneck (i.e., slowest) link on a network path between two computers, A and B, has a bit rate of 1 Megabyte/s and the round-trip time (RTT) between the computers is 60 milliseconds. The queueing and processing delays are negligible. You run a sliding window protocol between the two computers to reliably send data, with a packet size of 500 bytes and a window size of 20 packets.

A. What is the maximum utilization of the bottleneck link along this path assuming no lost packets or ACKs?

(points: 0.5)

B. Which of the following techniques will double the utilization of the bottleneck link? Assume that no packets are lost.

A. Double the window size. ✓
B. Double the speed of the bottleneck link. ✓
C. Halve the speed of the bottleneck link. ✗
D. Halve the RTT of the network path. ✓

Speed ~~total~~ is halve RTT

$$\text{Util} = \frac{\text{packet size} \cdot \text{window size} \cdot \frac{1\,\text{sec}}{RTT}}{\text{bandwidth}}$$

$T = \frac{1}{RTT}$

10 → 20 ⇒

1 → .2

.05

(points: 0.5)

Problem 3. Fritter

Stunned by Twitter's success, Alyssa P. Hacker starts Fritter, a service that thinks that 140-byte Twitter messages are 100 too many for the next generation: Fritter messages are only 40 bytes long. Fritter has a simple request-response interface. The client sends a request (called a frequest, which Fritter's Marketing person promptly trademarks), to which the server sends a response (a fresponse, also trademarked). The frequest fits in one 40-byte packet, as does the fresponse. When the client gets a fresponse, it immediately sends the next frequest. + headers ?

Alyssa's server is in Cambridge. Clients come from all over the world. Alyssa's measurements show that one can model the typical client as having a 100 millisecond round-trip time (RTT) to the server.

Each client connects to Fritter, sends some number of unique frequests, and after some period of frittering (their time) away, leaves. If a client does not get a fresponse from the server to a particular frequest in a timeout duration T, it assumes (correctly) that either the frequest or fresponse were lost, and resends the frequest. It keeps doing that until it gets a fresponse.

Normal packet system

A. Alyssa needs to provision the link bandwidth for Fritter. She hopes to be wildly successful, and anticipates that at any given time, the largest number of clients making frequests is 10000. What minimum outgoing link bandwidth from Fritter will ensure that all the fresponses for one round of frequests will be delivered before the next round of frequests starts to arrive? Give your answer in *megabits* per second.

4

(points: 0.33)

B. Suppose the probability of the client receiving a fresponse from the server for any given frequest is p. What is the expected time for a client's frequest to obtain a fresponse from the server? Your answer will depend on p, RTT, and T. Give your answer (in seconds) when p=0.8, RTT=0.2 seconds, and T = 1.0 seconds.

.225

(points: 0.33)

C. Alyssa now wants to increase the rate at which each client is able to get fresponses using a windowed protocol where the client can send successive frequests while awaiting fresponses for earlier ones. If the data rate from the server to the client is C=10000 bytes/s and the RTT (as before) = 100 milliseconds, what is the smallest number of outstanding (i.e., awaiting a fresponse) frequests that the client should make to achieve the highest possible throughput? Assume that no packets are lost.

25

(points: 0.34)

Problem 4. Timeouts

We discussed how a reliable transport protocol sender computes an average round-trip time (RTT) for the connection using an exponential weighted moving average (EWMA) estimator:

$s(n) = \alpha*r(n) + (1-\alpha)*s(n-1)$,

where r(n) is the n^{th} RTT sample and s(n) the n^{th} smoothed RTT estimate updated after r(n) arrives.

Suppose that at time 0, the initial estimate, s(0), is equal to the true value, r(0). Suppose that immediately after this time, the RTT for the connection increases to a value R >> r(0) (i.e., much larger than r(0)) and remains at that value for the remainder of the connection.

Suppose that the retransmission timeout value at step n, TIMEOUT(n), is set to 2*s(n), and α = 1/8 (old TCP implementations used this method for their timeout; modern TCPs use the linear deviation, as we studied in class). Calculate the number of RTT samples before we can be sure that there will be no spurious retransmissions. Note that your answer should be an integer.

6

(points: 1)

Problem 5.

In the reliable transport protocols we studied, the receiver sends an acknowledgment (ACK) saying "I got k" whenever it receives a packet with sequence number k. Ben Bitdiddle invents a different method using cumulative ACKs: whenever the receiver gets a packet, whether in order or not, it sends an ACK saying "I got every packet up to and including L", where L is the highest, in-order packet received so far.

The definition of the window is the same as before: a window size of W means that the maximum number of unacknowledged packets is W. Every time the sender gets an ACK, it may transmit one or more packets, within the constraint of the window size. It also implements a timeout mechanism to retransmit packets that it believes are lost using the algorithm from class and pset.

Network assumptions: The protocol runs over a best-effort network, but no packet or ACK is duplicated at the network ~~or link layers.~~

A. The sender sends a stream of new packets according Cool to the sliding window protocol, and in response gets the following cumulative ACKs from the receiver:

1 2 3 4 4 4 4 4 4 4
5 6 7 8 9 10

Suppose that the sender times out and retransmits the first unacknowledged packet. When the receiver gets that retransmitted packet, which of the following is the best you can say about the ACK, a, that it sends?

a. a = 5 ✓ can say
b. a ≥ 5 ✓ better
c. 5 ≤ a ≤ 11 better still
d. a = 11 No
e. a ≤ 11 ok

c

(points: 0.25)

B. Is it possible for the sequence of cumulative ACKs given above to have arrived at the sender even when no packets were lost en route to the receiver when they were sent?

Other ACK lost? -no Arrived out of order -but then # would be here

(points: 0.25)

C. A little bit into the data transfer, the sender observes the following sequence of cumulative ACKs sent from the receiver:

6 26 27
21 22 23 25 28
24

The window size is 8 packets. What packet(s) should the sender transmit upon receiving each of the above ACKs, if it wants to maximize the number of unacknowledged packets?

???? for each

What has it sent up to now? ?? Confused

(points: 0.25)

D. Give one example of a situation where the cumulative ACK protocol gets higher throughput than the sliding window protocol described in class and the pset.

When packets out of order and it completes a large batch. ~~Ab~~

(points: 0.25)

"or does it" when all recieved same time

If Ack 24 lost

Problem 6. Windows and queues

A sender S and receiver R communicate reliably over a series of links using a sliding window protocol with some window size, W packets. The path between S and R has one bottleneck link (i.e., one link whose rate bounds the throughput that can be achieved), whose data rate is C = 1000 packets/second. When the window size is W the queue at the bottleneck link is always full, with Q data packets in it. The round trip time (RTT) of the connection between S and R during this data transfer with window size W is T = 50 milliseconds, **including** the queueing delay. When the queue size is 0, the RTT of the connection is 20 milliseconds. There are no data packet or ACK losses, and there are no other connections sharing this path.

What is the value of Q? Littles Law

(points: 1)

Introduction to Python tasks

In this lab, you will develop the core logic of the `ReliableSenderNode` and `ReliableReceiverNode` classes. These two classes implement the functions of a *reliable data transport protocol* between sender and receiver that delivers packets *reliably and in order* to the receiving application. The sender and receiver are connected over one or more network hops via nodes of the Router class. You don't have to worry about how routing and forwarding work in this lab.

This lab has two tasks. You will write the code for the main components of a *stop-and-wait protocol* and a *sliding window protocol* (with fixed window size). Your code will involve both the sending and receiving sides of the protocol.

You can run the Python programs for this lab using the python command line; this lab will not work in IDLE.

To understand the different parameters in NetSim for this lab, go to a shell and enter:

```
# python PS10_1.py -h
```

```
# python PS10_2.py -h
```

The programs take the following options:

- `-l LOSS_PROB` causes both data and ACK packets to be lost on *each link* with the specified `LOSS_PROB`, which must be a number between 0 and 1. The default per-link loss probability is 0.01.
- `-b BOTTLENECK_RATE` specifies the rate of the bottleneck link between sender and receiver, in packets per time slot. It should be between 0 and 1 (default is 1).
- `-q QUEUE_SIZE` specifies the queue size at each node for each link (default is 10 packets).
- `-g` runs the program with the GUI, useful for debugging.
- `-t SIMTIME` sets the number of time slots in the simulation (default is 5000).
- `-v` runs the program in "verbose" mode; in this mode, the program prints out a line whenever the sender gets an ACK, the receiver application gets a packet, or a packet is dropped on a link. ← nice

In addition, for the sliding window protocol, the following option is important:

- `-w WINDOW_SIZE` sets the window size (in number of packets); the maximum number of *unacknowledged* packets sent by the sender must not exceed this window size setting.

This lab has two main tasks that implement two different reliable transport protocols. Each task has a few sub-tasks. You will test these implementations on the test topology that will be generated when you run the corresponding task files.

Debugging and Testing Procedures

At any point in the simulation, when you run the programs with the `-g` option, you can click on the sender, node S, to see an estimate of the round trip time (RTT) to the receiver (R) and the current timeout being used by the sender. Clicking on R will show the current throughput at the receiver measured at the number of useful (i.e., unique and in-order) packets passed up to the application per time slot, as well as the total number of spurious (duplicate) packets received. We will judge the quality of your transport protocol by the throughput printed at the end of the simulation, and also by the RTT measurements displayed at the end.

If your protocols work correctly, two things should happen:

1. *No deadlocks.* Neither the receiver nor the sender should "hang" -- i.e., the protocol should not deadlock with both sides waiting for something to happen.
2. *In-order delivery.* The receiver should not print an error message and terminate the program. That will happen if your protocol delivers an out-of-sequence packet to the receiving application in the `app_receive` call, as explained below.

If the protocol is correct (i.e., no deadlocks and only in-order delivery), then the only potential problems that might remain are performance problems: the protocol may be unacceptably slow or unexpectedly sluggish. Obviously, we want your protocol to come as close as possible to the theoretically expected performance. The performance metric of interest is the *receiver throughput, measured in packets per time slot.* By default, the receiver throughput is printed by the programs only at the end of the simulation, but if you use the `-v` (verbose) option, it will be printed roughly every 100 time slots (more precisely, it is printed the first time the receiver application gets a new in-order packet at least 100 time slots since the last such event). Note that if you want to extract only the throughput numbers in the verbose mode, you can use the `grep` utility, running a command such as:

```
python PS10_2py -w 12 -l 0.02 -v | grep throughput
```

In addition, you can, and probably should, initially run the program with the GUI on for debugging, and click on the sender (S) and receiver (R) in the GUI to view useful information about the performance of your implementation.

To help debug your code, you can print out your own debug statements whenever a significant event occurs at the sender or receiver. You can also see the total number of pending packets at various nodes on the bottom panel of the simulator; if this number is persistently in the hundreds, then very likely something is wrong, especially if the window size is much smaller than the number of pending packets.

You can use the `-l` option to test both the correctness and the performance of your protocols at different link loss rates. Note that both packets and ACKs will get lost, and the value set is the per-link packet loss probability.

Another test worth running (which we may do during check-off) is to introduce variable *cross-traffic* into the network. You can do that using the `-x` option, setting a value between 0 and 1 as the rate of the cross traffic on the bottleneck link. Note that cross-traffic will both take away from the link bandwidth available for your data transfer, and will make the round-trip times more variable.

In this lab, each data packet and ACK are the same size, 1 time slot long. Each link sends one packet per time slot. Hence the maximum possible throughput of the protocol is 1 packet per time slot; because of packet losses and cross traffic, you won't achieve that maximum, but your goal should be to maximize throughput while providing reliable, in-order delivery.

Please note: In both tasks below, please use the variable names `srtt` for the estimate of the smoothed RTT, `rttdev` for the estimate of the mean linear RTT deviation, and `timeout` for the sender's retansmission timeout value. These variables should be members of the `ReliableSenderNode` class. All these quantities will of course vary with time in your protocol, and you will write the code to maintain these values. By using the variable names as mentioned here, the debugging and diagnostics information obtained when you click on the sender node will be correct (the diagnostics code assumes that these variables exist).

Note: This lab is best done either after reading the notes for the transport protocol lectures, or in conjunction with reading them. as always!

Python Task #1: Stop-and-wait protocol

Useful download links:

> PS10_netsim.py -- network simulator (modified for this lab)
> PS10_1.py -- template file for this task

The stop-and-wait protocol works as follows:

- Each data packet includes a unique sequence number, derived from a counter that is incremented for each new data packet (see the lecture notes, § 20.2). Each data packet also includes a timestamp giving the time at which the data packet was transmitted by the sender. A data packet is saved by the sender until it receives an ACK from the receiver. You may use `p.start` to set and view the time at which a packet `p` was sent.

- In each time slot the sender's `reliable_send` method is called. This method decides what packet to send, if any. It has two choices:
  - If there is a saved data packet, the sender should check to see if `self.timeout` slots have passed since the transmission by comparing the current time with the timestamp of the saved packet. If `self.timeout` slots have passed, the sender should retransmit the saved data packet, using the old sequence number *but with a new timestamp* that indicates the time of the retransmission. If `self.timeout` slots have not passed, the sender should not retransmit the packet.

    Only one saved

  - If there is no saved packet, the sender should increment the sequence number counter and send a new data packet, remembering to also save the data packet pending the receipt of an ACK.

- When a data packet arrives at the receiver, the receiver's **`reliable_receive`** method is called in the code provided to you. This method, which you will write, should send an ACK to the sender (i.e., to the address specified in the source field of the received packet). The ACK should include the sequence number and timestamp of the received data packet. The method should then deliver the data packet to the application by calling the `app_receive` method. Note that that if an ACK is lost, the receiver may receive the same packet more than once and must take steps to ensure only one copy of the packet is delivered to the application (see §20.2.2) Note that `app_receive` maintains the `self.app_seqnum` instance variable, which indicates the sequence number of the last packet packet to be delivered to the application. `self.app_seqnum+1` should be the sequence number of the next packet to be delivered to the application. This instance variable may be useful in your code; please feel free to use it. The "time" value passed to the call to app_receive should be the time at which the function is called, not the time at which the packet being passed to app_receive originally arrived (for out-of-order packets, the two times won't be the same).

- When an ACK arrives at the sender, the sender's process_ack method is called by the code provided to you. process_ack should clear the saved packet, indicating that the next call to reliable_send is free to send the next data packet. It should also call the calc_timeout method to compute the ACKed packet's round trip time and update the value of self.timeout, using the formulas described in §20.3 of the lecture notes (or something better, if you can improve on that).

In this task, you will implement the following functions in ReliableSenderNode:

reliable_send(self,time)

This function, invoked every time slot at the sender, decides if the sender should (1) do nothing, (2) retransmit the previous data packet due to a timeout, or (3) send a new data packet. timeis the current network time measure in time slots. Please use the send_pkt can be used to build and send a data packet -- see the definition of this function in PS10_1.py for details about the calling sequence. send_pkt also returns the packet it transmitted, so it can be saved until an ACK for it arrives.

current time packet timestamp

process_ack(self, time, acknum, timestamp)

The code provided to you invokes this function whenever an ACK arrives. time is the network time when the ACK was received, acknum is the sequence number of the packet being acknowledged, and timestamp is the sender's timestamp that is echoed in the ACK. This function must call the calc_timeout function described below, among other things.

calc_timeout(self, time, timestamp)

This function should be called by your process_ack method to compute the most recent data packet's round trip time (RTT) and then recompute the value of self.timeout.

In ReliableReceiverNode please implement:

reliable_recv(self, sender, time, seqnum, timestamp)

The code provided to you invokes this function at the receiver upon receiving a data packet from the sender. You can use the send_ack method to build and transmit the ACK packet.

Note: If you introduce additional instance variables in the sender or receiver, you should add the appropriate initialization code to the `reset` method for the class.

The template code we have provided has additional comments that you may find helpful; please read them.

After writing the required functions, run the protocol for a few different link loss rates. Observe the throughput; click on the sender and observe the RTT and timeout estimates. These observations will help you answer the lab questions for this task.

When your code is working, pleease submit it on-line:

Upload code for Task 1:

Browse...

(points: 10)

Run the following (you may use the `-g` option if you wish):

```
python PS10_1.py -l .04
```

What is the throughput of your protocol? Looking at the reported srtt and timeout, can you come up with a model and explanation for why your throughput is as observed? Note that the **per-link** loss rate in this experiment is 0.04, and that the number of hops in the topology is 12.

.03 timeout climes to 2 the RTT

(points: 1)

If we reduced the number of hops between sender and receiver, and kept everything else unchanged, would the throughput increase or decrease? Explain your answer.

(points: 1)

If we changed the timeout to be a little smaller than the RTT, would the throughput increase or reduce? Explain your answer.

actually ↑

(points: 1)

Python Task #2: Sliding window protocol

Useful download links:

- PS10_2.py -- template file for this task
- PS10_runsliding.sh -- script file for questions
- PS10_prac-theory.py -- python plotter for questions

The sliding window protocol extends the stop-and-wait protocol by allowing the sender to have multiple packets outstanding (i.e., unacknowledged) at any given time. In this protocol, the maximum number of unacknowledged packets at the sender cannot exceed its window size, and is specified on the command line of the program with the `-w` option (default is 1). The window size is available as `self.window`.

Upon receiving a packet, the receiver sends an ACK for the packet's sequence number as before. The receiver then buffers the received packet and delivers each packet in sequence number order to the application. Check out §20.4.2 of the lecture notes to understand the semantics of the protocol and how it works.

In this task, you will implement the same functions that you did for the previous task, but with the necessary modifications to handle window sizes bigger than 1. A naive way of implementing the receiver will be to throw away all out-of-order packets, but this approach will have low throughput. A better strategy will be to create a buffer and deliver packets in order to the application.

You should copy over the `calc_timeout` function from the previous task; if you implemented it correctly, then it won't have to change. When you run your code with a window size of 1 (which is the default), you must get the same throughput as you did in the previous task; this will serve as a necessary (*but not sufficient*) sanity check for the correctness of your sliding window protocol.

The template code we have provided has additional comments that you may find helpful; please read them.

After writing the required functions, run the protocol for various values of the window size ranging from 1 to 20, and also vary the link loss rate. Observe how throughput depends on window size and think about out reasons for the observed throughput. These observations will help you answer the lab questions for this task.

When your code is working, please submit it on-line:

Upload code for Task 2:

Browse...

(points: 10)

Experiment with the sliding window protocol for a per-link loss rate of 0.02 (`-l 0.02`) and a topology with 5 hops between sender and receiver (`-n 5`), for various window sizes. What is the smallest window size at which the throughput reach its maximum possible value? What is the maximum value in your protocol? Note that you can set the window size using `-w`. You may want to come up with a reasonable initial guess for a good window size and go from there to the right answer.

W = 7 → .51
W = 1 → .88
W=2 → .15
W=3 → .22
W=4 → .30
5 → .36
6 → .43
8 .58
9 .65
10 .71
11 .77
12 .80
13 .82
14 .83
15 .83 ←
16
17
18
19
20

(points: 1)

As you increase the window size from this smallest value, what happens to the RTT? Why?

1 - 16
2 18
3 10
4 10
5
6
7
8

9
10 10.5
11
12
13
14
15
16

20 17.26
30 17.4
58 18.9
100 19

↑ Collisons?
- I don't think so
Note:
No packets sit in queues

(points: 1)

not going to work on windows

Now let's experiment with 6 hops between sender and receiver (the default setting) and a window size of 16 (-w 16). PS10_runsliding.sh is a shell script that runs PS10_2.py with various per-link loss rates. You can use the following commands to capture that output in a file and use PS10_prac-theory.py to produce two plots, one showing the throughput as a function of loss rate ("experiment") and the other showing (1 - loss rate)**12 ("theory" -- recall from lecture that this number is the estimated throughput when there's no variation in the RTT).

let me go down to a station

```
csh PS10_runsliding.sh >data
python PS10_prac-theory.py data
```

Oh they tell you

Please save the plot as a PNG file and then upload your PNG file using the following input field.

Upload PNG file for Task 2:

Browse...

(points: 1)

You can save your work at any time by clicking the Save button below. You can revisit this page, revise your answers and SAVE as often as you like.

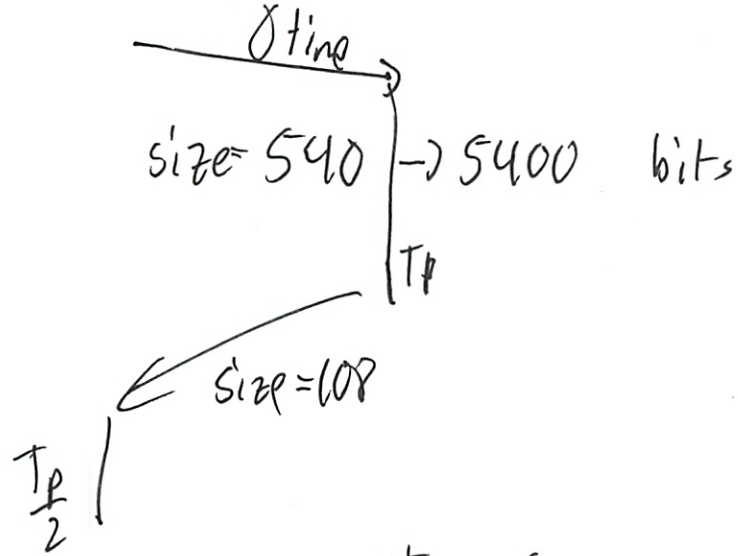
Save

To submit the assignment, click on the Submit button below. YOU CAN SUBMIT ONLY ONCE after which you will not be able to make any further changes to your answers. Once an assignment is submitted, solutions will be visible after the due date and the graders will have access to your answers. When the grading is complete, points and grader comments will be shown on this page.

Submit

PS 10 #1

R = 54 M bits/sec



Max T_p for

flow get $\geq \frac{2}{3}$ of bit rate ~~for~~ That qu

↑ means =

~~Thro~~ Throughput $= \frac{1}{T}$

$T = \frac{3T_P}{2}$ → Throughput $= \frac{2}{3T_P}$

bit rate = bits / seconds

Throgput = bits/sec sucessful

So $\frac{2}{3} \leq \frac{2}{3T_p}$ $T_p = 1$

↑ but to make ~~TP~~ ↑ must make T_p ↓

2

Wait

$$\frac{2}{3}\,54{,}000{,}000 = \frac{2}{3TP_i}$$

where does ~~fit in~~ it fit in?

$$36{,}000{,}000 \le \frac{2}{3TP}$$

$$T_p = \frac{1}{54{,}000{,}000}$$

Actually makes more sense

Relook

Need to consider packet size

$$T_p + \frac{T_p}{2} + \frac{\text{Size}}{R}$$

try both

pick one which gives bigger

$$T_p + \frac{T_p}{2} + \frac{5400}{54{,}000{,}000}$$

$$T_p + \frac{T_p}{2} + \frac{1}{10{,}000}$$

or $\frac{1}{100{,}000}$

Use larger time to be sure

$\frac{3}{2}T_P$

$T_p = 53\ 985\ 000$ -even worse

I'm ignoring TA's advice

~~(12)~~ (13)

2. Sliding window util

wp Utilization = $\frac{\text{arrival rate}}{\text{service rate}}$

↑ not what I think

$= \frac{\text{Used for data}}{\text{Available}}$

~~Can only send~~

~~So 20~~

If window fully used ← what we care about here?

$$\frac{500 \cdot 20 = 10{,}000}{1{,}000{,}000} = \frac{1}{100}$$

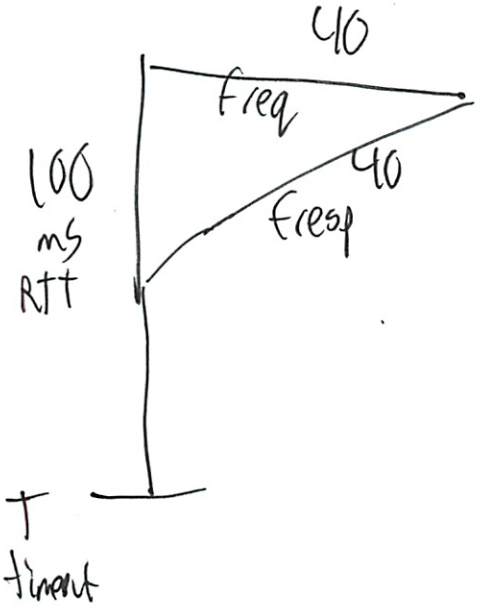
Oh no need time

20 RTTs in 1 sec

$$\frac{500 \cdot 20 \cdot 20}{1{,}000{,}000} = \frac{1}{5}$$

(4)

3a 40 bytes



Stop + wait

10,000 clients

So amt of bandwidth needed

~~40m~~ $10{,}000 \cdot 40 \cdot \frac{1}{100ms}$

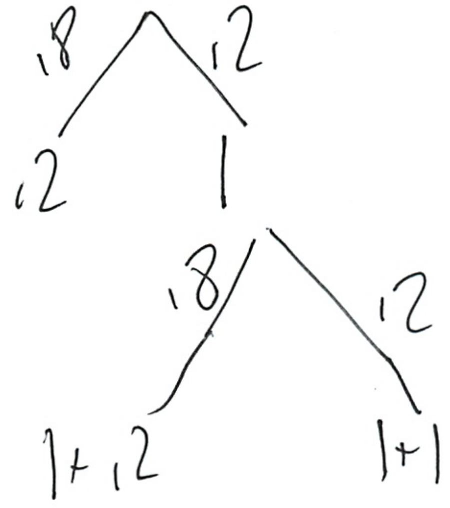
$10{,}000 \cdot 40 \cdot 10$

(Only cares about outgoing
but wants all delivered
(cut RTT in half?
Oh no - 1 in + out (stop + wait) system

= 4 million = 4 megabits

b) (5) Now only recieves w/ p

What is E[ ]   p = .8   RTT = .2   T = 1



etc

did in 6.042 today!

Mean time to failure $= \frac{1}{p} = \frac{1}{.8} = 1.25$ sec

↑ p of failure ? or P(sucess)?
↑
failure

$\frac{1}{.2} = 5$

? This means fail after every 5 ??

just do it the long way

(6)

$s = .2 \cdot .8 + .2 \cdot (1 + s)$

$s = .16 + .2 + .2s$

$s(1 - .2) = .18$

$s = \frac{.18}{.8} = .225$

---

c) Nav window 6

$C = 10,000$ bytes/sec

$RTT = 100$

What is smallest # of outstanding requests for larges throughput

$\text{Throughput} = \frac{1}{T}$ Should = bitrate

$10,000 = \frac{W}{100 \text{ ms}}$ where does w go here?

? .1 sec

✓ confirmed from recitation today

$W = 1000$ does this make sense

8

Throughput = RTT + re transmits

On 10,000 bytes

And packets 40 ~~bits~~ bytes

$$\frac{80,000}{40 \cdot 8} = \frac{w}{.1}$$

$$\frac{10,000}{40} = \frac{w}{.1}$$

$1,000 = 40w$

$w = 25$

TA ✓

8

4. Timeouts

Using exponential weighted moving avg

$$s(n) = \alpha\, r(n) + (1-\alpha)\, s[n-1]$$

nth RTT sample

5th smoothed RTT same

At $t=0$ $s(0) = r(0)$

true value

Then RTT goes to R and remains there

timeout $= 2\, s(n)$

$\alpha = \frac{1}{8}$

Calc # time steps (RTTs) till sure no spurious retransmit

Oh smoothed given above

So what is time out at each step

(i as fn of R?

(9)

	R	S	
$t=0$	0	0	
$t=1$	R	$\frac{1}{8}R$	
$t=2$	R	$\frac{1}{8}R + \frac{7}{8}\left(\frac{1}{8}R\right)$	$= \frac{15}{64}R$
$t=3$		$\frac{1}{8}R + \frac{7}{8}\left(\frac{15R}{64}\right)$	$= \frac{169}{512}R$
		clever some of new and some of old	
$t=4$			$= \frac{1695}{4096}R$
$t=5$			$= \frac{15961}{32768}R$ .48
$t=6$			$= \frac{144495}{262144}R = .55$

When is $RTT \leq 2(S)$

$$\frac{R}{2} = S$$

so when are these fancy fractions past $\frac{1}{2}$

$t=6$

10

5. On paper

6. Window + Queues

Littles law

Queue time = 30 ms

$L = \lambda W$ ← time staying in store

↑ # in system

arrival rate

?? so how does that fit in

? or # in queue

or leaving rate for quee

? or queue time

0 queue size = 20 sec

~~where is lenght of queue~~

~~or is that arrival rate~~ No

$0 = Q \cdot 20$

? No

$Q = \lambda \cdot 30$

No this seems wrong!

(11)

Look up

recitation

λ = rate served sucessfully

D = Delay

Q = avg # people in sys

$Q = \lambda D$

---

lecture

P packets forwarded in T

Rate $= \lambda = \frac{P}{T}$

Mean # packets in queue $= N = \frac{A}{T}$

A = aggregate delay

mean delay $D = \frac{A}{P}$

$N = \lambda D$

Q ~~W~~ = 1000 packets/sec · 30 ms sec delay

↑ # in queue ↑ rate leaving

so ~~30,000 packets~~

ms = .03 sec so 30 packets

✓ makes sense - make sure to know

Coding PS 10

5/4

What is packet?

Where to save it?

How to retransmit?

Oh I have to save seq #

ϕ_a is what?

Nice they send

Now rec

- if seq is ~~4~~ ~~the~~ 1 + current
  - deliver
  - Send ACK
- otherwise do nothing

What are the 2 time stamps?

Now process ACKs

Now what is timeout - how to compute

- Section 20.3

Oh the smoothed

$$s_n = \alpha r_n + (1-\alpha) s_{n-1}$$

②

~~SRTT~~ srtt
rtdev
timeout

What is linear thing?
Do I have to use?

Why are all packets dropping?

Need to mess up my nice code w/ lots of prints

~~How~~ works...

Each time stuck diff part
→ start at packet 1

When does it say "Dropping packet"?

Oh not forwarding saved packets
→ use make packet process again?
w/ new timestamp

Is there any data / payload?
Oh hardcoded "DATA"

(3)

~~Q~~ ? Retranx too often?

Oh got > and < flipped

I am not good at thinking about that!

Now not sending ACKs!

~~Q~~ Since was starting ACKs at 0 not 1

Oh Dropping is when network drops it

Not handling - why?

Oh when the ACKs fail!

When get < want

- send ACK
- but not to app

Ok works w/ Throughput .07

Timeout not updating?

On wrong sign

Float 1/8 not working

(4)

Fixed but throughput = .07

Might want to do notes method

---

dev_samle = |rtt_sample − srtt|

Sttdev = β dev sample + (1 − β) Srttdev

timeout = srtt + k · srtt dev
↑4

throughput = .05

What is the best? / good?

Is my timestamp wrong?

Don't think so

When I change timeout to .5 RTT - ~~the~~ throughput ~~actuall~~ actually ↑!

- Oh supposed to

1/12 ~ .08 is best throughput

#2 More on own on this one

need to save buffer of packets on both sides

¿Can you trans more than one on a time step?

Should have kept new scheme in mind as building this

Reciever - build buffer

- always send ack

Just append seq #

- Make easy
- no real data!

Good time should be current time!

For Tx buffer - just do seq # - no data!

- No need time-down

Caught incrementing timestep

Think kinda work

No - pop is wrong!

Need a pop - based on value!

↳ remove!

0

Now one w/ ack filter in

At home

How get packet # from the format?

Found it - They could have made that much eaiser!

Seems to be working now!

But not clearing stuff

(windows terminal is much worse!)

Think not Re-transmitting

(Oh "more" works in windows)

Oh the remove needs to be done

Add print statements

So much going on now!

- Use gui?

It never ~~clears~~ sends the back log

Or it ~~even~~ always removes items?

(3)

But where is it doing that?

? Perhaps make my own w/ time ~~and~~ and seq #

Still getting cleared!

Or name collision self. packets taken

Make something else!

Perhaps still want my time, # tuple

- easier to see!

How to drop?

Got it

Now to add back retransmit

Why verbose not working?

Always RTX

Messed up < again!

Throughput = .01 - worked though

Doing multiple retransmit for a time step

Not reappend rtx

④

Now through put = .47 much better

Try w=1

(suppose to have been!)

Yeah command not working

Oh no space

Still .54!

Oh another place did not change but

Now .07 - good

Back to w=7 = .48

Nice, now qv

Oh can improve by sending multiple in 1 time step
Oh well

6.02 Quiz 3 Review

in Recitation

Topics

- Modulation
- Network Laws
  - Little's Law
  - Architecture Philosophy
- Routing (shortest path)
  - ~~Wind~~ Distance vector
  - Link state
- Rate control/congestion control
  - Send + wait
  - Sliding window

(Modulation will be the big issue)

Modulation

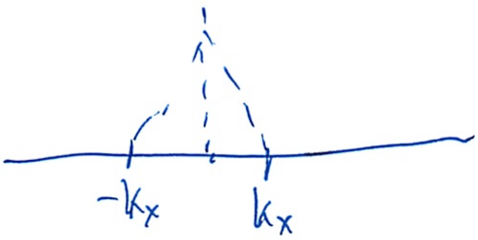
Why talk about freq?

Have a band assigned to us to transmit in

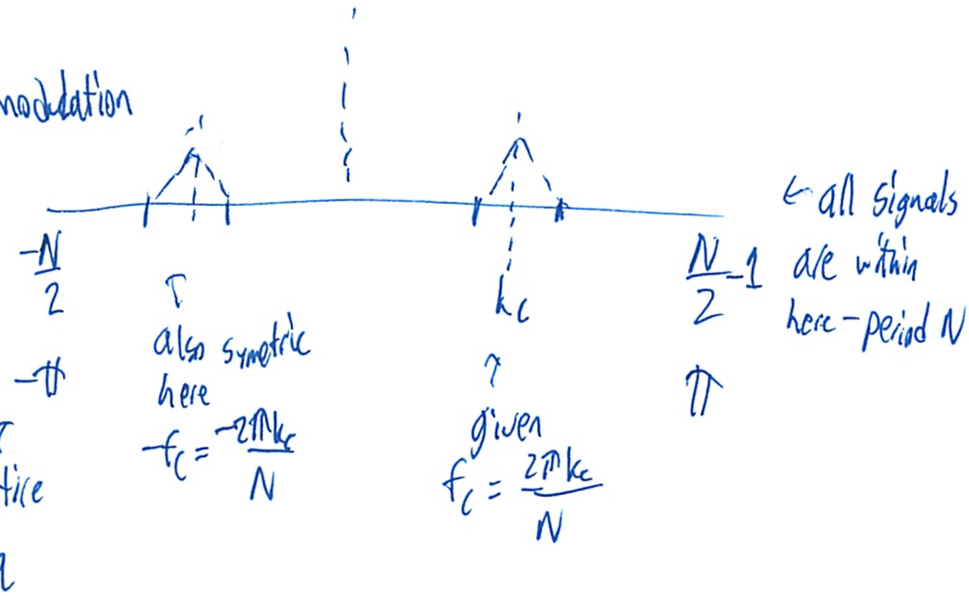
Modulation makes sure you only transmit within this band

2

Periodic signal Period N $\frac{2\pi k}{N}$



modulation



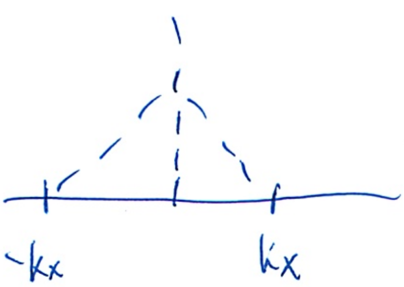
Change freq so only transmit in band where allowed to

Demodulation

We have the signal

- now with noise
- but freq does not change

Hopefully get original thing back



baseband (around 0)
band limited

③

Now we know what we want

But how to get it?

$$x[n] \rightarrow \otimes \rightarrow y[n] \rightarrow \boxed{\text{LTI}}$$

$$\uparrow$$

$$\cos\left(\frac{2\pi k_c}{N} n\right)$$

$\cos(f_c\, n)$ ↲ shortcut/alias

take input, multiply by cos

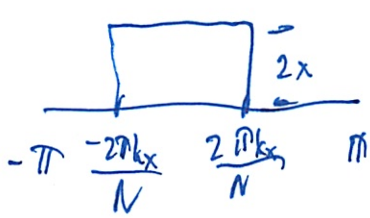
(Review my code to get better understanding)

$$\boxed{\text{LTI}} \xrightarrow{y[n]} \otimes \xrightarrow{z[n]} \boxed{\text{LPF}} \xrightarrow{w[n]} \text{demodulated signal}$$

$$\uparrow$$

$$\cos\left(\frac{2\pi k_c}{N} n\right)$$

multiply by same cos!



(4)

shortcut $f_c = \frac{2\pi k_c}{N}$

$$y[n] = x[n] \cdot \cos(f_c n)$$

$$= x[n] \left(\frac{e^{jf_c n} + e^{-jf_c n}}{2} \right)$$

shifts spectrum

$$z[n] = y[n] \cdot \cos(f_c n)$$

$$= x[n] \cdot \frac{1}{4} \left(e^{jf_c n} + e^{-jf_c n} \right) \left(e^{jf_c n} + e^{-jf_c n} \right)$$

what ends up happening

when these multiply

$$e^{2jf_c n} + e^{0jf_c n} + e^{-2jf_c n}$$

is 1

$$= x[n] \cdot \frac{1}{4} \left(2 + e^{j2f_c n} + e^{-j2f_c n} \right)$$

$\frac{1}{4}$ of original

kept half of signal where it was

push to right by $2f_c$

(5)

Now LPF

- gets rid of $e^{j2f_c n} + e^{-j2f_c n}$
- only keep center

$= \frac{x[n]}{2}$ ~~...~~

Then add 2 gain w/ amplifier

$= x[n]$

If $f_c = \frac{3\pi}{4}$

needs to be not close to π

So $f_c \le \frac{\pi}{2}$ So it works visually $k_n \le \frac{N}{4}$

But what happens when problems?

Like Phase Shift

From lack of sync of clocks

Say the small ns don't match

I have $n=0$, you have $n=4$

So what you are seeing is really $n-D$

which happens to $= 4$ now

(6)

$$\cos\left(\frac{2\pi k_c}{N}(n+D)\right)$$

$$= \cos(f_c n + f_c D)$$

How does this change calculations?

The ~~y[n]~~ modulation part is same

Demodulation is what changes

$$z^D[n] = y[n] \cos(f_c n + f_c D)$$

$$= x[n] \frac{1}{4} \underbrace{\left(e^{jf_c n} + e^{-jf_c n}\right)}_{\text{from modulation}} \underbrace{\left(e^{jf_c n + jf_c D} + e^{-jf_c n - jf_c D}\right)}_{\text{added for demodulation}}$$

So for exp 1

$$jn\left(2f_c + \frac{D}{n}\right)$$

exp 2

$$-jn\left(2f_c + \frac{D}{n}\right)$$

D is same for transmission

After LPF

$$= \underbrace{\left(e^{-jf_c D} + e^{jf_c D}\right)}_{\text{what we get}} \frac{1}{4} x[n]$$

constant, does not change w/ N

Check - put D = 0

That is where the 2 went!

7

$$= \left(e^{-jf_cD} + e^{jf_cD}\right)\frac{1}{4}x[n]$$

this is 2

$$= \frac{2}{4}\cos(f_cD) \cdot x[n]$$

So if $D=0$ - its same

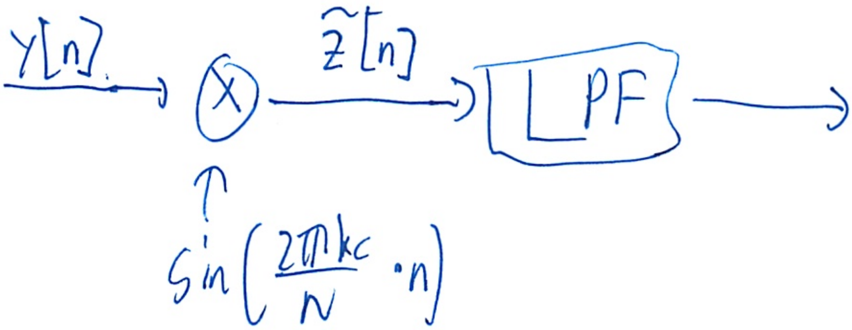
$= \frac{\pi}{2}$ - you lose everything

$D =$ something else - get a little bit of signal

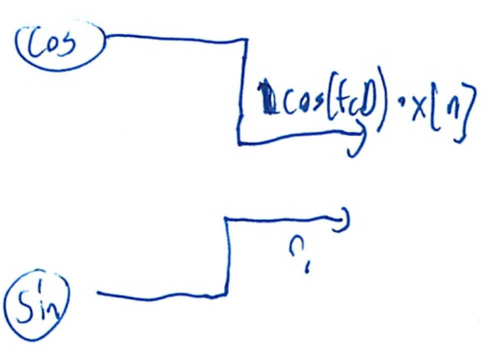
But how to fix this?

Look at sins!

So take $y[n]$ and multiply by sin



So



8

$$z^{0}[n] = y[n] \cdot \sin(f_c n + f_c D)$$

$$= x[n] \cdot \frac{1}{4}\left[e^{jf_c n} + e^{-jf_c n}\right]\left[-j\, e^{jf_c n + jf_c D} + j\, e^{-jf_c n - jf_c D}\right]$$

the division has been taken care of

So what will the LPF throw away?

- $2f_c n$ don't care
  - will just push signal further away

$$= x[n] \cdot \frac{1}{4}\left(-j\, e^{jf_c D} + j\, e^{-jf_c D}\right)$$

what remains after low pass

$$= \frac{1}{2} \cdot x[n] \cdot \sin(f_c D)$$ turns out to be sin!

So now

cos → $\cos(f_c D) \cdot x[n]$

sin → $\sin(f_c D) \cdot x[n]$

So can combine and get amplitude of the energy!

So you can see if < or > threshhold (normally .5)

9

All the accronyms on the last sheet are based on what was shown today

Quiz is on this general

$$e^{j\theta} = \cos\theta + j\sin\theta$$

$$e^{-j\theta} = \cos\theta - j(\sin\theta)$$

So $e^{j\theta} - e^{-j\theta} = \cos\theta + j\sin\theta \; \cancel{+} (\cos\theta - \cancel{2} j\sin\theta)$

$= \cancel{2\cos\theta}$

$= 2j\sin\theta$

Then $\frac{1}{2}\left(j e^{j\theta} - j e^{-j\theta}\right) = -\sin\theta$

$\boxed{j \cdot j = -1}$

Checkoff 10

5/6

- last weeks stuff

- Only count through in terms of T_p
  - but no propagation delay

~~But have~~

- but have data rate - so limitation
- rate of transmission
- 54 Megabits/sec
- in a certain time - can only send so much through
- takes 10 μs packet

  Ack 2 μs

$$\frac{\text{Data}}{\text{Rate}} =$$

$$\frac{\text{Bytes}}{\frac{\text{bytes}}{\text{sec}}} = \text{sec}$$

Is bottlenecked - so 2x speed ↑ link rate

but if W, RTT unchanged, no difference

(2)

Actually half util

- % that is used

Fritter A

- ? ~~BM~~ minor math error

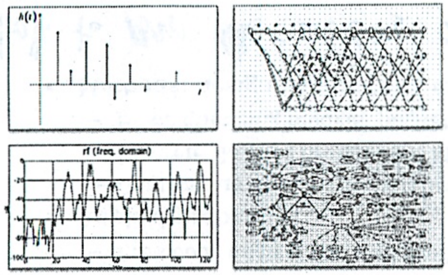
Had extra if statement

Throughput =.05 - should be .04

- small bug
- diff did timeout

Not supposed to send multiple packets at 1 timestep

He likes my why no util change



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #24

- Wrap-up
- Your feedback

HKN Eval! Done

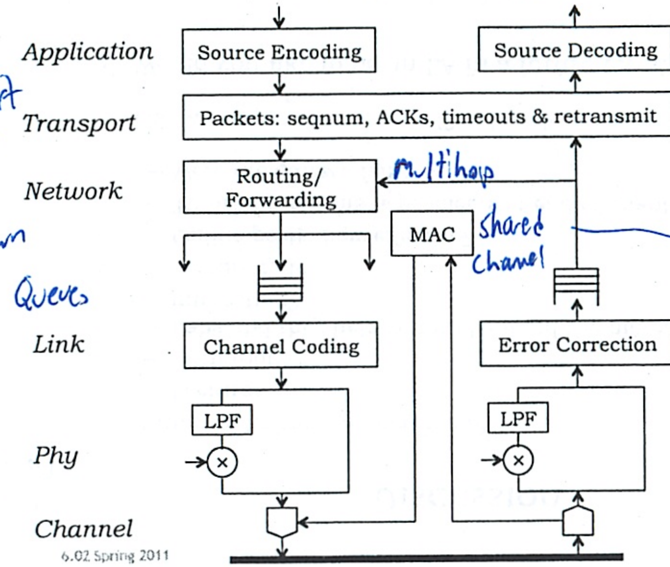
best effort network Simplifies further down

Wrap up class

Quiz 3 5/17 1:30 Johnson

Modeling is important

What We Built



Entropy
Huffman
LZW

taking out bad redundancy

Sliding window
EWMA

Distance-vector
Link-state

TDMA, Contention

Little's Law

CRC, Parity
Block codes
Conv. Codes
Viterbi - lots of redundancy

Digitization - threshold
Modulation

LTI systems
ISI & Deconvolution
Noise & BER

could string stuff together

Reliability

- Digitization abstraction of micro-volt issues
- Error detection
  - Gaussian noise models, predicting BER
  - CRC, checksums
  - Parity, block codes, convolutional codes

defend against errors

- Error correction
  - Single-error correction in block codes
  - Viterbi algorithm: maximum-likelihood decoder
- Reliable sharing via media access control
- Best-effort packet switching
  - Retransmission to recover from dropped packets

continue to send you stuff till hear back

- Approaches to reliability
  - Redundancy
  - Detect failure, invoke recovery mechanism
  - Accurate models → simulation

Sharing & Scalability

- Dedicated links are impossibly expensive
- Time-division multiplexing - simple switching when constant traffic
- Contention protocols for bursty traffic
- Frequency domain
  - Spectrum sharing by bandwidth-limited signals
  - Modulation/demodulation
  - Filters
- Network-level sharing
  - MAC protocols (TDMA, contention)
  - Best-effort packet networks, queues

collecting a lot of traffic

  - ~~Queues~~
- Scalability
  - Use local mechanism instead of global mechanism

did not spend much time on future classes

Approach

- Understand **tools and techniques**
  - Concepts and principles
  - Labs
  - Small problems (calculations, analysis)
- Begin to understand **trade-offs**
  - The essence of all engineering systems
  - Science, art, or a mix?
  - Principles and tools matter, as do intuition and experience

Trade-Offs

- A number of techniques – how to apply them and make them work together?
- Reliability: apply redundancy in creative ways to build reliable systems out of unreliable components where to add reliability
- Sharing: reduce the amount of resources consumed
- Scalability: hide information, reduce amount of state to be managed

EECS 2

~~[illegible]~~

EECS Ideas

30 min in

- Signals and systems
  - LTI, superposition, unit-sample response, frequency response, modulation
- Algorithms, centralized and distributed
  - Viterbi decoding, shortest paths (Dijkstra), distance vector (Bellman-Ford), compression (Huffman, LZW)
- Computer systems
  - Protocols, abstraction and modularity, layering
- Applied probability
  - Continuous-domain probability (PDF & CDF): bit errors
  - Discrete-domain probability: MAC protocol analysis
  - Basic queueing models: packet switch sharing analysis
- Methods: design, simulation, experimentation

talk to ~~[illegible]~~ other advisors about future paths

Discussion

- Which activities worked well?
  - Lectures
  - Recitations
  - Labs: did they help you understand the material? Were they interesting?
  - Lecture notes?
  - Online psets: how effective?
  - Tutorial problems & problems at end of chapters?
  - Were the quizzes fair?
- Did we cover too much? Too little?
- Would you like to be an LA in a future term?

6.02 Feedback

5/9

In Lecture

what people said

- Who was staff in lab?
- Close queue?
- Lab seems widly less used
- Piazza useful
  - good response time
  - visable stuggle
- more visualization
- hardware?
  - Was actually less than last semester
- he is a simulation guy
- lab hrs on weekend
  - Sun or PM
- grading fair?
  - more wiggleroom on subjectivity
  - The takes a while question

2

reuse code - tell to copy + paste

liked automated tests

- less useful at end?

longer understanding own code

~~more~~ more freedom ~~w~~ / less hand holding

but hard to test

Undergraduate edu very good in course 6

- but he wants are to improve

Freq domain - whats the goal?

- more transforms

Overview slide more often

Videos nice

6.02 Recitation

Review 2

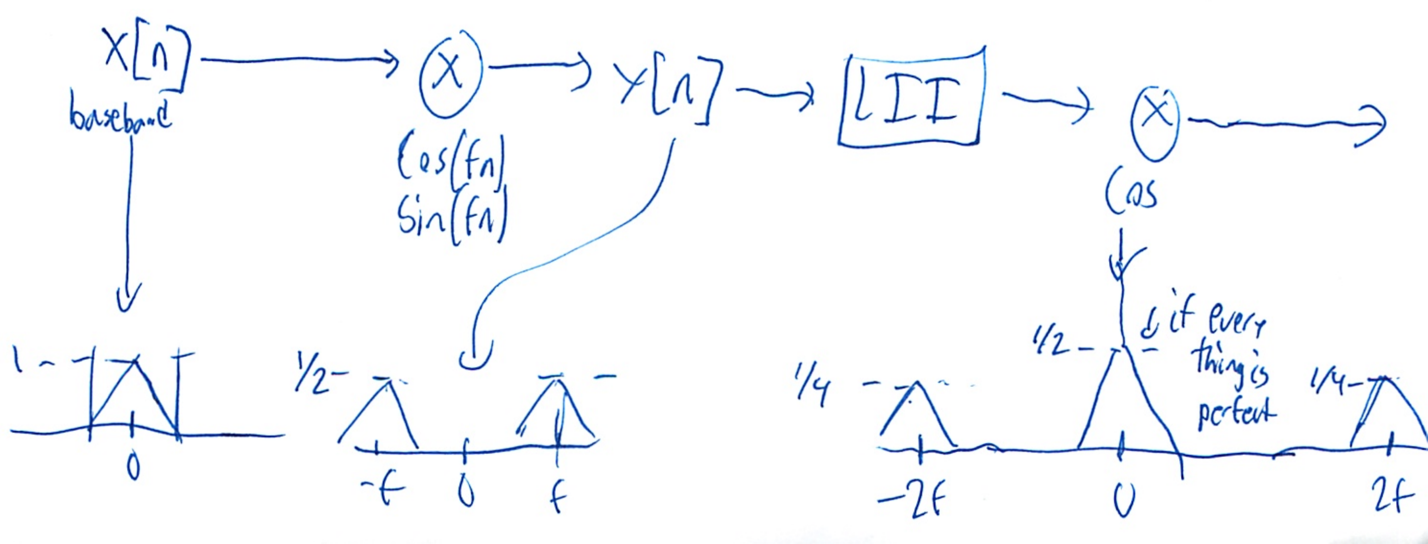
5/10

Today's topics

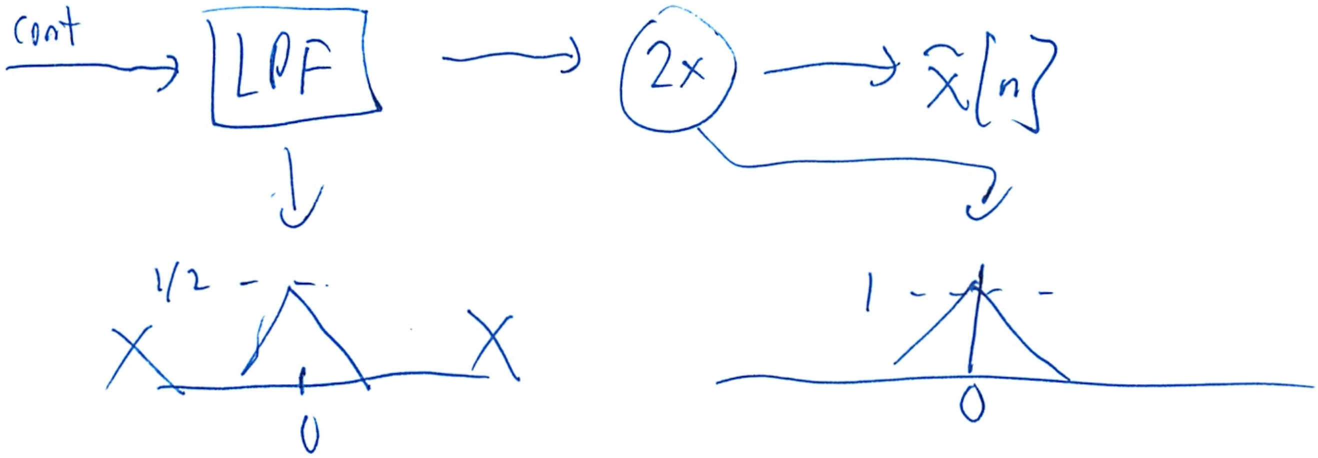
- Modulation BPSK, etc
- Architecture / Laws
  - Little's Law
  - Packet Switched vs Circuit switched
- Routing
  - Distance Vector
  - Link state
- Congestion/Rate control
  - Stop + Wait
  - Sliding Window

BPSK

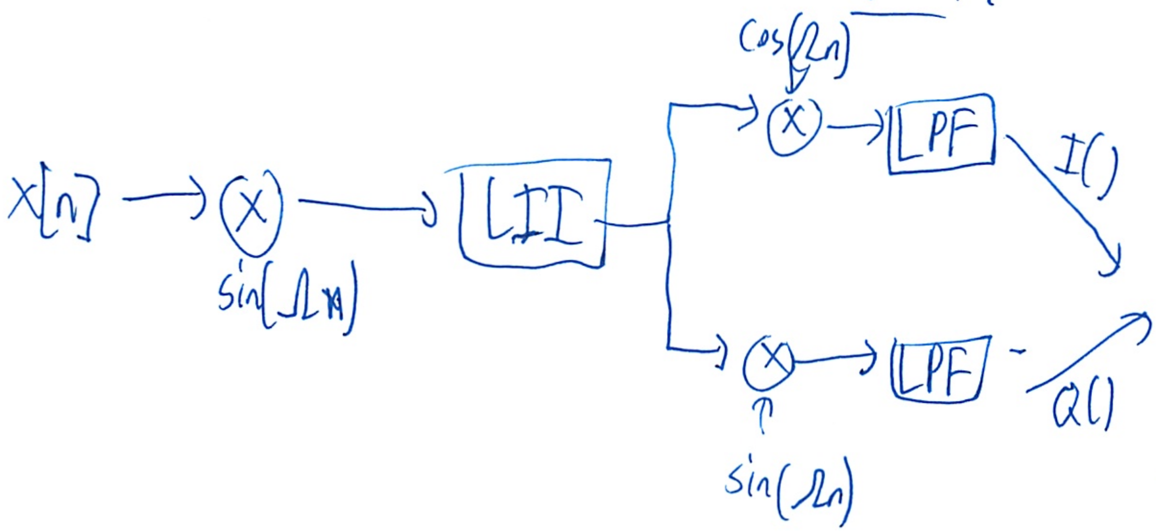
i Modulation

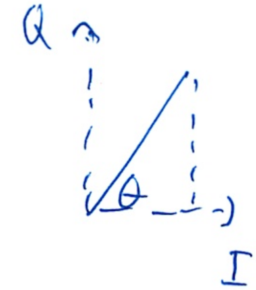


②



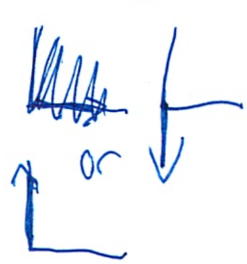
But sometimes delays - so the triangles do not line up exactly on top of each other. Need to do BPSK.





send
(1 or -1)

so angle is either



(3)

$$y[n] = x[n]\sin(\Omega n)$$
$$= x[n]\left(-\frac{j}{2}e^{j\Omega n} + \frac{j}{2}e^{-j\Omega n}\right)$$

Signal we send over wire

Now split

Part that goes through <u>cos</u>

$$z_1[n] = y[n]\cos(\Omega n)$$
$$= x[n]\left(-\frac{j}{2}e^{j\Omega n} + \frac{j}{2}e^{-j\Omega n}\right)\left(\frac{1}{2}e^{j\Omega n} + \frac{1}{2}e^{-j\Omega n}\right)$$

only care about terms that won't be LPFed

$$= \frac{x[n]}{4}\left[\frac{j}{2} - \frac{j}{2}\right] = 0$$

Now part that goes through <u>sin</u>

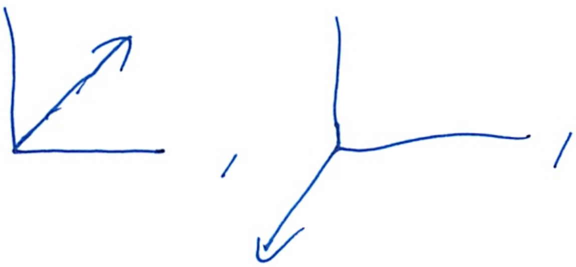
$$z_2[n] = y[n]\sin(\Omega n)$$
$$= x[n]\left(-\frac{j}{2}e^{j\Omega n} + \frac{j}{2}e^{-j\Omega n}\right)\left(-\frac{j}{2}e^{j\Omega n} + \frac{j}{2}e^{-j\Omega n}\right)$$
$$= x[n]\left(-\frac{j^2}{4} - \frac{j^2}{4}\right)$$
$$= \frac{x[n]}{2} \rightarrow \otimes 2 \rightarrow x[n]$$

gain

4

Right, but this is ideal case.

If was delay could go



etc.

Will always be 180° off

Train by sending 0's

Can be add. noise on each signal bit

BPSK tries to fix the angles

Also ~~angle~~ radius you are sending

- Can transmit on both at same time
- purposly put a delay in
- reciever trained to tell them apart
  - where are the axis?

Rotation or Scaling

? delay

? power scaling
gains not right

5

Tx sends

$(r \cos\theta, \ r\sin\theta)$ ← Then convert back

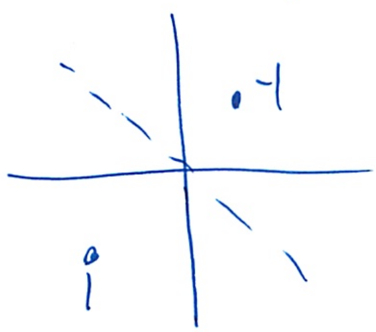
Rx recieves

$\left(\frac{r}{4}\cos(\theta+10), \ \frac{r}{4}\sin(\theta+10)\right)$

All of this is terms discussed in class

Could have denser constelation in very good channels

But BPSK only 2 points



during training

← draw perp line

Anything within $\pm\frac{\pi}{2}$ of -1 is called -1

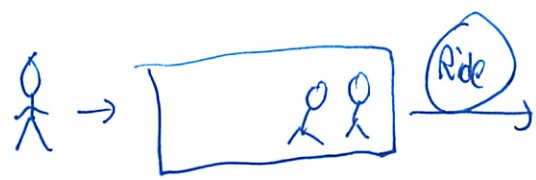
$\pm\frac{\pi}{2}$ 1 1

Use BPSK in very ~~goo~~ bad channels

(6)

Little's Law

- Amusement Park



Queue

$\lambda = 2.5$ person / min

≈ rate of people at exit
But is also same as rate of people leaving queue

Say queue = 100 persons long
- 100 people in line

So $Q = \lambda D$

$100 = 25 \cdot D$

$D = 40$ minutes

≈ wait to get on ride

Can Include Ride in lenght to get time
to tell your mom

⑦

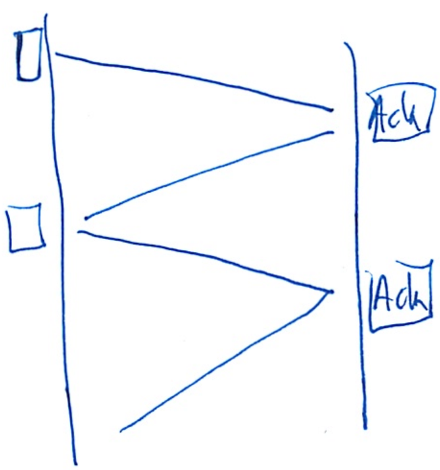
Packet Switched Aritecture

Lecture 18 Slide 23 - comparison slide

vs circuit switched

both have + and -

Stop + Wait



You wait until ~~[illegible]~~ packet comes back

Or timeout

~~[illegible] wait~~
Perfect ↗ $R = \frac{1}{RTT}$ pkt/sec

~~P of loss/drop~~
avg time to transmit 1 packet $= \underbrace{N \cdot \text{timeout}}_{\text{time of dropout}} + \underbrace{RTT}_{\text{normal case}}$

⑧

$(1-p)$ is prob of sucess

How many times do you need to transmit sucessfully?

$$N = \left(\frac{1}{1-p} - 1\right)$$

So ~~prob~~

avg time to send 1 packet $= \left(\frac{1}{1-p} - 1\right) TO + RTT$

$= \frac{p}{1-p} TO + RTT$

$$\text{rate} = \frac{1}{RTT + \frac{p}{1-p} TO}$$

Post: Questions followed the formula!
-(Need to study)

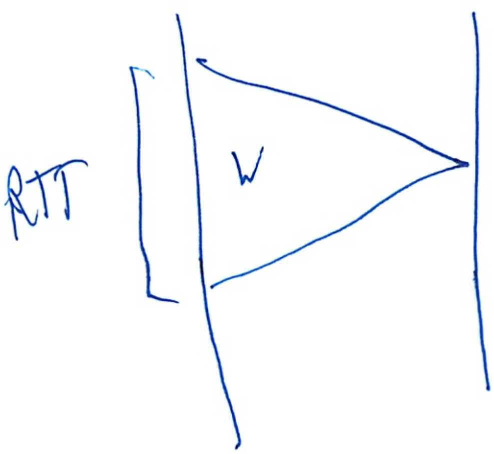
Window

Why not ~~wait~~ send multiple packets and then wait!

Invariant # packets in flight $\leq W$

↑ Need to satisfy this law

But what is right window size?



Rate $= \frac{W}{RTT}$ assuming can keep W packets going

But what if high latency link 1 pk/sec

was $W = 10$

$RTT = .1$

But rate is actually $\min\left(\frac{W}{RTT}, C\right)$

↑ slowest link's capacity

So rate = 1 pkt/sec

10

Window size has been given to us here

Read about Bellman Ford vs Dikstra