6.02 Recitation

3/10

Ask test questions to Chris

2 polynomial generators

Scheme 1 G_1 $\overset{h_0 h_1 h_2 h_3}{1101}$ G_2 1110

$x[0]$ $x[1]$ ...., $x[n]$ ....

$P_1(0)$ ..... $P_1(n)$

$P_2(0)$ .... $P_2(n)$

$P_1[n] = x[n] \cdot h_0 + x[n-1] \cdot h_1 + x[n-2] \cdot h_2 + x[n-3] h_3$

$= x[n] \cdot 1 + x[n-1] \cdot 1 + x[n-3] \cdot 1$

$P_2[n] = x[n] \oplus x[n-1] \oplus x[n-2]$

-Uses G_2

for each input bit/timestep transmit $P_1 P_2$

addition mod 2

1+1 mod 2
add normall
then divide by 2
take remainder

0+0=0
0+1=1
1+0=1
1+1=0

equivilen
XOR

Scheme 2 G_1 110101 G_2 111011

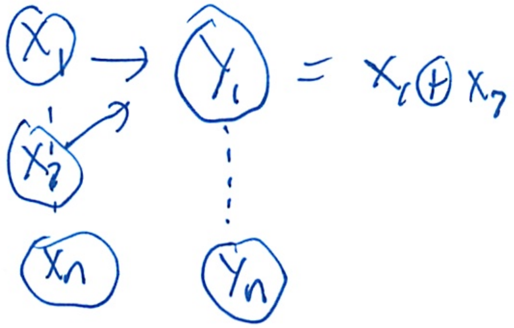
$P_1[n] = x[n] \oplus x[n-1] \oplus x[n-3] \oplus x[n-5]$

$P_2[n] = x[n] \oplus x[n-1] \oplus x[n-2] \oplus x[n-4] \oplus x[n-5]$

Digression

Gs found emperically by testing for fixed lenght of code

How to find emperically



Can put all sorts of edges in

Art is deciding which edges to put in

"Expanders"

Won't go into details how to find them

What is the rate of the two codes?

Scheme 1

(n, k, d)

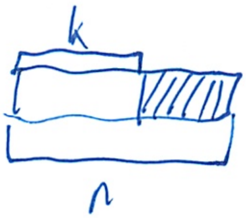
rate $= k/n$

$k = k$
$n = 2k$

rate $\frac{1}{2}$ since 2 bits transmitted for every message bit

③

Scheme 2



rate $= \frac{k}{n}$

$k = k$
$n = 2k$

$(2k, k, \quad)$

^ can find

find non zero codeword w/ min # of 1s
See last recitation

rate $= \frac{k}{2k} = \frac{1}{2}$

k = Constraint Length

The lenght of the polynomials

how much history are you involving

Scheme1 $G_1 = 4$ → since 0, 1, 2, 3
$G_2 = 4$

^ the constraint lenght for scheme max of any polynomial

max = 4

Scheme 2 $G_1 = 6$
$G_2 = 6$

$L_{max} = 6$

Which code leads to better performance (lower BER)

- not precise
- touchy feely

Scheme 2 has more history

- more error correction possibility

A good G w/ shorter contraint lenght is better than
bad G w/ longer " "

d) Someone says add G_3. New rate?

Rate $= \frac{1}{3}$

Constraint lenght $= 4$

Certainly not weaker

Adding more structure - so more error correction
At cost of rate/bandwith

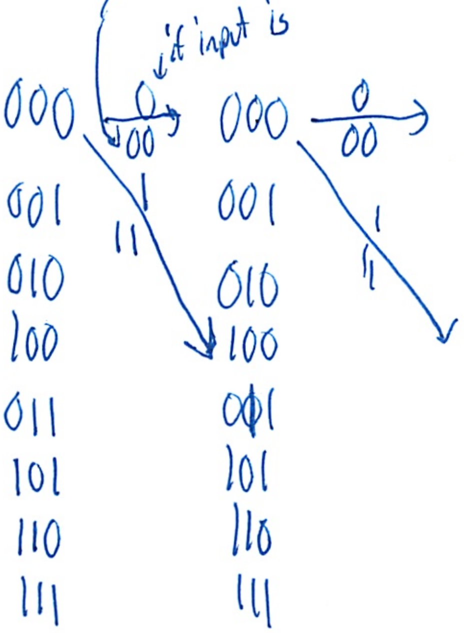
Decoding

↓

2^{k-1}

$x[n]+x[n-1]+x[n-3]$

Parity bits $x[n]+x[n-1]+x[n-2]$ for the state it is going towards



← OK transitions are same at each time

- - -

Verterbi - Dynamic Programming

On tree graph

Belief prediction

A different convolution code

$G_0 = 111$ $G_1 = 110$

Construct state diagram

$P_1[n] = x[n] \oplus x[n-1] + x[n-2]$

$P_2[n] = x[n] \oplus x[n-1]$

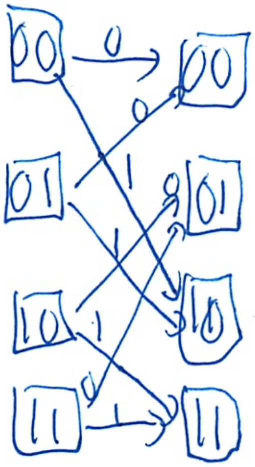
Rate = $\frac{1}{2}$

Constraint lenght = ~~[illegible]~~ = 3

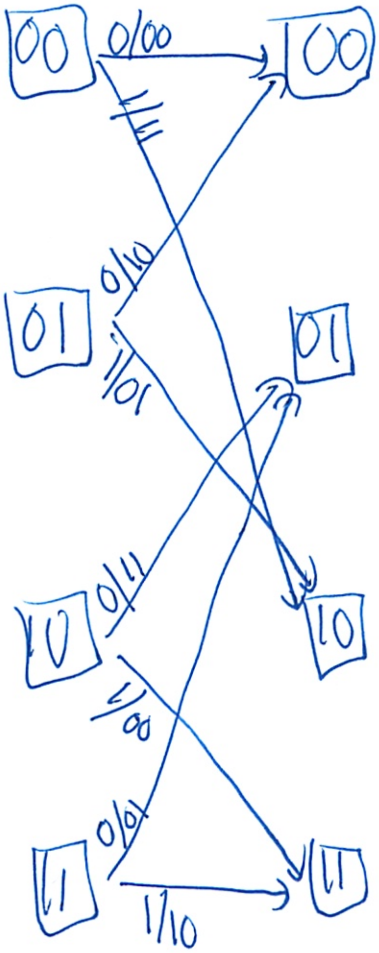
History = $2^{k-1} = 4$ states possible

6

Need 2 bits of history + current bit



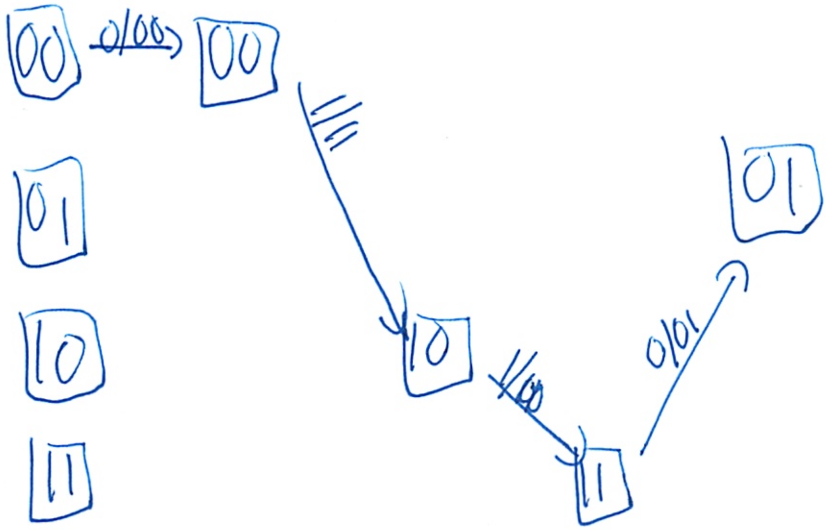
Rewrite better



7

Encode 0110 ← message

Remember start w/ 00 state



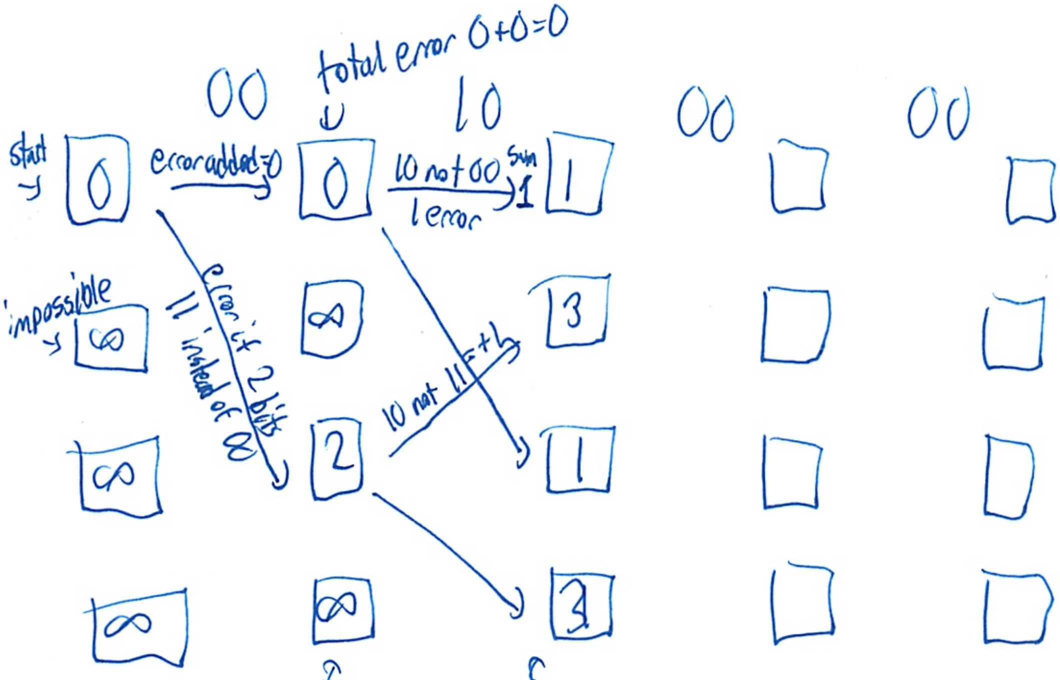
Tx	00	11	00	01
State	00	10	11	01

2 errors

wire/channel

00 10 00 00 reciever gets

Decode (Shortest path algorthm)



When multiple arcs coming in only take minimum

are really multiple arcs coming in from ∞ though rem. take min

Arcs tell you what bits were transmitted

Interview 4

3/11

St dev - not abs value

- otherwise only half dist
- not as large as normal since only half

Consider 3 x 3 or 4x4

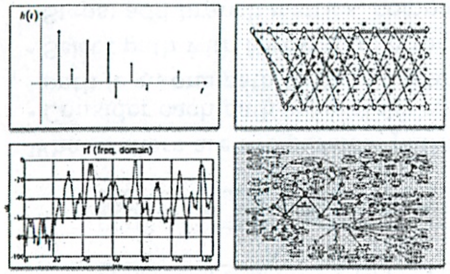
- can only fix 1 error
- hamming d = 3
- or SECC ?

- can extend out to mutiple errors

but harder to code

Lots of material

Classes not any more - 6.451 old videos



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

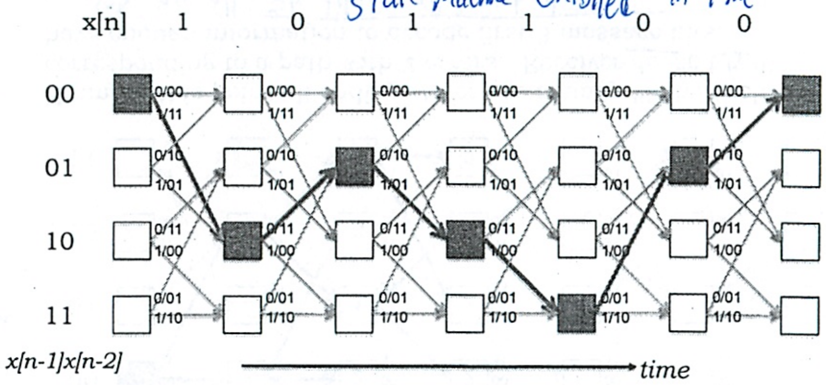
6.02 Spring 2011
Lecture #11

- state machines & trellises
- path and branch metrics
- Viterbi algorithm: add-compare-select
- hard decisions vs. soft decisions

Last time in 6.02...

Trellis View @ Transmitter

state machine unrolled in time



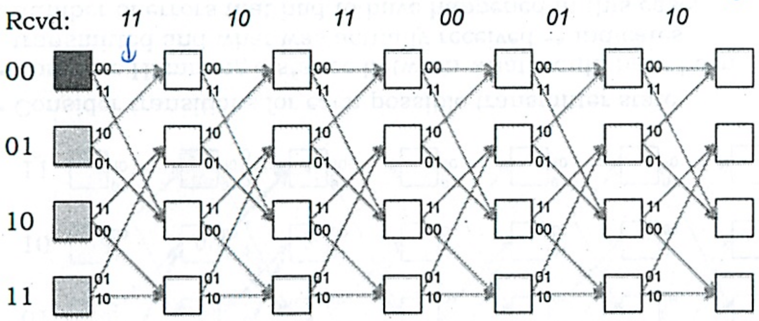
Send: 11 11 01 00 01 10

Current state ~~[illegible]~~ → ~~old~~ 2 recent bits

Reciever

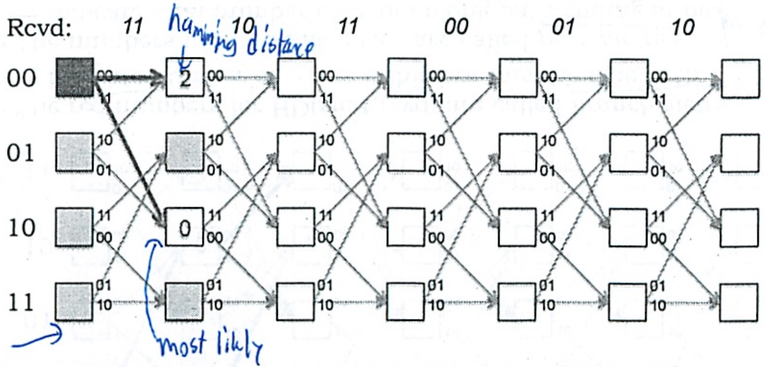
Finding the Most-likely Path

since some bits are wrong



Given the received parity bits, the receiver must find the most-likely sequence of transmitter states, i.e., the path through the trellis that minimizes the Hamming distance between the received parity bits and the parity bits the transmitter would have sent had it followed that state sequence.

Processing 1st pair of parity bits



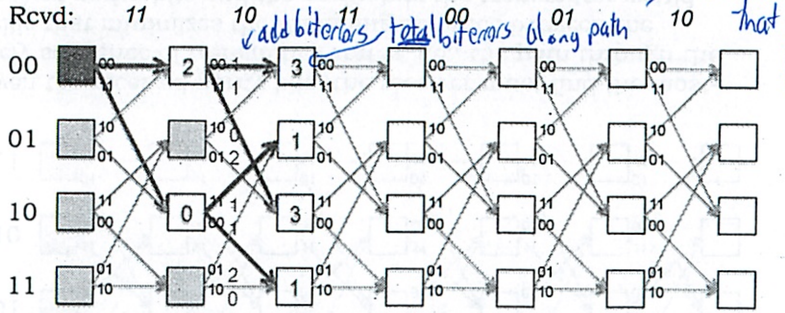
- Transmitter is known to be in state 00 at start of message.
- If next message bit 0, next state is 00, transmit 00
  - So receiving 11 means there have been two errors in the channel
- If next message bit is 1, next state is 10, transmit 11
  - So receiving 11 means there have been no errors in the channel

but can't fully be sure here

3/9

Processing 2nd pair of parity bits

the # of errors that must have been ocured to get to that state

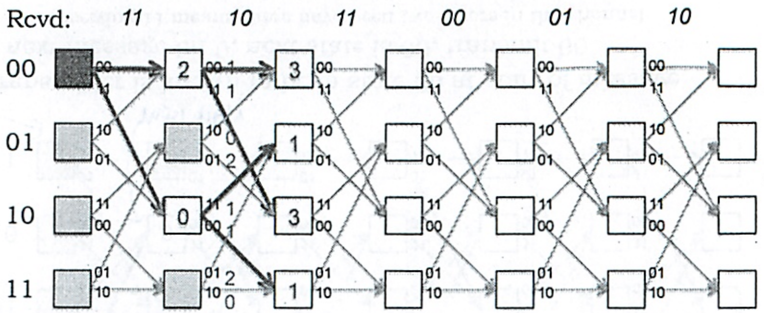


add bit errors total bit errors along path

- Consider transitions for each possible transmitter state
- Compute Hamming distance between what would have been transmitted and what was actually received → indicates number of errors that had to have happened in this case.
- Enter total errors along path in each destination state in next column of the trellis, color transition arc red

so far only 1 unique state to each path

Branch Metrics, Path Metrics

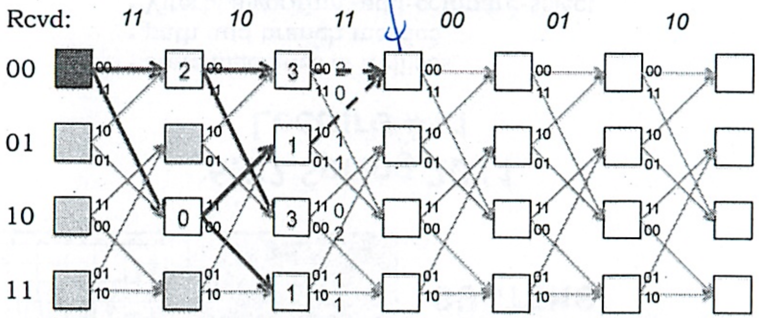


- The red numbers for HD(xmit,rcvd) are called *branch metrics* → indicate number of errors if this arc was the true path arrows
- The numbers in the trellis boxes are called *path metrics* → indicate total number of errors along path ending at box boxes
- The red arrows indicate sequence of transmitter states that end at a particular column and state.

say HD(00,01) = 1

2 states could have arrived from / 3+2 = 5 or 1+1 = 2 ← minimum/most likely so write in box remember that ~~[illegible]~~ arrow - disregard the other arrows

Processing 3rd pair of parity bits

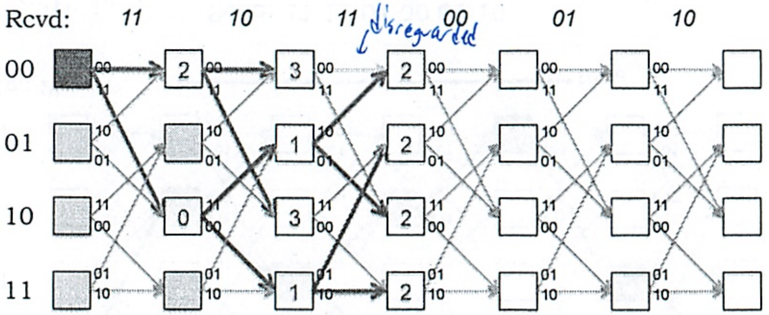


What if there are two possible paths to a particular state?

- Consider each path separately: compute total errors along each path (e.g., one path to 00 has 5 total errors, the other 2 errors)
- Select path with fewest total errors as the *most-likely path*
- Steps: add branch metrics, compare total errors, select path

add, compare, select

What Can We Tell About Message?



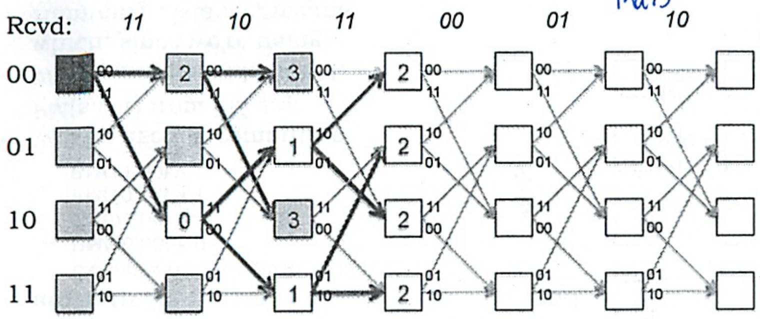
Hmm, at this point all ending states are equally likely, each corresponding to a path with 2 errors. Receiver doesn't (yet) have enough information to decode first 3 message bits.

why are all =ly likely - all have had 2 errors

Can receiver tell anything about the message?

Survivor Paths

Can start eliminating stuff thats not likley

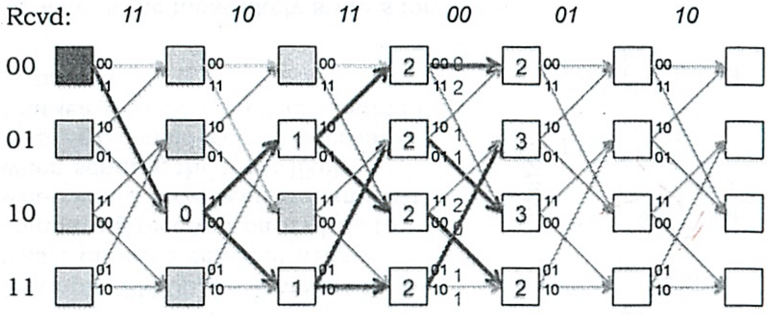


Color boxes grey and remove arrows (in next slide) that go to a higher than used/min state and anything before that if that is its exclusive arrow!

Receiver can make some deductions:

Some earlier states are no longer part of *any* most-likely path. We can eliminate partial paths leading to those states since they will not be part of the final most-likely path. Do this recursively…

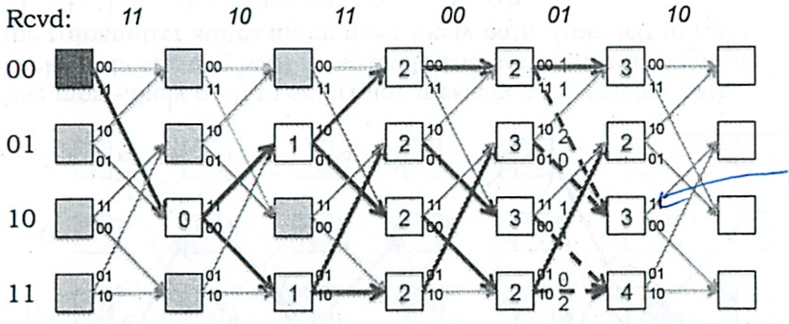
Processing 4th pair of parity bits



The usual: ↑ Starting to see message!

- Add branch metrics to previous total errors
- Compare paths arriving at destination state
- Select path with smallest total errors as most-likely path

Processing 5th pair of parity bits



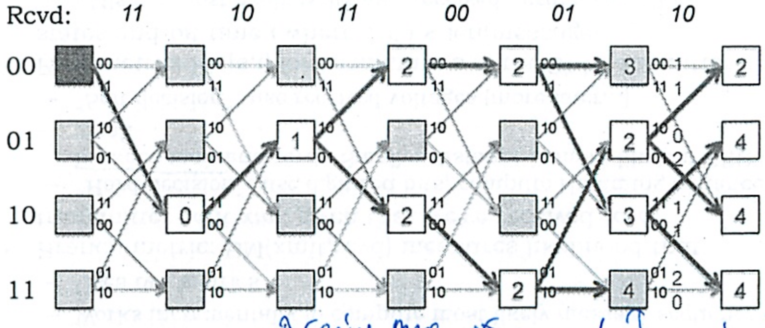
what if tie?

When there are "ties" (sum of metrics are the same) ↑ add, compare, select

- Make an arbitrary choice about incoming path
- If state is not on most-likely path: choice doesn't matter
- If state is on most-likely path: choice may matter and error correction has failed

pick one randomly ~guessing

Processing 6th pair of parity bits



← most likely final state

← follow its path backwards!

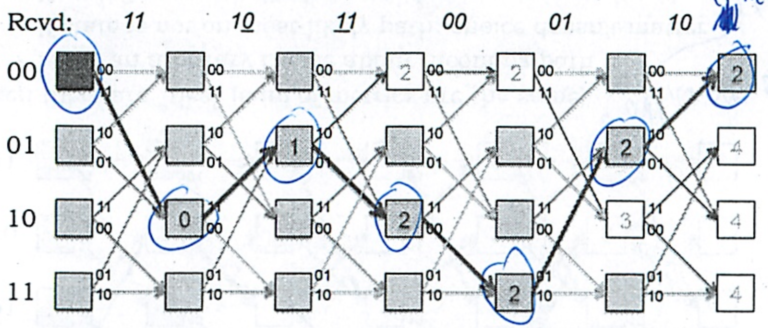
↑ Seeing more of message! ↑ gray since no sucessor state

When we reach end of received parity bits:

- Each state's path metric indicates how many errors have happened on most-likely path to state
- Most-likely final state has smallest path metric
- Ties mean end of message uncertain (but survivor paths may merge to a unique path earlier in message)

Traceback

○ = shaded red (others shaded grey)



← So now we see message

Use most-likely path to determine message bits, tracing path backwards starting with most-likely end state. In this example, the transmitter states along most-likely path, from left to right:

Write down states 10 01 10 11 01 00

Message bits from high-order state bit:

101100

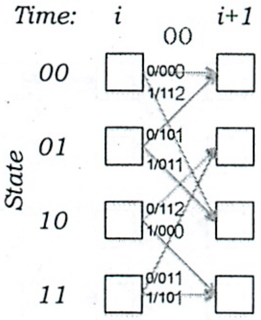
Viterbi Algorithm

- Want: Most likely message sequence
- Have: (possibly corrupted) received parity sequences
- Viterbi algorithm for a given k and r:
  - Works incrementally to compute most likely message sequence
  - Uses two metrics
- Branch metric: BM(xmit,rcvd) measures likelihood that transmitter sent *xmit* given that we've received *rcvd.*
  - "Hard decision": use digitized bits, compute Hamming distance between xmit and rcvd. Smaller distance is more likely if BER is small
  - "Soft decision": use received voltages (more later…)
- Path metric: PM[s,i] for each state *s* of the 2^{k-1} transmitter states and bit time *i* where $0 \le i < \text{len(message)}$.
  - PM[s,i] = most likely BM(xmit_m,received parity) over all message sequences *m* that leave transmitter in state *s* at time *i*
  - PM[s,i+1] computed from PM[s,i] and $p_0[i],\ldots,p_{r-1}[i]$

Hard-decision Branch Metric

Actual algorithm written out tediously

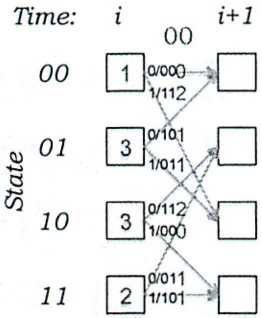
- BM = Hamming distance between expected parity bits and received parity bits
- Compute BM for each transition arc in trellis
  - Example: received parity = 00
  - BM(00,00) = 0
    BM(01,00) = 1
    BM(10,00) = 1
    BM(11,00) = 2
- Will be used in computing PM[s,i+1] from PM[s,i].
- We'll want most likely BM, which, since we're using Hamming distance, means minimum BM.



Computing PM[s,i+1]

Starting point: we've computed PM[s,i], shown graphically as label in trellis box for each state at time *i.*

Example: PM[00,i] = 1 means there was 1 bit error detected when comparing received parity bits to what would have been transmitted when sending the most likely message, considering all messages that leave the transmitter in state 00 at time i.



Q: What's the most likely state *s* for the transmitter at time *i*?

A: state 00 (smallest PM[s,i])

Computing PM[s,i+1] cont'd.

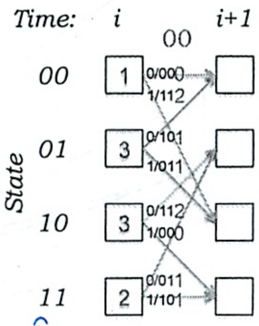
Q: If the transmitter is in state s at time i+1, what state(s) could it have been in at time i?

A: For each state s, there are two predecessor states α and β in the trellis diagram

Example: for state 01, α=10 and β=11.

Any message sequence that leaves the transmitter in state s at time i+1 must have left the transmitter in state α or state β at time i.

d



4 diff possible last 2 bits (constraint level of 3)

k=4 8 boxes

twice work & grows exponentially w/ constraint lenght

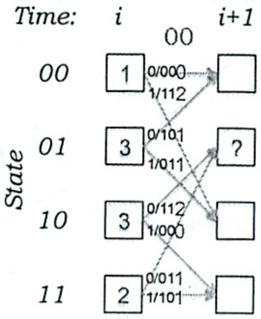
k=15 2^{14} boxes

Computing PM[s,i+1] cont'd.

Example cont'd: to arrive in state 01 at time i+1, either

1) The transmitter was in state 10 at time i and the i^th message bit was a 0. If that's the case, the transmitter sent 11 as the parity bits and there were 2 bit errors since we received 00. Total bit errors = PM[10,i] + 2 = 5 *OR*
2) The transmitter was in state 11 at time i and the i^th message bit was a 0. If that's the case, the transmitter sent 01 as the parity bits and there was 1 bit error since we received 00. Total bit errors = PM[11,i] + 1 = 3

Which is mostly likely?



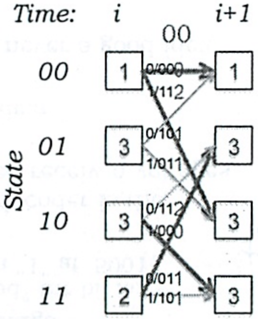
Computing PM[s,i+1] cont'd.

Formalizing the computation:

PM[s,i+1] = min(PM[α,i] + BM[α→s], PM[β,i] + BM[β→s])

Example:

PM[01,i+1] = min(PM[10,i] + 2, PM[11,i] + 1) = min(3+ 2,2+1) = 3



Notes:

1) Remember with arc was min; saved arcs will form a path through trellis
2) If both arcs have same sum, break tie arbitrarily (e.g., when computing PM[11,i+1])

Viterbi Algorithm Summary

- Branch metrics measure the likelihood by comparing received parity bits to possible transmitted parity bits computed from possible messages.
- Path metric PM[s,i] measures the likelihood of the transmitter being in state s at time i assuming the mostly likely message of length i that leaves the transmitter in state s.
- Most likely message? The one that produces the most likely PM[s,N].
- At any given time there are 2^{k-1} most-likely messages we're tracking → time complexity of algorithm grows exponentially with constraint length k.

Hard Decisions

Hamming distance

- As we receive each bit it's immediately digitized to "0" or "1" by comparing it against a threshold voltage
  - We lose the information about how "good" the bit is: a "1" at .9999V is treated the same as a "1" at .5001V
- The branch metric used in the Viterbi decoder is the Hamming distance between the digitized received voltages and the expected parity bits
  - This is called *hard-decision Viterbi decoding*
- Throwing away information is (almost) never a good idea when making decisions
  - Can we come up with a better branch metric that uses more information about the received voltages?

throws away info vs

V_{th} 31

30

looks closer at Voltage

.999

.50001

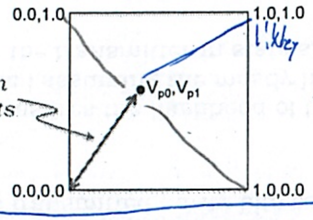
both "1" but difference

Soft Decisions

- Let's limit the received voltage range to [0.0,1.0]
  - $V_{eff} = \max(0.0, \min(1.0, V_{received}))$
  - Voltages outside this range are "good" 0's or 1's

say 1.3 is clearly a 1

- Define our "soft" branch metric as the square of the Euclidian distance between received V_{eff} and expected voltages



"Soft" metric when expected parity bits are 0,0

See §9.3 In Notes

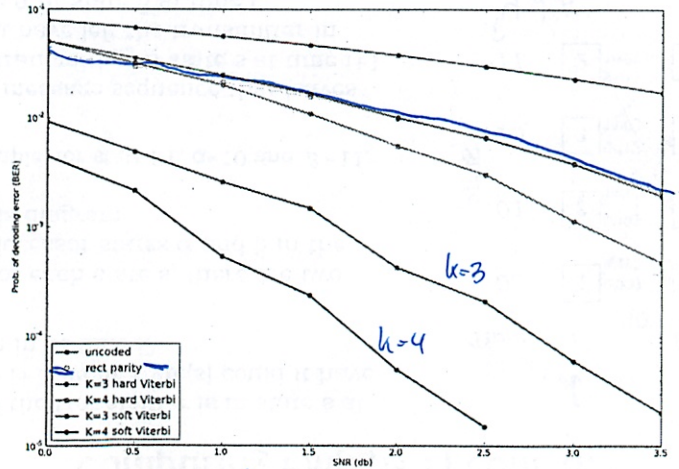
$(V_{p0}-0)^2+(V_{p1}-0)^2$

no sq root!

- Soft-decision decoder chooses path that minimizes sum of the squares of the Euclidian distances between received and expected voltages
  - Different branch metric but otherwise the same recipe

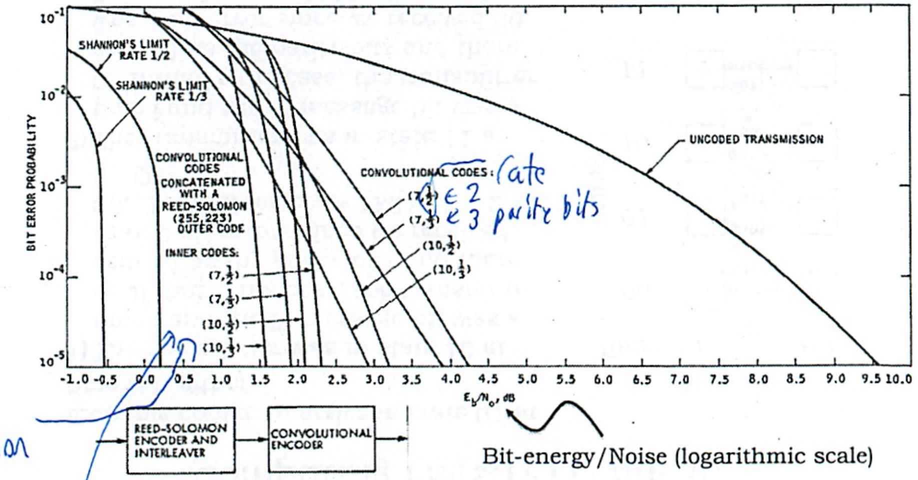
Adding the folding pt # not hamming distance just a few lines different

More Work, Better BER



k = # of places using a bit

Code performance



Bit-energy/Noise (logarithmic scale)

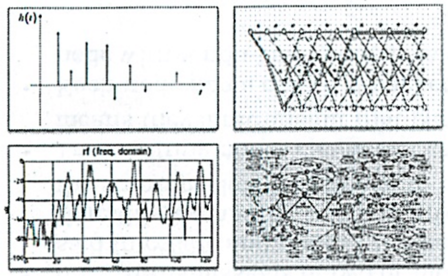
new generation here

turbo codes

Source: Butman, Deutsch, Miller, "Performance of Concatenated Codes for Deep Space Missions"

Conv. bad at burst errors

How to do sharing of channel



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

Ethernet was Originally shared

Now mostly point to point

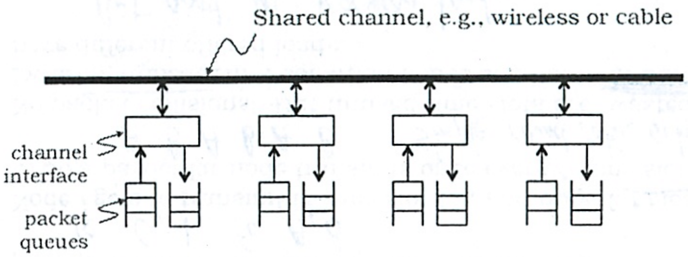
Wireless is shared

6.02 Spring 2011
Lecture #12

- shared medium → media access protocol
- time division multiple access (TDMA)
- contention protocols, Alohanet

time sharing

Shared Communications Channels



- Basic constraint: avoid collisions between transmitters
  - Collisions can be detected from corrupted packets
- Wanted: a communications protocol ("rules of engagement") that ensures good performance overall

what qualifies as good?

"Good" Performance

- High utilization
  - Channel capacity is a limited resource → use it efficiently
  - Ideal: use 100% of channel capacity in transmitting packets
  - Waste: idle periods, collisions, protocol overhead
- Fairness
  - Divide capacity equally among requesters
  - But not every node is requesting all the time...
- Bounded wait
  - An upper bound on the wait before successful transmission
  - Important for isochronous communications (e.g., voice/video)
- Scalability
  - Accommodate changing number of nodes, hopefully without changing implementation of any given node

don't want overhead of "raise your hand to transmit"

everyone can talk

can't just divide evenly

Sharing Protocols

- Protocols ≡ "rules of engagement" for good performance
  - Known as media access control (MAC) or multiple access control
- Time division (multiple
  - Share time "slots" between requesters
  - Prearranged: time division multiple access (TDMA) (round robin
  - Not prearranged: contention protocols (e.g., Alohanet). These are interesting because each node operates independently
- Frequency division
  - Give each transmitter its own frequency, receivers choose "station"
  - Our topic for after spring break
- Code division
  - Uses unique orthogonal pseudorandom code for each transmitter
  - Channel adds transmissions to create combined signal
  - Receiver listens to one "dimension" of combined signal using dot product of code with combined signal.

Someone x,y,z ~~channel~~ dimension
Seperate out w/ dot product

3/14

Utilization

- Utilization measures the Useful throughput of a channel:

$$U_{channel} = \frac{\text{total throughput over all nodes}}{\text{maximum data rate of channel}}$$

- Example: 10 Mbps channel, four nodes get throughputs of 1, 2, 2 and 3 Mbps. So utilization is (1+2+2+3)/10 = 0.8.
- $0 \le U \le 1$. Utilization can be less than 1 if
  - The nodes have packets to transmit (nodes with packets in their transmit queues are termed *backlogged*), but the protocol is inefficient. ← have stuff to send if turn
  - There is insufficient *offered load*, i.e., there aren't enough packets to transmit to use the full capacity of the channel. ← usually we assume people could make full use of channel
- With backlogged nodes, perfect utilization is easy: just let one node transmit all the time! But that wouldn't be fair…

Fairness

- Many plausible definitions. A standard recipe:
  - Measure throughput of nodes = x_i, over a given time interval
  - Say that a distribution with lower standard deviation is "fairer" than a distribution with higher standard deviation.
  - Given number of nodes, N, fairness F is defined as

$$\text{index} \quad F = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N\sum_{i=1}^{N} x_i^2}$$

- $1/N \le F \le 1$, where $F=1/N$ implies single node gets all the throughput and $F=1$ implies perfect fairness.
- We'll see that there is often a tradeoff between fairness and utilization, i.e., fairness mechanisms often impose some overhead, reducing utilization.

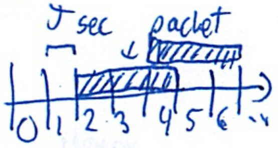
if 3, 3, 3, 3

$$\text{then} = \frac{12^2}{4\cdot(9+9+9+9)} = 1$$

if 3, 0, 0, 0

$$\frac{9}{4\cdot 9} \to \frac{1}{4}$$

Abstraction for Shared Medium



- Time is divided into *slots* of equal length
- Each node can start transmission of a packet only at the beginning of a time slot

Can only start at a start of time slot

- All packets are of the same size and hence take the same amount of time to transmit, equal to some integral multiple of time slots.

and packet is multiple of T

- If the transmissions of two or more nodes overlap, they are said to *collide* and none of the packets are received correctly. Note that even if the collision involves only part of the packet, the entire packet is assumed to be lost.

Overlap - so both packets failed

- Transmitting nodes can detect collisions, which usually means they'll retransmit that packet at some later time.

- Nodes know they failed, retransmit

- Each node has a queue of packets awaiting transmission. A node with a non-empty queue is said to be *backlogged*.

Time Division Multiple Access (TDMA)

- Suppose that there is a centralized resource allocator and a way to ensure time synchronization between the nodes – for example, a cellular base station.

  gives it the time points

- For N nodes, give each node a unique index in the range [0,N-1]. Assume each slot is numbered starting at 0.

  ie 0, 1 for A, B

- Node i gets to transmit in time slot t if, and only if, $t \bmod N = i$. So a particular node transmits once every N time slots.

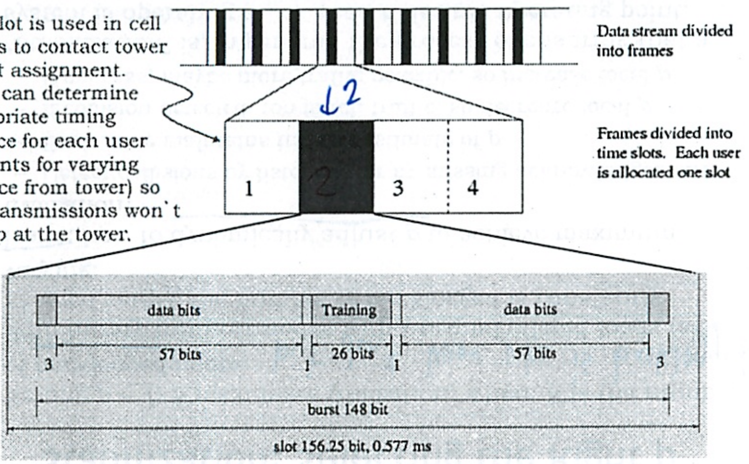
  A B A B A B - Simple round robin scheme

- No packet collisions! But unused time slots are "wasted", lowering utilization. Poor when nodes send data in bursts or have different offered loads.

  Not good on varying loads

TDMA for GSM Phones

First slot is used in cell phones to contact tower for slot assignment. Tower can determine appropriate timing advance for each user (accounts for varying distance from tower) so that transmissions won't overlap at the tower.



Data stream divided into frames
Frames divided into time slots. Each user is allocated one slot
1
2
3
4
2
data bits
Training
data bits
3
57 bits
1
26 bits
1
57 bits
3
burst 148 bit
slot 156.25 bit, 0.577 ms

http://en.wikipedia.org/wiki/Time_division_multiple_access

1st slot = RATCH – request to tower when to translate

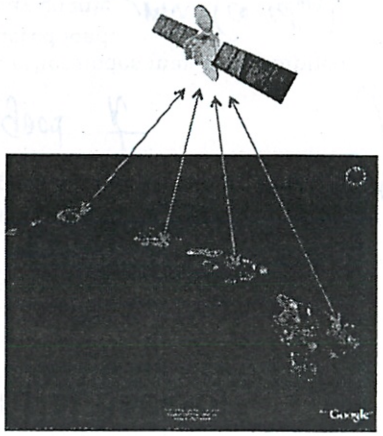
tower gives you back timing instructions inc allowance for time of flight; a few bits left if moving

Contention Procotols: Alohanet

To improve performance when there are burst data patterns or skewed loads, use a contention protocol where allocation is not predetermined.

Alohanet was a satellite-based data network connecting computers on the Hawaiian islands. One frequency was used to send data to the satellite, which rebroadcast it on a different frequency to be received by all stations.

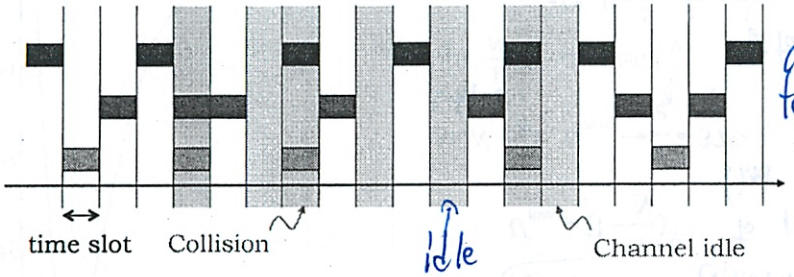
Stations could only "hear" the satellite, so had to decide independently when it was their turn to transmit.



uplink freq to sat
downlink freq to everyone
But when to transmit? Did not want sat to decide

Success, Idleness, Collisions

Slotted everything happens in slots



a transmission takes 1 timeslot

- Throughput = *Uncollided* packets per time interval
- Utilization = Throughput / Channel Rate = 12/20 = .6

actually quite high usually ~.4

* tranmitters can detect collisions

Sucessful transmission

Slotted Aloha

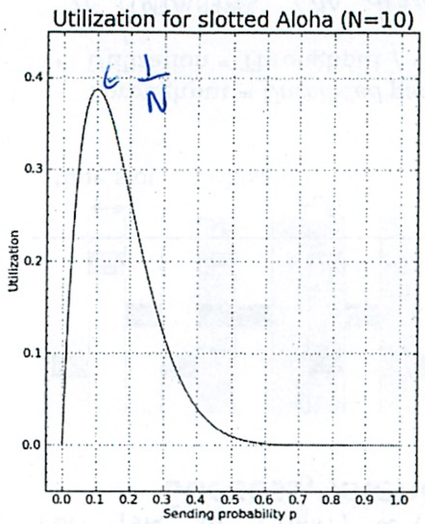
- Aloha protocol followed by each of N nodes:

  if a node is backlogged, it sends a packet in the next time slot with probability p. b/w 0 and 1 random.random() < p

- Assume (for now) each packet takes exactly one time slot to transmit (*slotted* Aloha)
- Utilization when all nodes backlogged? The probability that exactly one node sends a packet.
  - prob(send a packet) = p
  - prob(don't send a packet) = $1-p$
  - prob(only one sender) = $p(1-p)^{N-1}$
  - There are *(N choose 1)* = N ways to choose the one sender

$$\boxed{U_{\text{slotted Aloha}} = Np(1-p)^{N-1}}$$

Maximizing Utilization



To determine maximum:
set $dU/dp = 0$, solve for p.

Result: $p = 1/N$, so differentiate w/ resp to p to find max

$$U_{\max} = (1-\frac{1}{N})^{N-1}$$

As $N \rightarrow \infty$, $U_{\max} \rightarrow \frac{1}{e} \approx 37\%$

why e?

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

✓ taylor series expansion

$$= (N-1)(-\frac{1}{N} - \frac{1}{2N^2} - \frac{1}{3N^3} - \ldots)$$

$$= -1 + \frac{1}{N} + \frac{1}{6N^2} + \frac{1}{12N^3} + \ldots$$

$$= -1 \text{ as } N \rightarrow \infty$$

Should remember in this class

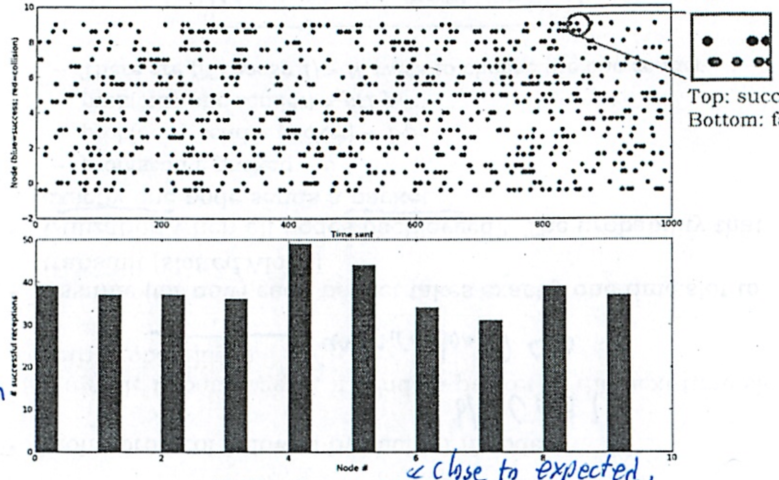
$$U_{\max} = (1-\frac{1}{10})^9 = .9^9$$

? what is this - ? mathematically way to take limit ?

Can you do 37% utilization?

→ It's very simple

Simulation of Slotted Aloha (N=10)



Top: success
Bottom: failure

← close to expected

Utilization = .38, Fairness = .98

```
python PS6_2.py -r -n 10 -p .1 -t 1000
```

Stabilization: Selecting the Right p

- Setting $p = 1/N$ maximizes utilization, where N is the number of *backlogged* nodes. but don't know who is backlogged
- With bursty traffic or nodes with unequal offered loads (aka *skewed loads*), the number of backlogged is constantly varying.
- Issue: how to dynamically adjust p to achieve maximum utilization?
  - Detect collisions by listening, or by missing acknowledgement
  - Each node maintains its own estimate of p
  - If collision detected, too much traffic, so decrease local p
  - If success, maybe more traffic possible, so increase local p
- "Stabilization" is, in general, the process of ensuring that a system is operating at, or near, a desired operating point.
  - Stabilizing Aloha: finding a p that maximizes utilization as loading changes.

simple strategy

Binary Exponential Back-off

- Decreasing p on collision
  - Estimate of N (# of backlogged nodes) too low, p too high
  - To quickly find correct value use ~~multiplicative decrease~~: halving p
    $p \leftarrow p/2$
  - k collisions in a row: p decreased by factor of 2^{-k}
  - Binary: 2, exponential: k, back-off: smaller p → more time between tries

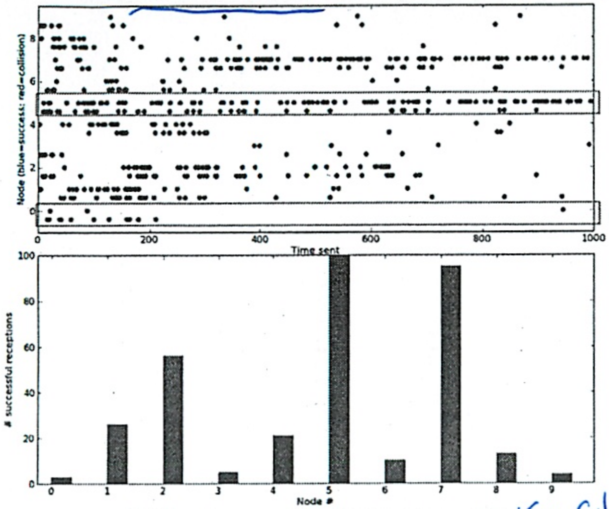
reduce p till good N

switch b/w

- Increasing p on success
  - While we were waiting to send, other nodes may have emptied their queues, reducing their offered load.
  - If increase is too small, slots may go idle
  - Try multiplicative increase: $p \leftarrow$ ~~max~~ min $(2*p, 1)$ (correction)
  - Or maybe just: $p \leftarrow 1$ to ensure no slots go idle

Exponential - if k things in a row p is changing exponentially over time

Simulation of Stabilized Aloha



Some nodes did well

Others didn't

Starvation node

Utilization = .33, Fairness = .47 ← awful

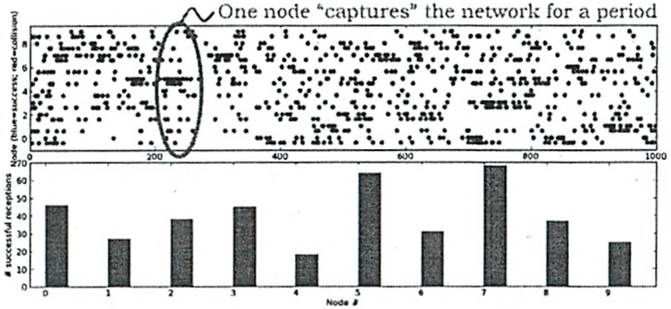
```
python PS6_3.py -r -n 10 -t 1000
```

What Went Wrong?

- Starvation
  - Too many successive failures → p very small → no xmit attempts
  - Result: significant long-term unfairness
  - Try a reduction rule with a lower bound: $p \leftarrow max(p_{min}, p/2)$
  - Choosing p_{min} << $1/max(N)$ seems to work best

starves other node

add lower band



One node "captures" the network for a period

Utilization = .4, Fairness = .87

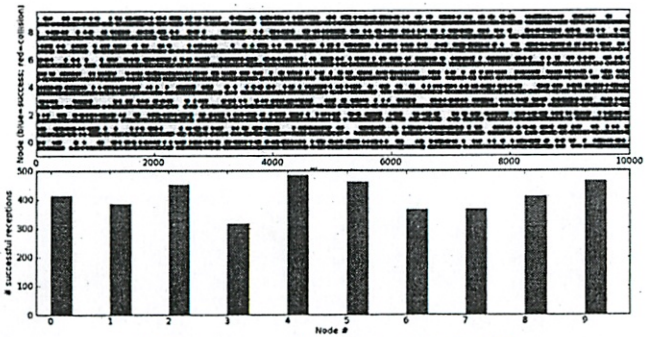
```
python PS6_3.py -r -n 10 -t 1000 --pmin=.05
```

better

Limiting the Capture Effect

- Capture effect
  - A successful node maintains a high p (avg. near 1)
  - Starves out other nodes for short periods
  - Try an increase rule with an upper bound: $p \leftarrow min(p_{max}, 2*p)$

So match w/ upper bound



Utilization = .41, Fairness = .99

```
python PS6_3.py -r -n 10 -t 1000 --pmin=.05 --pmax=0.8
```

6.02 Recitation

3/15

Media Access Control (MAC)

Machines interested in total throughput

Not just their throughput

So backs off so others can talk

IEEE 802.11xx standards

Prof. Shah's topic

Want

- simple - utalize only local info, little computation
- efficent - try to always make useful transmissions
- fair - how to judge that?
  - "proportional fair
  - max-min fair "Socialist"
  - "capitalist"

"Cake Cutting Algorithms"

$T \sim o(T)$

$$\lim_{T \to \infty} \frac{g(T)}{T} \to 0$$

Example $\sqrt{T} \to \frac{T-\sqrt{T}}{T} = 1 - \frac{1}{\sqrt{T}}$

loss should be much smaller than total transmission

Cake Cutting

For $n=2$

- person A cuts
  B chooses piece

$n=3$

Move knife L→R at ∞ly slow rate

Anyone can stop the knife by saying "I want it"

2

Read book "Cake cutting algorthms"
"Stable mariage algorthms"

Aloha algorthm

$p(T)$ local information only
- each node on own

$p(0) = 1$

$$p(t+1) = \begin{cases} p(t)/2 & \text{if raised at } T + \text{ failed since conflict} \\ p(t) & \text{Didn't raise hand} \\ 1 & \text{Raised hand + sucessful reset } p \text{ to } 1 \end{cases}$$

- greedy
- monopolizes chanel?
- does not do well when channel becomes congested

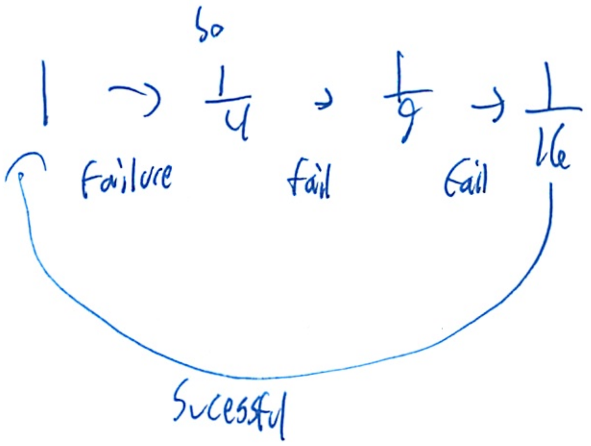
So then Polynomial backoff

$p(k) \approx 2^{-k}$ k = # of unsucessful attempts

$\to \frac{1}{(k+1)^{\alpha}}$ $\alpha > 1$

always works

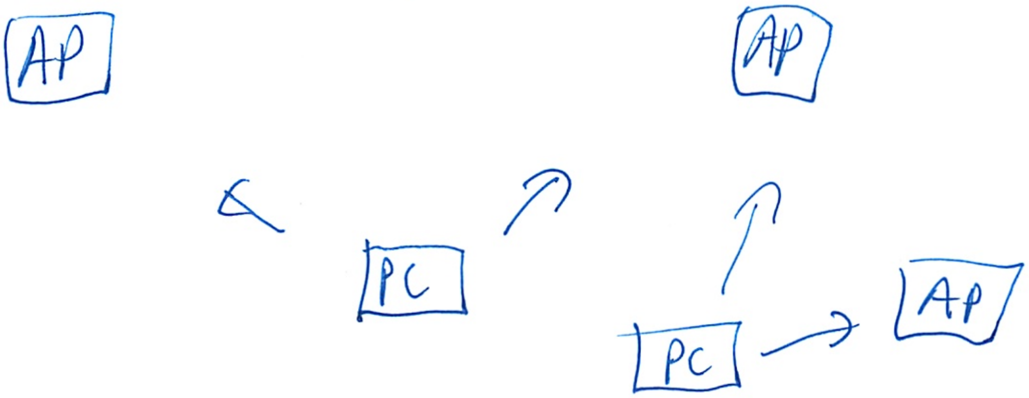
3



p of transmitting next round $\alpha = 2$

Doesn't work whenever one source everyone is transmitting

Have bunch of APs to connect to



But if each PC talks to diff AP - you are actually talking to both
- interfearence
- talking loudly allows other people to hear

#2 Binary Exponential Backoff

Will it ensure 2 nodes will never collide

False

One person A 2^{k_1}, collide, so $2^{-(k_1+1)}$

B 2^{-k_2} $2^{-(k_2+1)}$

↑ at each time step, p of trans.

(4)

$$P(\text{collision}) = 2^{-(k_1+1)} \cdot 2^{-(k_2+1)}$$

Which means still small chance of collision

Polynomial - since it gets smaller we may have collisions - but it will eventually work

Since $P() \underset{\substack{\cancel{\text{Vm}} \\ \text{tends}}}{\to} 0$

b) Correct (did not here)

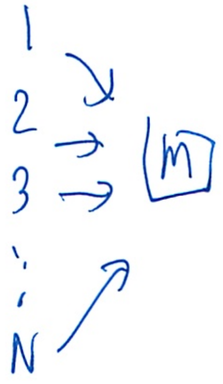
c) Skip

d) Over timescale it improves throughput vs not using ______

No - since starvation can happen

(Need to see on paper)

Why did e appear?



each transmits at every time step w/ P( )

$p = \frac{1}{N}$ is best case (From simulations)

$P_{\text{sucess}} = \left(1-\frac{1}{N}\right)^{N-1}$

└ P(Node 1 tran sucess) + P(Node 2 trans sucess) + ... + P(Node N trans sucess)

call Synetic so = N · P(Node 1 sucess trans)

5

For P(Node 1 trans sucess) = P(Node 1 trans) AND P(No one else trans)

$$\cdot \frac{1}{N}\left(1-\frac{1}{N}\right)^{N-1}$$

So in total

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

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

~~Clear~~ Seq decreases to $\frac{1}{e} \approx .3762 \ldots$

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

Pi comes from circle

e comes from i

3/13

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 PS5

Dates & Deadlines

issued:	Mar-09-2011 at 00:00
due:	Mar-17-2011 at 06:00
checkoff due:	Mar-22-2011 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.

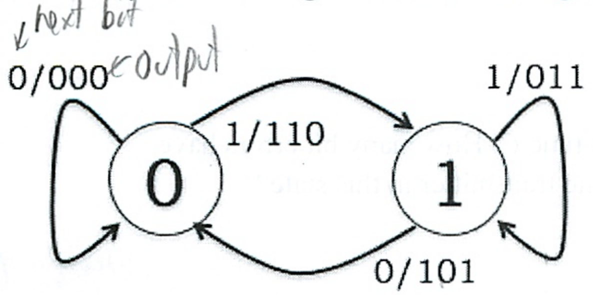
Problem 1.

Consider a convolutional code with three generator polynomials

```
p0[n] = (x[n] + x[n-1]) mod 2
p1[n] = x[n]
p2[n] = x[n-1]
```

Recitation 3/10

as shown by the following state transition diagram:



A. What are the constraint length (K) and code rate (r) for this code?

(n,k,d) but its n=0 k=3 (0 | 3

r = rate = $\frac{k}{n}$ n = k r = $\frac{1}{3}$ 3 bits trans. for every message bit

k = non zero codewords w/ min # of 1s

(points: 0.5) = # possible code bits to transmit 2^k

Oh

k

B. What sequence of bits would be transmitted for the message "0110"? Assume the transmitter starts in the state labeled "0".

perhaps 2 since that is how many words in past used - why all this conflicting info k = 2 - how much history including n

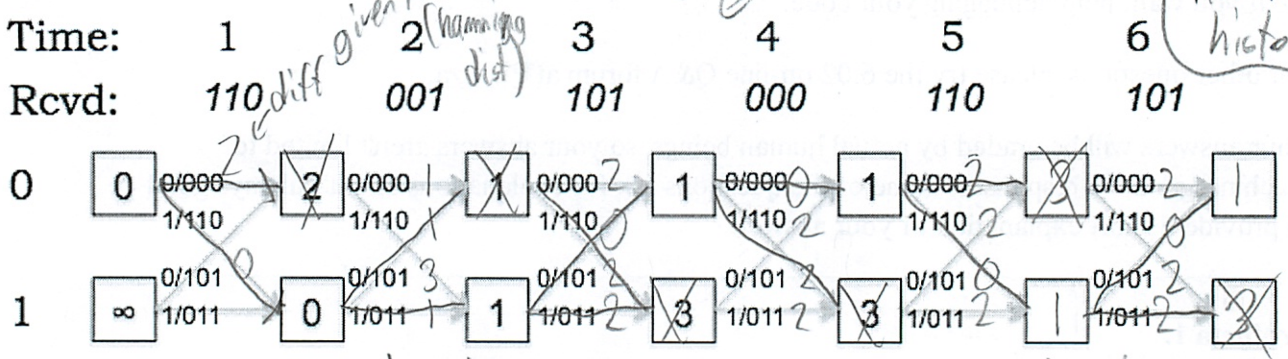
~~Try SM~~

000 110 011 101

(points: 0.5) ; how to decode again - the deconvolver

The Viterbi algorithm is used to decode a sequence of received parity bits, as shown in the following trellis diagram:

Oh right this



The boxes - add up

; pick min incoming sum

C. Please determine the path metrics for the last two columns of the trellis.

3 1
1 3

(points: 0.5)

D. What is the most-likely state of the transmitter after time 6? How many bit errors have been detected for the most likely message that left the transmitter in this state?

State 0
1 possible error

; should try crossing at unneeded

(points: 0.5)

E. What is the most-likely message sequence as decoded using the trellis above?

how exactly do you ~~@~~ nock off - just following 0 error path is that right? - or want to go backward

01 10010

(points: 0.5)

F. Given the answer to part E, at what time during the transmission did the first bit error occur?

2 The only

(points: 0.5)

Draw arrows darker that are used in the min test

(ok this is the 0error-following path) - wrote in

Problem 2.

A technique called **puncturing** can be used to obtain a convolutional code with a rate higher than 1/r, when r generators are used. The idea is to produce the r parity streams as before and then carefully omit sending some of the parity bits that are produced on one or more of the parity streams. One can express which bits are sent and which are not on each stream as a vector of 1's and 0's: for example, on a given stream, if the normal convolutional encoder produces 4 bits on a given parity stream, the vector (1 1 0 1) says that the first, second, and fourth of these bits are sent, but not the third, in the punctured code.

L that particular bit

Punctured codes are useful in communication systems that adjust their code rate in response to the observed BER of the channel (in packet systems, this would be measured by the number of packets with CRC failures). If the BER is low, using a punctured code may still yield an acceptable BER after decoding, while enjoying the increased throughput of a higher code rate.

how doesn't communicate puncturing decisions?

(same as previous

A. Suppose we start with a rate 1/2 convolutional code without puncturing. The encoder then uses the vector (1 0) on the first parity stream and the vector (1 1) on the second one: that is, it sends every other parity bit produced on the first parity stream and every parity bit produced on the second one. What is the rate of the resulting convolutional code?

So diff code for each generator/stream

says for every time step

$r = \frac{k}{n}$ What is new k? guessing k=k, n=2k for $\frac{1}{2}$

$\frac{1}{2}$ means take 2x as many bits to send same message

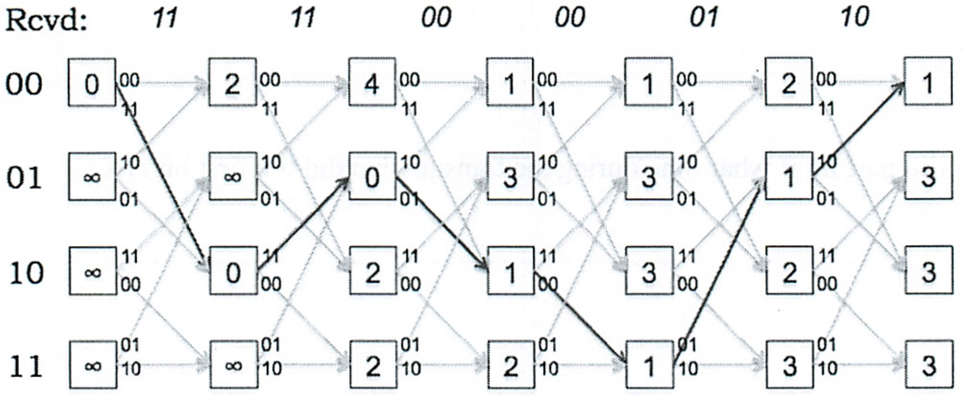
So now $\frac{1}{\frac{1}{2}\cdot 2 + \frac{1}{2}\cdot 1} = \boxed{\frac{2}{3}}$

half the

(points: 0.5)

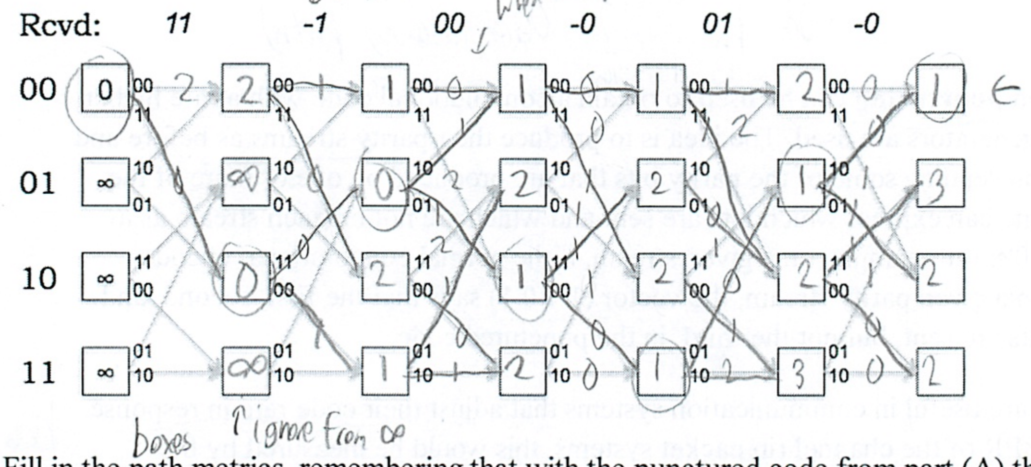
B. When using a punctured code, missing parity bits don't participate in the calculation of branch metrics. Consider the following trellis from a transmission using k=3, r=1/2 convolutional code that experiences a single bit error:

3 history used



Here's the trellis for the same transmission using the punctured code from part (A) that experiences the same error:

So just blank, when both, draw both arrows



boxes ↑ ignore from ∞

Fill in the path metrics, remembering that with the punctured code from part (A) that the branch metrics are calculated as follows: So just ignore that position in calculating

```
BM(-0,00)=0  BM(-0,01)=1  BM(-0,10)=0  BM(-0,11)=1
BM(-1,00)=1  BM(-1,01)=0  BM(-1,10)=1  BM(-1,11)=0
```

where the missing incoming parity bit has been marked with a "-". Use a text representation of the matrix (a 2-D array of numbers) when typing in the answer.

So can just draw w/o eliminating and trace back later

See above

(points: 1)

C. Did using the punctured code result in a different decoding of the message?

no

(points: 0.5)

Python Task 1: Viterbi decoder for convolutional codes -- Hard decision decoding

Useful download links:

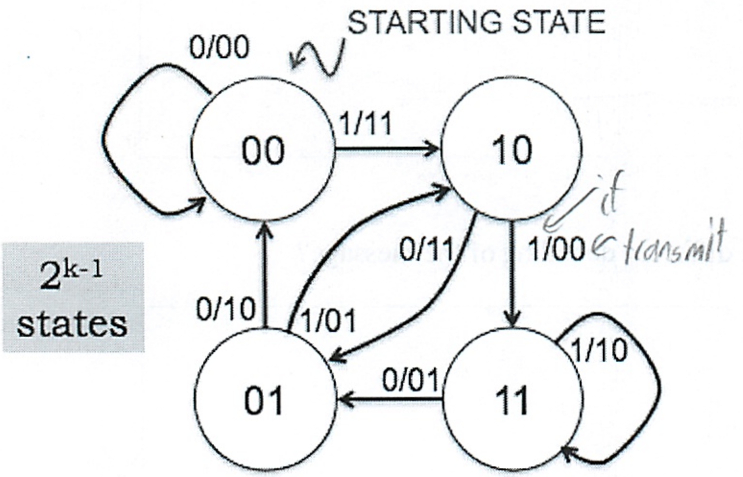
PS5_tests.py -- test jigs for this assignment
PS5_1.py -- template file for this task

As explained in lecture, a convolutional encoder is characterized by two parameters: a constraint length `K` and a set of `r` generator functions {`G0, G1,` ...}. The encoder processes the message one bit at a time, generating a set of `r` parity bits {`p0, p1,` ...} by applying the generator functions to the current message bit, `x[n]`, and `K-1` of the previous message bits, `x[n-1], x[n-2], ..., x[n-(K-1)]`. The `r` parity bits are then transmitted and the encoder moves on to the next message bit.

The operation of the encoder may be described as a state machine, as you know (we hope). The figure below is the state transition diagram for a rate 1/2 encoder with `K`=3 using the following two generator functions:

```
G0: 111, i.e., p0[n] = (1*x[n] + 1*x[n-1] + 1*x[n-2]) mod 2
G1: 110, i.e., p1[n] = (1*x[n] + 1*x[n-1] + 0*x[n-2]) mod 2
```

A generator function can described compactly by simply listing its `K` coefficients as a `K`-bit binary sequence, or even more compactly (for a human reader) if we construe the `K`-bit sequence as an integer, e.g., `G0: 7` and `G1: 6`. We will use the latter (integer) representation in this task.



which are parity bits

In this diagram the states are labeled with the two previous message bits, `x[n-1]` and `x[n-2]`, in left-to-right order. The arcs -- representing transitions between states as the encoder completes the processing of the current message bit -- are labeled with `x[n]/p0p1`.

You can read the transition diagram as follows: "If the encoder is currently in state 11 and if the current message bit is a 0, transmit the parity bits 01 and move to state 01 before processing the next message bit." And so on, for each combination of current states and message bits. The encoder starts in state 00 and you should assume that all the bits before the start were 0.

A stream of parity bits corrupted by additive noise arrives at the receiver. Using the information in these parity bits, the decoder attempts to deduce the actual sequence of states traversed by the encoder and recover the transmitted message. Because of errors, the receiver cannot know exactly what that sequence of states was, but it can determine the **most likely** sequence using the Viterbi decoding algorithm.

Smallest # errors

The Viterbi decoder works by determining a **path metric** `PM[s,n]` for each state `s` and bit time `n`. Consider all possible encoder state sequences that cause the encoder to be in state `s` at time `n`. In hard decision decoding, the most-likely state sequence is the one that produced the parity bit sequence nearest in Hamming distance to the sequence of received parity bits. Each increment in Hamming distance corresponds to a bit error. `PM[s,n]` records this smallest Hamming distance for each state at the specified time.

The Viterbi algorithm computes `PM[…,n]` incrementally. Initially

```
PM[s,0] = 0 if s == starting_state else ∞
```

The decoding algorithm uses the first set of `r` parity bits to compute `PM[…,1]` from `PM[…,0]`. Then, it uses the next set of `r` parity bits to compute `PM[…,2]` from `PM[…,1]`. It continues in this fashion until it has processed all the received parity bits.

Here are the steps for computing `PM[…,n]` from `PM[…,n-1]` using the next set of `r` parity bits to be processed:

For each state s:

1. Looking at the encoder's state transition diagram, determine the two predecessor states α and β, which have transition arcs that arrive at state s.

   ? how do programatically ?

   Using the rate 1/2, K=3 encoder above, if s is 10 then, in no particular order, α = 00 and β = 01.

2. For the state transition α→s determine the r parity bits that the encoder would have transmitted; call this r-bit sequence p_α. Similarly, for the state transition β→s determine the r parity bits the encoder would have transmitted; call this r-bit sequence p_β.

   Continuing the example from Step 1: p_α = 11 and p_β = 01.

3. Call the next set of r received parity bits p_received. Compute the Hamming distance between p_α and p_received. This Hamming distance is a **branch metric** for the state transition α→s, so we'll label it BM_α. Similarly, compute the Hamming distance between p_β and p_received; we'll call it BM_β.

   Continuing the example from Step 2: assuming the received parity bits p_received = 00, then BM_α = hamming(11,00) = 2 and BM_β = hamming(01,00) = 1. ? do compare separate ? so can do soft

4. Compute two trial path metrics that correspond to the two possible paths leading to state s:

   ```
   PM_α = PM[α,n-1] + BM_α
   PM_β = PM[β,n-1] + BM_β
   ```

   PM_α is the Hamming distance between the transmitted bits and the parity bits received so far, assuming that we arrive at state s via state α. Similarly, PM_β is the Hamming distance between the transmitted and parity bits received so far, assuming we arrive at state s via state β.

   Continuing the example from Step 3: assuming PM[α,n-1]=5 and PM[β,n-1]=3, then PM_α=7 and PM_β=4.

5. Now compute PM[s,n] by picking the smaller of the two trial path metrics. Also record which state we chose to be the most-likely predecessor state for s:

   ```
   if PM_α <= PM_β:
       PM[s,n] = PM_α
       Predecessor[s,n] = α
   else
       PM[s,n] = PM_β
       Predecessor[s,n] = β
   ```

   Completing the example from step 4: PM[s,n]=4 and Predecessor[s,n]=β.

So never drop steps

We can use the following recipe at the receiver to determine the transmitted message:

1. Initialize PM[...,0] as described above.

2. Use the Viterbi algorithm to compute `PM[...,n]` from `PM[...,n-1]` and the next `r` parity bits; repeat until all received parity bits have been consumed. Call the last time point N.

3. Identify the most-likely ending state of the encoder by finding the state `s` that has the mimimum value of `PM[s,N]`. If there are several states with the same minimum value, choose one arbitrarily.

4. Determine message bit that caused the transition to state `s`. You can make this determination simply from knowing `s`; you don't need to know the predecessor state. Set `s=Predecessor[s,N]`, decrement N, and repeat this step as long as N > 0, accumulating the message bits in reverse order. So never "dump" unneeded steps

In this task we'll write the code for some methods of the `ViterbiDecoder` class. One can make an instance of the class, supplying `K` and the parity generator functions, and then use the instance to decode messages transmitted by the matching encoder.

The decoder will operate on a sequence of received voltage samples; the choice of which sample to digitize to determine the message bit has already been made, so there's one voltage sample for each bit. The transmitter has sent a 0 Volt sample for a "0" and a 1 Volt sample for a "1", **but those nominal voltages have been corrupted by additive Gaussian noise zero mean and non-zero variance**. Assume that 0's and 1's appear with equal probability in the transmitted message.

glist = parity bits
- so they calc when you call it

PS5_1.py is the template file for this task (not shown inline here).

You need to write the following functions: ← so just small parts?

`number = branch_metric(self,expected,received)` ← expected for what

what is this
- oh for the branch
- Oh given!
- don't worry how they got it

expected is an `r`-element list of the expected parity bits (or you can also think of them as voltages given how we send bits down the channel). *received* is an `r`-element list of actual sampled voltages for the incoming parity bits. In this task, we will do hard decision decoding, so digitize the received voltages to get bits and then compute the Hamming distance between the expected sequence and the received sequences. That's the branch metric.

Consider using `PS5_tests.hamming(seq1,seq2)` which computes the Hamming distance between two binary sequences. Oh done already

Can't write yet!
- they should do in strepps

`viterbi_step(self,n,received_voltages)` - can pass in string

compute `self.PM[...,n]` from the batch of `r` parity bits and the path metrics for `self.PM[...,n-1]` computed on the previous iteration. Follow the algorithm described above. In addition to making an entry for `self.PM[s,n]` for each state `s`, keep track of the most-likely predecessor for each state in the self.Predecessor array. You'll find the following instance variables and methods useful (scan the code to understand how to use them):

```
self.predecessor_states
self.expected_parity ←
self.branch_metric()
```

called each time step?

n = time

`s,n = most_likely_state(self,n)`

Identify the most-likely ending state of the encoder by finding the state `s` that has the mimimum value of `PM[s,n]`, where `n` points to the last column of the trellis. If there are several states with the same minimum value, choose one arbitrarily. Return the info as a tuple `(s,n)`.

already know

`message = traceback(self,s,n)`

Starting at state `s` at time `n`, use the `self.Predecessor` array to trace back through the trellis along the most-likely path, determining the corresponding message bit at each time step. Note that you're tracing the path *backwards* through the trellis, so that you'll be collecting the message bits in reverse order -- don't forget to put them in the right order before returning the final decoded message at the result of this procedure. Hint: you can determine the message bit at each state along the most likely path from the state itself -- think about how the current message bit is incorporated into the state when the transmitter makes a transition in the state transition diagram. Be sure that your code will work for different [illegible] because we'll be testing it at least for `K=3` and `K=4` codes.

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111
~~8~~	

do step by step

Get from state to message

no just append ? how? transition diagram? w/o having to regenerate? Oh expected bits!

The testing code at the end of the temp[illegible]ages and reports whether the decoding was successful or not.

File to upload for Task 1: Browse...

(points: 8)

Here's the debugging printout generated when Alyssa P. Hacker ran the example from lecture.

PM

Pred

IR

x[n-1] x[n-2] x[n-3] etc

```
1 [  2.00000000e+00   1.00000000e+06   0.00000000e+00   1.00000100e+06] [0 2 0 3]
2 [ 3.  1.  3.  1.] [0 2 0 2]
3 [ 2.  2.  2.  2.] [1 3 1 3]  error
4 [ 2.  3.  3.  2.] [0 3 1 2]
5 [ 3.  2.  3.  4.] [0 3 0 2]
6 [ 2.  4.  4.  4.] [1 2 0 2]
```

error

Oh that is not what I was interested in

Each line shows the new column added to the PM and Predecessor matrices after each call to `viterbi_step`. The last line indicates that the most-likely final state is 0 and that there were two bit errors on the most-likely path leading to that state.

A. Consider the most-likely path back through the trellis using the Predecessor matrix. At which processing steps did the decoder detect a transmission error? Call the first line of the debugging printout Step 1; the next, Step 2; etc.

— oh only use first!

even though wrong dir

Step 6 0
1
3
2 (error)
2 (error)

inital not set up w/ ∞

(points: 0.5)

Ben Bitdiddle ran Task #1 and found the following path metrics at the end for the four states:

```
[ 6200.   6197.   6199.   6199.]
```

after long message? w/ lots of errors

B. What is the most likely ending state? And what can you say about the number of errors the decoder detected? Were they all corrected?

?Qu? wrong?

too big!??

(points: 0.5)

Here's an example printout from running decoder tests with a 500000-bit message using two different convolutional codes, one with a contraint length of 3 (with generators 111 and 110), the other with a constraint length of 4 (with generators 1101 and 1110).

found w/ research

```
**** ViterbiDecoder: k=3, rate=1/2 ****
BER: without coding = 0.006200, with coding = 0.000218
Trellis state at end of decoding:
[ 6200.   6197.   6199.   6199.]

**** ViterbiDecoder: k=4, rate=1/2 ****
BER: without coding = 0.006200, with coding = 0.000022
Trellis state at end of decoding:
[ 6203.   6204.   6205.   6200.   6203.   6202.   6203.   6202.]
```

Please answer the following questions based on the printout from the k=3, rate=1/2 decoder.

2 bits for every message

C. What was the most-likely final state of the transmitter? And what were the most-likely final *two* bits of the message? You can answer both questions from the final trellis state shown in the printout.

(points: 0.5)

D. The test message used was 500,000 bits long. Since we're using a rate=1/2 convolutional code, there were 1,000,000 parity bits transmitted over the channel. The BER "without coding" was calculated by comparing the transmitted bit stream that went into the noisy channel with the bit stream that came out of the noisy channel. How many transmission errors occurred?

$$BER = \frac{\text{\# of bits wrong}}{\text{\# total bits}}$$

Just without

$.006200 = \frac{x}{1{,}000{,}000}$ $x = 6200$

$.000218 = \frac{x}{1{,}000{,}000}$ $x = 218$

(points: 0.5)

E. After passing the received parity bits through the decoder and calculating the most-likely transmitted message, the BER "with coding" was calculated by comparing the original message with the decoded message. In this example, the density of bit errors was high enough that the decoder was unable to find and correct all the transmission errors. How many uncorrected errors remained after the decoder did the best it could at correcting errors?

Just 218?

(points: 0.5)

Now consider what happened when testing the the k=4, rate=1/2 decoder.

F. What was the most-likely final state of the transmitter? And what were the most-likely final *three* bits of the message? You can answer both questions from the final trellis state shown in the printout.

3
011

(points: 0.5)

G. How many uncorrected errors remained after the decoder did the best it could at correcting errors?

22

easy qu!

(points: 0.5)

H. Consider a given message bit -- how many transmit bits are influenced by that message bit in both the k=3 and k=4 cases? How does that difference improve the error correction capability of the k=4 code?

Oh earlier I confused message + transmit bits

(points: 0.5)

Python Task 2: Decoding with a soft-decision branch metric

Useful download link:

PS5_2.py -- template file for this task

Let's change the branch metric to use soft decision decoding, as described in lecture.

The soft metric we'll use is the **square of the Euclidean distance** between the received vector of dimension `r`, the number of parity bits produced per message bit, of voltage samples and the expected `r`-bit vector of parities. Just treat them like a pair of `r`-dimensional vectors and compute the squared distance.

Oh just

PS5_2.py is the template file for this task (not shown inline here).

$[rec[0] - exp[0])^2 + [rec[1] - exp[1])^2$

Complete the `branch_metric` method for the `SoftViterbiDecoder` class. Note that other than changing the branch metric calculation, the hard decision and soft decision decoders are identical. The code we have provided runs a simple test of your implementation. It also tests whether the branch metrics are as expected.

have to limit first

File to upload for Task 2: Browse...

- Totally wrong

- float?

(points: 2)

not using!

Python Task 3: Comparing the performance of error correcting codes

Oh in P-set I don't convert to int - well do in Hamming Fn

Useful download link:

PS5_3.py -- template file for this task

In this task we'll run some experiments to measure the probability of a decoding error (i.e., the bit error rate) using the codes you implemented in Tasks 2, 3, and 4 above. There's no additional code for you to write, just some results to analyze. The codes all have rate 1/2:

And no limiting (its changed later)

Super!

Cho 4

1. The stream code using rectangular parity from Task 2. What should `nrows` and `ncols` be if we want a rate 1/2 code?
2. A convolutional code with hard decision decoding and constraint length 3.
3. A convolutional code with hard decision decoding and constraint length 4.
4. A convolutional code with soft decision decoding and constraint length 3.

45 min!

5. A convolutional code with soft decision decoding and constraint length 4.

PS5_3.py is the template file for this task.

Run the code -- please be patient, it takes a while; you can do something else while this script does its job. In the end you should see a set of plots of the probability of decoding error (the bit error rate) as a function of the amount of noise on the channel (the x axis is proportional to log(1/noise), tracking the "signal to noise ratio" (SNR) of the channel. Larger values of x correspond to **lower** noise, and a lower error probability for a given coding scheme. Note that the y axis is also on a log scale.

The Task 3 script saves the graph in a .png file called PS5_plots.png -- please upload the result:

Plot to upload for Task 3: Browse... ✓

(points: 1)

Looking the generated graph, for an SNR of 2 db list the codes in order of performance, from highest to lowest (better performing codes have a lower BER for a given SNR). Considering the Viterbi codes, briefly comment on the how much extra work the receiver has to perform as the code performance improves.

(points: 1)

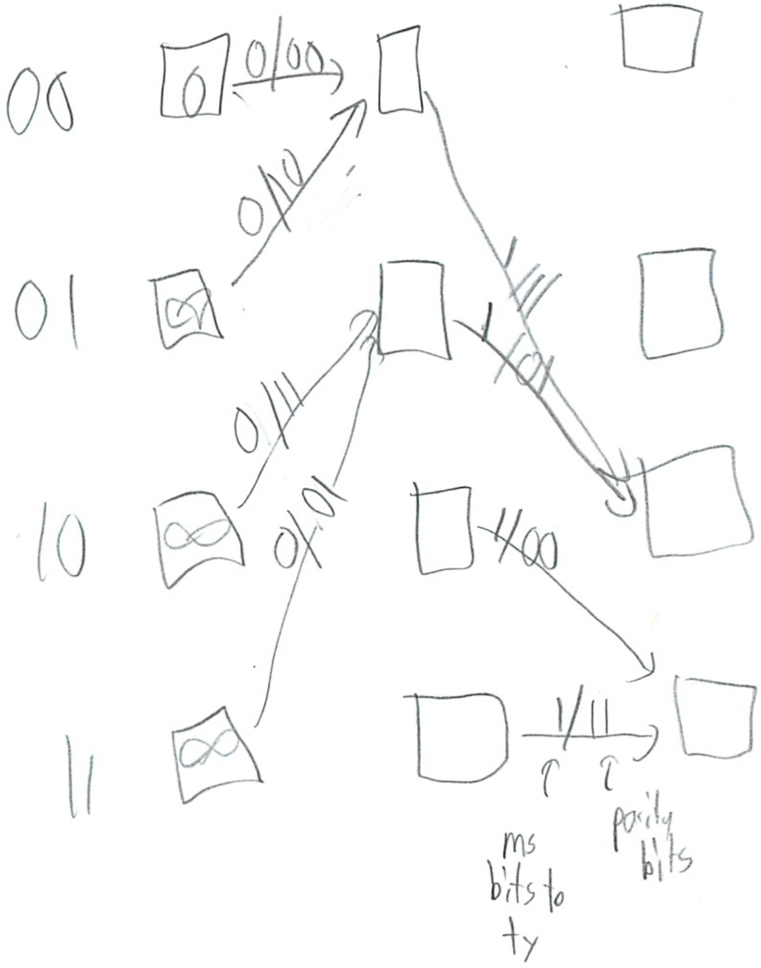
You can save your work at any time by clicking the Save button below. You can revisit this page, revise your answers and SAVE as often as you like.

Save

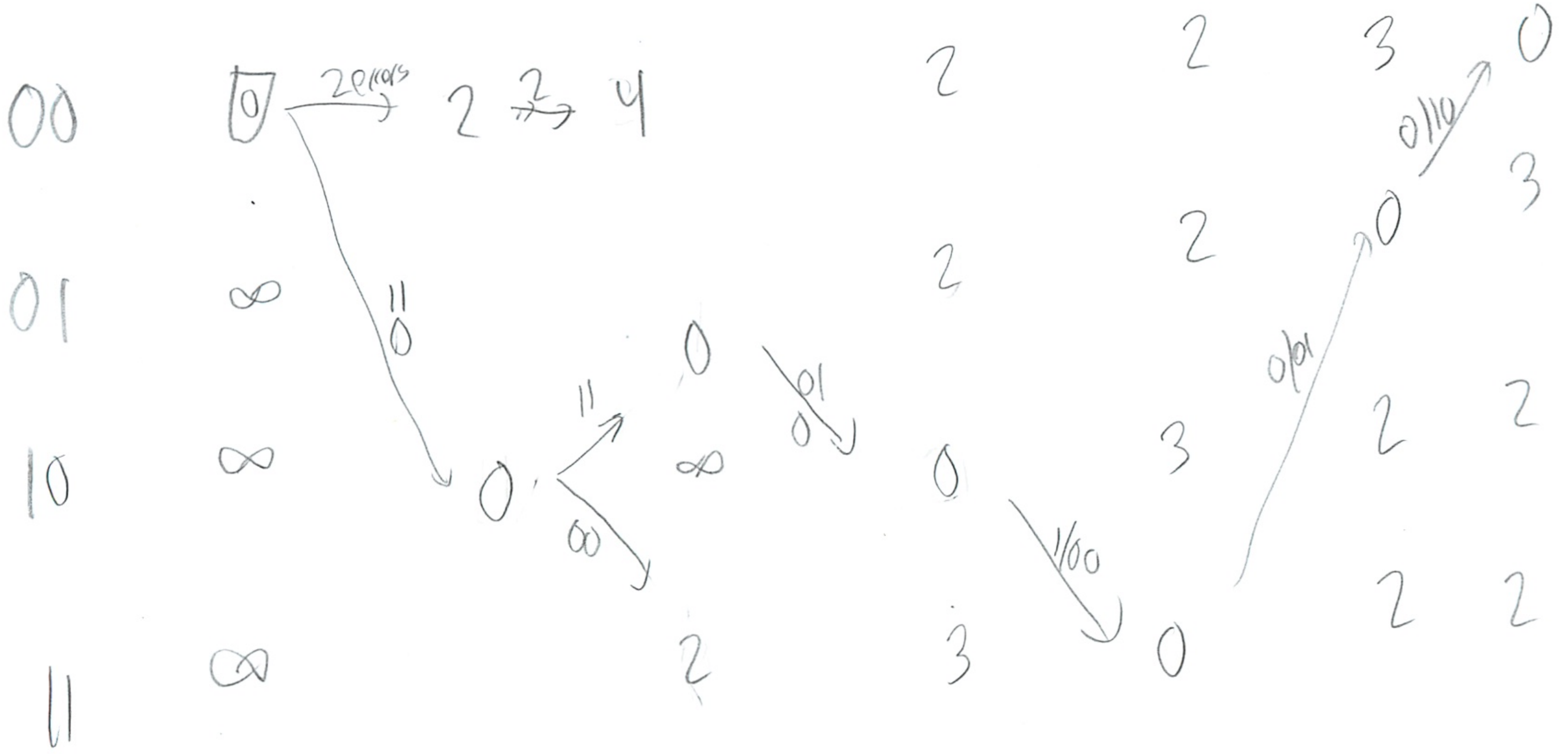
To submit the assignment, click on the Submit button below. YOU CAN SUBMIT ONLY ONCE after which you will not be able to make any further changes to your answers. Once an assignment is submitted, solutions will be visible after the due date and the graders will have access to your answers. When the grading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

Submit

Message	1	0	1	1	0	0
Generic Tx	11	11	01	00	01	10



Message	1	0	1	1	0	0
Tx	11	11	01	00	01	10



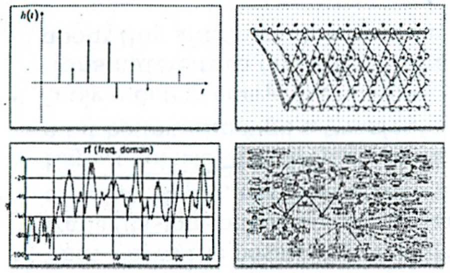
So past states 0 1 3 2 1 2 | 0

ms 0 0 1 1 0 1 | 0

← backwards

10 11 00 ✓

Oh never did first msg bit!



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #13

- unslotted Aloha
- carrier sense, contention windows
- code division glimpse

best $\frac{1}{N}$

Contention protocol

Who gets to talk?

Not everyone always using bandwidth

Stabalized

Slotted Aloha Summary

- Assumptions
  - Time is divided into slots
  - Transmissions begin on slot boundaries
  - Packets are 1 slot long
- Choose to transmit packet with probability *p*
  - Each nodes uses collisions/success to adjust p
  - Adjust *p* between p_{min} (avoid starvation) and p_{max} (avoid capture)
  - On collision $p \leftarrow max(p_{min}, p/2)$, on success $p \leftarrow min(p_{max}, 2*p)$
- Utilization = throughput achieved/maximum data rate
  - *U* = prob(exactly one transmission in a slot) $= N*p*(1-p)^{N-1}$, where *N* is number of backlogged nodes
  - Maximized when $p = 1/N$, $U_{max} = (1 - 1/N)^{N-1}$
  - As $N \rightarrow \infty$, $U_{max} \rightarrow 1/e \approx 37\%$
  - While each node's *p* is never exactly *1/N*, the goal is to have its average value over modest intervals be approximately *1/N*.

Stabalized

but upper + lower bounds to prevent starvation

~37% - looks low but pretty good

and ~~[illegible]~~ very simple - ~~[illegible]~~

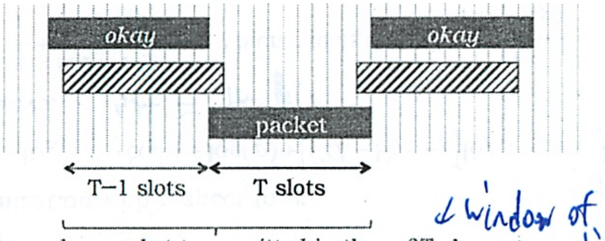
but can't get exactly .125 : .8 : .4 : .2 : .1 .05 but on avg will get

any overlap collides

Unslotted Aloha

- Packets take T time slots to transmit
  - As slots get smaller and T grows, approximates transmission at arbitrary times. continous-ish
- Collisions are no longer "perfect"
  - Any overlap between multi-slot packets is a collision
  - Larger window of vulnerability to other transmissions

no more nice slots



Any other packet transmitted in these 2T−1 time slots will collide with target packet

↓ window of vuneability

Utilization in Unslotted Aloha

Probability of no transmission for 2T−1 slots:

$$(1-p)^{2T-1}$$

Probability of a sender experiencing no collisions:

$$\geq \left[p(1-p)^{2T-2}\right]\left[(1-p)^{2T-1}\right]^{N-1} = p(1-p)^{(2T-1)N-1}$$

= for nodes that try to send new packet while busy with last one!

Exactly = for OCD nodes

$(1-p)^{2T-1}$ for not self interference

Self-interference

Utilization = throughput/maximum rate:

$$\boxed{U_{\text{unslotted Aloha}} \geq \frac{Np(1-p)^{(2T-1)N-1}}{1/T} = TNp(1-p)^{(2T-1)N-1}}$$

lower band

3/16

U_{max} for Unslotted Aloha

Maximization with respect to *p*:

$$\log(\ldots) = \text{const} + \log(p) + \left[(2T-1)N-1\right]\log(1-p)$$

Paiser to max, log of expression

Derivative: Set = to p0

$$\frac{1}{p} + \frac{(2T-1)N-1}{1-p}, \text{ which equals 0 at } p = \frac{1}{(2T-1)N}$$

is it bigger or smaller

Plugging back into *U*:

$$U_{max} = \frac{T}{2T-1}\left(1-\frac{1}{(2T-1)N}\right)^{(2T-1)N-1}$$

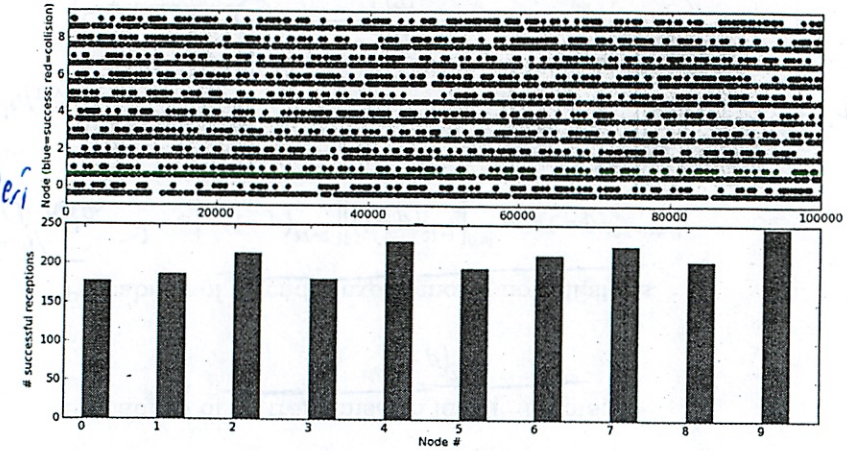
↓small time slots

For large *N*: $U_{max} \approx \left(\frac{T}{2T-1}\right)\frac{1}{e}$ For large *N, T*: $U_{max} \approx \frac{1}{2e}$

Half the utilization of slotted Aloha:
makes sense: twice the window of vulnerability

interested in asymptotic behavior

Simulation of Unslotted Aloha



Utilization = .21, Fairness = .99

↓size of 10

```
python PS6_3.py -r -n 10 -t 100000 --pmin=.005 --pmax=0.8 -s 10
```

had to drop down

Carrier Sense

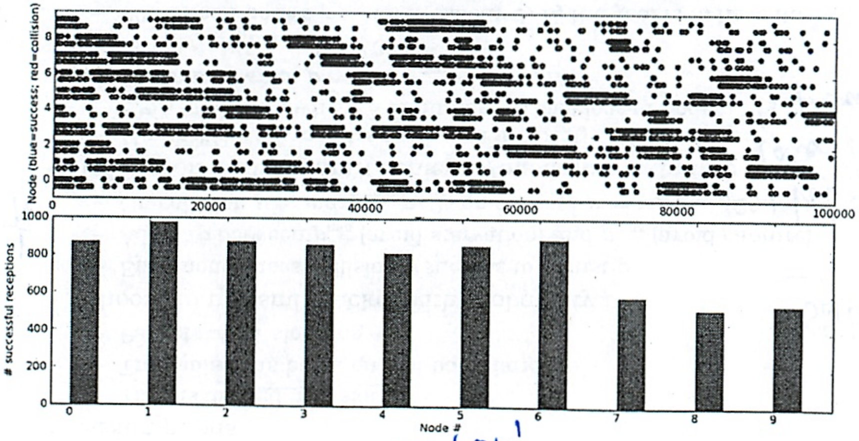
to see if someone else is trans

just look to see that channel no busy wait for idle period

- Reduce collisions with on-going transmissions by transmitting only if channel appears not to be busy.
- For large T (slots/packet) if channel is busy this cycle, the same sender will probably be transmitting more of their packet next cycle
- When the channel is idle, there's no chance of interrupting an on-going transmission.
- That leaves the possibility of colliding with another transmission that starts at the same time – a one slot window of vulnerability, not 2T-1 slots.
- Expect collisions to drop dramatically, utilization to be quite a bit better, although a "wasted" slot is now necessary
- Busy = detect energy on channel. On wireless channels, transmitters turn on carrier to transmit (we'll learn more about this after break), hence the term "carrier sense".

detecting business is hard, slow

Simulation of Carrier Sense



Wow!

Utilization = .78, Fairness = .96

```
python PS6_4.py -r -n 10 -t 100000 --pmin=.005 --pmax=0.8 -s 10
```

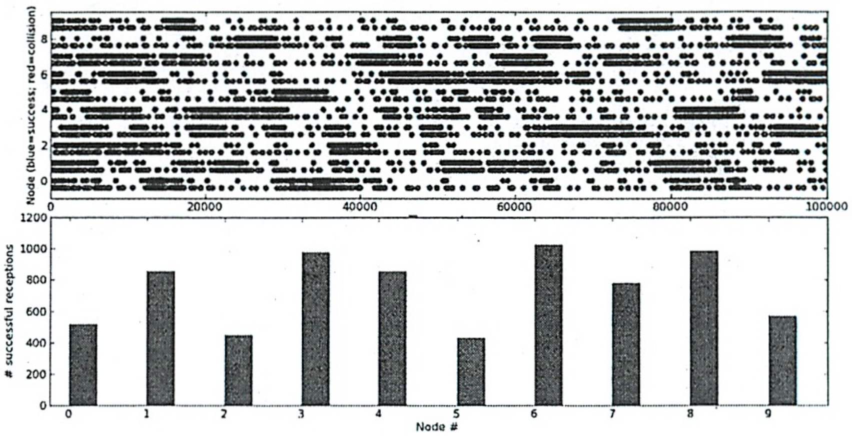
Only 15 characters to change

Request to send
Clear to send
but spread out requests

Contention Windows

- Contention Window: parameter is some integer *CW*
- When node wants to transmit, it picks a random number *r* uniformly in [1,CW] and sends after the r^{th} idle slot from the current time. size = # timeslots
- If transmission collides: $CW \leftarrow max(CW_{min}, CW/2)$
  If transmission succeeds: $CW \leftarrow min(CW_{max}, CW*2)$ not much contension
- Node is guaranteed to attempt a transmission within CW slots. With the earlier scheme, there was always the chance (though exponentially decreasing) that a node may not transmit within some fixed number of time slots.

Simulation of Contention Windows



Utilization = .74, Fairness = .92

```
python PS6_5.py -r -n 10 -t 100000 -W 256 -s 10
```

Summary of MAC Protocols

- Goal of MAC protocols is to maximize utilization and fairness
- TDMA is a good choice when nodes are all (or mostly) backlogged like Cell Phones
  - Round-robin sharing provides known communication capacity and bounded wait
  - It's precisely fair, 100% utilization if all nodes have packets
  - Poor choice when traffic is bursty or if some nodes have a higher offered load than others
- Contention protocols dynamically adapt to changing traffic data transfer
  - Distributed protocol (each node makes its own decisions based on transmission experience) avoids cost of centralized controller having to know which nodes have packets to send
  - Parameter (p, CW) that controls when packets are sent is adjusted so that prob(sending packet) is lowered when collisions are detected and raised when transmissions are successful.

CW dynamically updated

Summary (cont'd.)

- Slotted Aloha – based on very simple rule: transmit with probability p.
  - Dynamic adjustment of p "stabilizes" the protocol.
    - Use binary exponential backoff to adjust p downward
  - Utilization maximized when p = 1/(number of backlogged nodes)
  - For large numbers of backlogged nodes U ≈ 1/e ≈ 37%
  - For fairness: $p_{min} \leq p \leq p_{max}$
- Unslotted Aloha – packets take multiple time slots to send, models transmissions at arbitrary times
  - Gets half of the max utilization of slotted Aloha due to doubled window of vulnerability to collisions
  - Carrier sense avoids collisions from packets once transmission has started → much better utilization
  - Fairness still requires bounds on p

dot product $\vec{A}\cdot\vec{B}=0$

Code Division Multiple Access (CDMA)

- Two vectors are orthogonal if their dot products are 0. Here's a set of 4 mutually orthogonal vectors:

```
– V1: (1,  1,  1,  1)
  V2: (1,  1, -1, -1)
  V3: (1, -1,  1, -1)
  V4: (1, -1, -1,  1)
```

chip

- Assign each transmitter a particular orthogonal vector (Vi) it will use to encode its transmissions (called the "chip code"). With vectors shown above we can support 4 transmitters.
  - If message bit is 0, transmit –Vi
  - If message bit is 1, transmit Vi

  } 1 message bit → len(Vi) "chips"

4 bits transmitted for each message bit

- Channel will sum the transmitted values:

```
– send 00 using V1: -1 -1 -1 -1 -1 -1 -1 -1
  send 01 using V2: -1 -1  1  1  1  1 -1 -1
  send 11 using V3:  1 -1  1 -1  1 -1  1 -1
  send 10 using V4:  1 -1 -1  1 -1  1  1 -1
  channel:           0 -4  0  0  0  0  0 -4
```

Sums up in air

#can get large

– biggest prob: don't make message so large ~~am~~ recieve can't sense

Asynchronous CDMA

- Use N orthogonal vectors to multiplex N transmitters (e.g., use a NxN Walsh/Hadamard Matrix) all rows + columns are orthagonal
- Scheme described above works for synchronous CDMA when all symbols are transmitted starting at same moment. For example this works fine for a cell tower transmitting to mobile phones. not accurate enough
- But hard to synchronize mobile phone transmissions, so use asynchronous CDMA:
  - Can't create transmissions that are truly orthogonal if they start at different times
  - Approximate orthogonality with longer uncorrelated pseudo-random sequences (called pseudo-noise or PN). "pseudo" implies that sequence can be reconstructed at receiver given a known starting point. Overlaps uncorrelated
  - Assuming equal signal strengths from each transmitter at receiver, if we decode bits using a particular PN sequence synchronized with desired transmitter, we'll get desired signal plus some uncorrelated noise from other transmitters.

CDMA Receiver

- At receiver take groups of len(V) bits and form dot product with Vi for desired channel.
  - If result is negative, message bit is 0
  - If result is positive, message bit is 1

```
channel:             0 -4  0  0  0  0  0 -4

receive using V1:    1  1  1  1  1  1  1  1
dot product:                 -4          -4
message bits:                 0           0

receive using V2:    1  1 -1 -1  1  1 -1 -1
dot product:                 -4           4
message bits:                 0           1

receive using V4:    1 -1 -1  1  1 -1 -1  1
dot product:                  4          -4
message bits:                 1           0
```

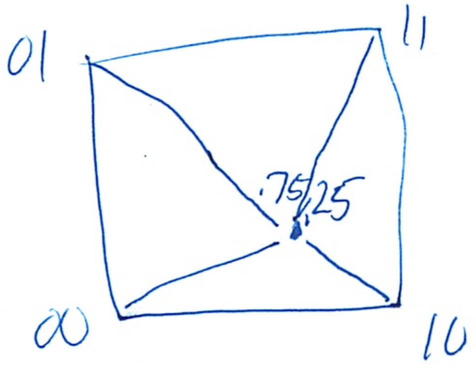
← chip codes for each V.

dot product for each Vi assigned

V3 no problem either – just out of space on slide

P-Set 5 Interview

3/6



$X^2 + Y^2$

Cases when close to Threshhold

SNR - ratio $\frac{\text{signal}}{\text{noise}}$

~~lower~~ higher better

usually log

Charastistic of channel

k=3,4 both have gain $\frac{1}{2}$

↑ history

↑ # generators

rect parity - last pset

1	0	
1	0	

4 8

rates $\frac{1}{2}$

all have same rate

6.02 Recitation

3/17

(15 min late)

Unslotted Aloha

- Finding the probabilities of collission

(skipping old stuff on board)

$$U(T,N,p) = T \cdot p(1-p)^{N(2T-1)-1}$$

$$\frac{\partial U}{\partial p} = T(1-p)^{N(2T-1)-1} + Tp(1-p)^{N(2T-1)-2} \cdot (-1) \cdot N(2T-1)-1)$$

$$= T(1-p)^{N(2T-1)-2}\left((1-p) - p(N(2T-1)-1)\right)$$

$= 0$ set = to 0 and solve for p

$$\frac{p}{1-p} = \frac{1}{N(2T-1)-1} \rightarrow p^* = \frac{1}{N(2T-1)}$$

$$U^* = \frac{T}{N(2T-1)} \cdot \left(1 - \frac{1}{N(2T-1)}\right)^{N(2T-1)-1}$$ for one node

$$U_{total} = U^* N$$

$$= \frac{T}{2T-1}\left(1 - \frac{1}{N(2T-1)}\right)^{N(2T-1)}\left(1 - \frac{1}{N(2T-1)}\right)^{-1}$$

When large T, N

$$\underbrace{\uparrow}_{\frac{1}{2}} \qquad \underbrace{\qquad \uparrow \qquad}_{\frac{1}{e}} \qquad \underbrace{\quad \uparrow \quad}_{1}$$

$$= \frac{1}{2e}$$ cost of lack of sync.

These formulas were with self transmission (OCD nodes)

keep for math

Does not really matter at large N,T

$$P_{sucess} = p \underbrace{(1-p)^{2T-2}}_{\text{self tars}} \left((1-p)^{2T-1}\right)^{N-1}$$

$$= p(1-p)^{(2T-1)(N-1)+2T-2}$$

With Carrier Sense

- Other ~~people~~ transmitters can listen to channel and listen it someone transmitting
- if They hear a packet on network, wait for ~~it to end firs~~ idle time
- Only really works on Short ping time

2 types of slots

- waste
  - idle since no one wants to talk
  - Collission
  - idle since people want to transmit, but p is low
- Useful

What fraction of slots are useful?

Slot can be in useful or wasteful state

Ⓦ Ⓤ

⑤

CDMA

Each transmitter gets a "signature"

T_x	1	1	1	1
$\{-1, 1\} \ni b$	b_1	b_1	b_1	b_1
0 1				
	-1	-1	-1	-1

Multiply bit by signature value each time

T_x	v_1	v_2	$\cdots$	v_{2n}
0 -1	b_1	b_1		b_1
1 1				

S_1	S_2	$\cdots$	S_{1x}
+	+		+
S'_1	S'_2		S'_2
$\parallel$	$\parallel$		$\parallel$
Y_1	Y_2		Y_n

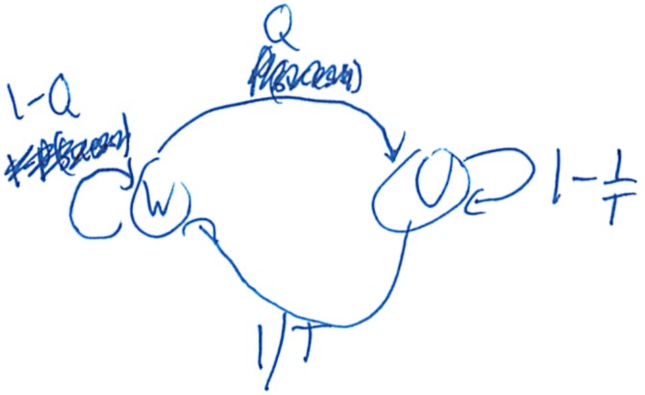
③

Making small change to protocal; At end of each timeslot

Stop w/ prob $\frac{1}{T}$
Contine otherwise

So transmit T slots at a time on average

For the System



After stop transmitting will be 1 wasteful slot

Next w/ another transmitters will starte sucesfully
Q
↳ prob. that a node becomes sucessful

Or with ~~At~~ 1-Q there is a ~~conflict~~ collission and still useless

$Q = N \cdot p \cdot (1-p)^{N-1}$ 1 of the people (N of them) must trans - the rest must be silent

Now solve Markov Chain to find steady state (remember 6.042)

$$P_W = P_W(1-Q) + P_U\left(\frac{1}{T}\right)$$

$P_U = - - - -$ See next pg ↓

$$P_W + P_U = 1$$

9

$$P_W \cdot Q = \frac{1}{T} P_U$$

$$P_W = \frac{1}{QT} P_U$$

$$P_U + \frac{1}{QT} \cdot P_U = 1$$

$$P_U = \left(\frac{1}{1 + \frac{1}{QT}} \right)$$

Best Q ~~was~~ $\left\{ \begin{array}{l} p = \frac{1}{N-1} \\ Q = Np(1-p)^{N-1} \end{array} \right.$ larger Q is better

$\sim \frac{1}{e}$

Greater if N is smal

$$P_U = \frac{1}{1 + \frac{e}{T}}$$

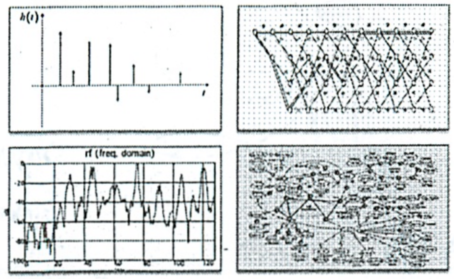
Say $T = 10e$ (related to lengh of packet)

$$P_U = \frac{1}{1 + \frac{1}{10}}$$

$$= \frac{10}{11} \approx 90\%$$

If you knew N

Rewatch 4/4



INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

Freq division multiplexing

6.02 Spring 2011
Lecture #14

- complex exponentials
- discrete-time Fourier series
- spectral coefficients
- band-limited signals

each gets small section of spectrum

- radio
- cable - internet and TV

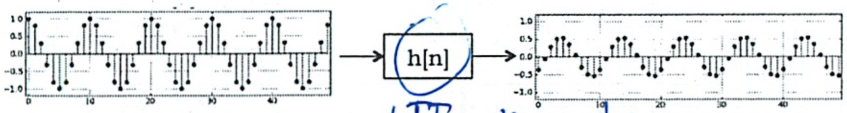
Frequency Division Multiplexing

To engineer the sharing of a channel through frequency division multiplexing we'll need a new set tools that will let us understand the behavior of signals and systems in the frequency domain. Plan:

- This week
  - Analyze the frequency content of signals using the discrete-time Fourier series
  - Determine what happens when we *band-limit* a signal
  - Characterize LTI systems by their frequency response either domain
  - Introduce *filters*: LTI systems that eliminate a region of frequencies from a signal remove energy from bands
- Next week
  - Using modulation to position band-limited signals in different regions of the frequency spectrum
  - Receiving a particular signal from a shared spectrum

need new tools

Sinusoids and LTI Systems



LTI system wire

Frequency division multiplexing depends on an interesting property of LTI channels:

> if the channel input x[n] is a sinusoid of a given amplitude, frequency and phase, the response will be a sinusoid *at the same frequency*, although the amplitude and phase may be altered. As we'll see, the change in amplitude and phase may depend on the frequency of the input.

through LTI same freq

The same property holds when the inputs are complex exponentials, which are closely related to sines and cosines (and, perhaps surprisingly, are much easier to analyze!).

tool used to analysic

Periodic Sequences all w/ period of 5

in discrete systems

A sequence x[n] is said to be periodic with a period of N samples ("periodic with period N") if

$$x[n] = x[n+N]$$

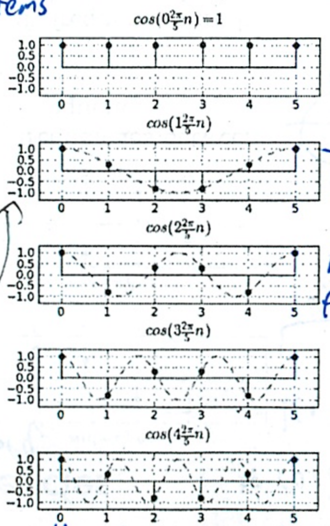
simply repeat

A sequence that is periodic with period N is also periodic with period 2N, 3N, …, and so on.

don't draw

A sinusoidal sequence that is periodic with period N can only have one of a finite number of frequencies: all the harmonics of the fundamental frequency 2π/N radians/sample, i.e., frequencies of the form k·(2π/N) for some integer k.

1 period



- Fund. freq

all harmonics of fund freq

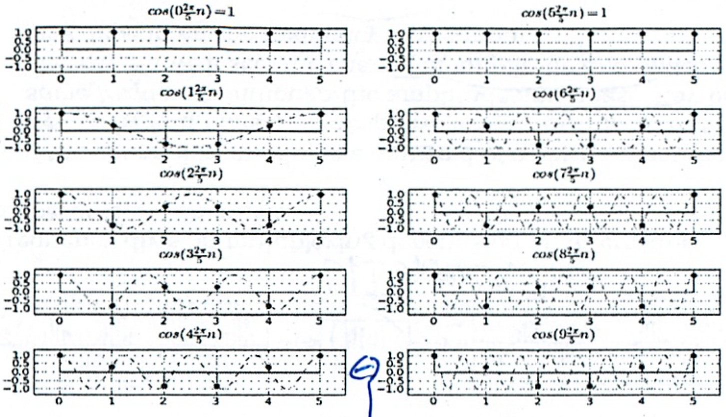
all periodic w/ period 5 — [illegible]
are no more possible

waveforms = value of seq

Frequencies k·(2π/N) when k ≥ N

5

Frequency k·(2π/N) yields the same sequence as (k mod N)·(2π/N)



← ks that ae ≥ 5

k=1

Unique sequences: k = 0, 1, 2, ..., N-1

Highest frequency: (N-1)·(2π/N) < 2π

its the same blue dots!

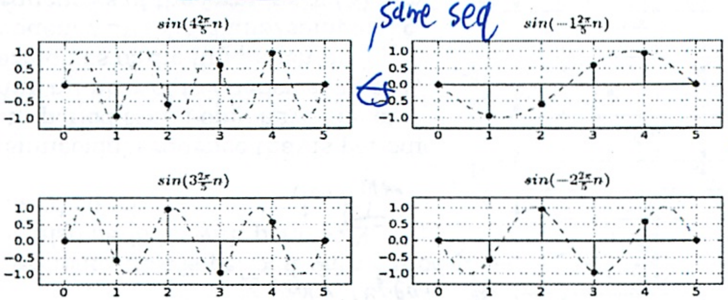
- discrete system can't tell diff. so only N possible freq

freq = # radians / sample

Negative Frequencies

- simply bookkeeping # radians/sample

same seq



$k \cdot \frac{2\pi}{N} \cdot n$

ω = rad/sample

Sequences of frequency k·(2π/N), i.e., frequencies between 0 and 2π, are identical to sequences of frequency -(N-k)·(2π/N), i.e., between -2π and 0.

e going this way

In 6.02, our convention will be to specify frequencies in the range -π and π, corresponding to k's in the range –(N/2) to (N/2).

$\frac{7\pi}{4} = -\frac{\pi}{4}$ (did a lot of this in 18s)

will go -π ← just packing up arrive at same pts

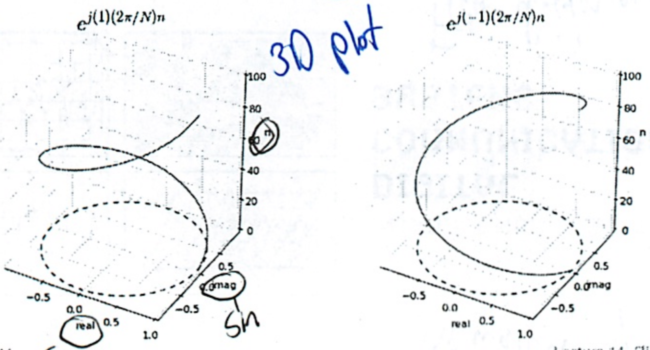
Complex Exponentials

A complex exponential is a complex-valued function of a single argument – an angle measured in radians. Euler's formula shows the relation between complex exponentials and our usual trig functions:

$$e^{j\varphi} = \overset{real}{\cos(\varphi)} + \overset{imag}{j\sin(\varphi)}$$

Produces a Complex value

3D plot



cos

sin

easier to work in exponentials

$\cos(\phi) \cdot \cos(\theta)$

$e^{j\phi} \cdot e^{j\theta}$

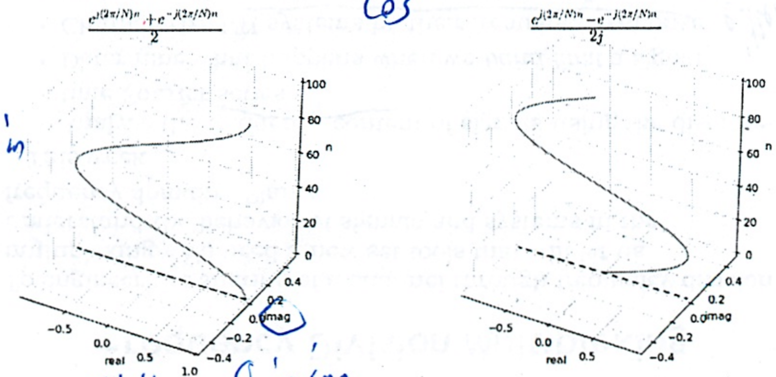
add exponents

Sine and Cosine and e^jφ

$$\cos(\varphi) = \frac{1}{2}e^{j\varphi} + \frac{1}{2}e^{-j\varphi}$$

Can write complex # as sin + cos

$$\sin(\varphi) = \frac{1}{2j}e^{j\varphi} - \frac{1}{2j}e^{-j\varphi}$$



Strictly real

0 in img plane

Useful Properties of $e^{j\varphi}$

When $\varphi = 0$:

$$e^{j0} = 1 = \cos(0) + j\sin(0) = 1 + 0j$$

When $\varphi = \pm\pi$:

$$e^{j\pi} = e^{-j\pi} = -1$$

$$e^{j\pi n} = e^{-j\pi n} = (-1)^n$$ alternates

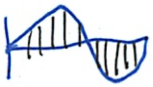
$e^{j\pi} + 1 = 0$

Summing samples over one period:

3rd interesting property

$$\sum_{n=\langle N\rangle} e^{jk\frac{2\pi}{N}n} = \begin{cases} N & k = 0, \pm N, \pm 2N, \ldots \\ 0 & \text{otherwise} \end{cases}$$

n ranges over any N consecutive integers, e.g., n = 0, 1, ..., N-1

when add  = 0

prove in tutorial problems

One of magic formulas of all time

Discrete-time Fourier Series

If x[n] is periodic with period N, it can be expressed as the sum of scaled periodic complex exponentials:

Can express any periodic fn as sum of scaled, complex exponentials

Complex exponential with period N and fundamental frequency 2π/N. – periodic w/ period n

$$x[n] = \sum_{k=\langle N\rangle} a_k e^{jk\left(\frac{2\pi}{N}\right)n}$$

constant that changes-complext

k ranges over any N consecutive integers. Two common choices:

- k starts at 0 (0 ≤ freq ≤ 2π) to N-1
- k starts at –N/2 (-π ≤ freq ≤ π) $\frac{N}{2}$

does not matter where start/stop

The *spectral coefficients* a_k for each of the discrete frequencies are, in general, complex, changing both the amplitude and phase of the associated complex exponential. If x[n] is real, $a_{-k} = a_k^*$. will change phases

Use to go back + forth time + freq terms

need to know a_k solve for

Solving for the a_k

Start with:

$$x[n] = \sum_{k=\langle N\rangle} a_k e^{jk\left(\frac{2\pi}{N}\right)n}$$

Multiply both sides by $e^{-jr(2\pi/N)n}$ and sum over N terms:

$$\sum_{n=\langle N\rangle} x[n] e^{-jr\left(\frac{2\pi}{N}\right)n} = \sum_{n=\langle N\rangle}\sum_{k=\langle N\rangle} a_k e^{jk\left(\frac{2\pi}{N}\right)n} e^{-jr\left(\frac{2\pi}{N}\right)n}$$

sum over N consecuitic integers

$$= \sum_{k=\langle N\rangle} a_k \underbrace{\sum_{n=\langle N\rangle} e^{j(k-r)\left(\frac{2\pi}{N}\right)n}}$$

this whole thing

From slide 9:
N if k=r, 0 otherwise

trying diff ks
– mostly 0
– except when k=r get N

$$= a_r N$$

k=r

$$\boxed{a_k = \frac{1}{N}\sum_{n=\langle N\rangle} x[n] e^{-jk\left(\frac{2\pi}{N}\right)n}}$$

Discrete-time Fourier Series Pair

F. series pair

$$x[n] = \sum_{k=\langle N\rangle} a_k e^{jk\left(\frac{2\pi}{N}\right)n}$$

freq to time

Synthesis equation

fourier series pair

$$a_k = \frac{1}{N}\sum_{n=\langle N\rangle} x[n]\, e^{-jk\left(\frac{2\pi}{N}\right)n}$$

spectral coeff.

Analysis equation

time to freq

If we have N samples of a periodic waveform of period N, we can find the waveform's spectral coefficients using the analysis equation.

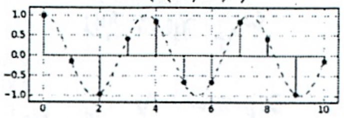
If we have the spectral coefficients, we can reconstruct the original time-domain waveform using the synthesis equation.

avg value sinusoid = 0

What are the a_k? - this is the crank the math way

$$x[n]=\cos(r\frac{2\pi}{N}n)$$

cos(3(2π/11)n)



$$a_k=\frac{1}{N}\sum_{n=\langle N\rangle}\cos(k\frac{2\pi}{N}n)e^{-jk\left(\frac{2\pi}{N}\right)n}$$

$$=\frac{1}{2N}\sum_{n=\langle N\rangle}\left[e^{jr\left(\frac{2\pi}{N}\right)n}+e^{-jr\left(\frac{2\pi}{N}\right)n}\right]e^{-jk\left(\frac{2\pi}{N}\right)n}$$

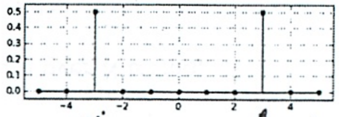
$$=\frac{1}{2N}\sum_{n=\langle N\rangle}e^{j(r-k)\left(\frac{2\pi}{N}\right)n}+\frac{1}{2N}\sum_{n=\langle N\rangle}e^{-j(r+k)\left(\frac{2\pi}{N}\right)n}$$

just crank the math

$$a_k=\begin{cases}\frac{1}{2} & k=\pm r\\ 0 & \text{otherwise}\end{cases}$$

rth spectral coefficent = 1/2

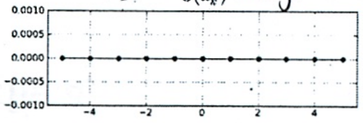
$\Re(a_k)$ reals



-3

3 of N

$\Im(a_k)$ imags



will write on p-set

r = index
= # cycles in N samples

$$x[n]=\sin(r\frac{2\pi}{N}n)$$

clever way

This time let's do it "by inspection". First rewrite x[n] (see slide #8):

$$x[n]=\frac{1}{2j}e^{jr\frac{2\pi}{N}n}-\frac{1}{2j}e^{j(-r)\frac{2\pi}{N}n}$$

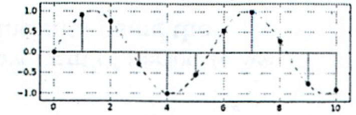
Now x[n] is a sum of complex exponentials and we can determine the a_k directly from the equation;

$$a_r=\frac{1}{2j}=-\frac{j}{2}$$

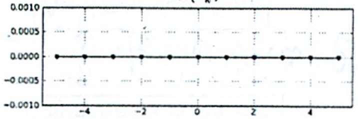
$$a_{-r}=-\frac{1}{2j}=\frac{j}{2}$$

$$a_k=0\quad\text{otherwise}$$

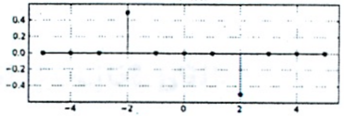
sin(2(2π/11)n)



$\Re(a_k)$



$\Im(a_k)$



avg will be 0

$$x[n]=1+2\cos(3\frac{2\pi}{11}n)-3\sin(5\frac{2\pi}{11}n)$$

Again, by inspection: since the cos and sin are different frequencies, we can analyze them separately.

a_0 = average value = 1

↓real

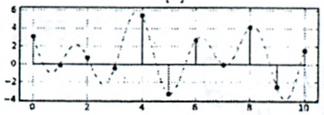
$a_{\pm3}$ = 2(1/2) = 1 [from cos term]

a_{-5} = -3(j/2) = -1.5j [from sin term]
a_5 = -3(-j/2) = 1.5j

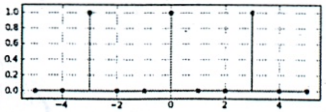
↑img

a_k = 0 otherwise

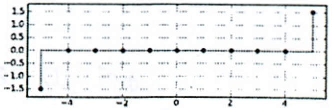
$x[n]$



$\Re(a_k)$



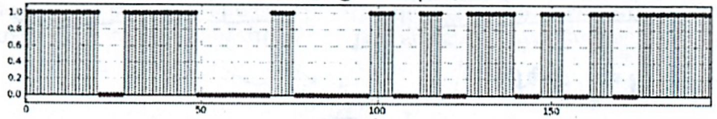
$\Im(a_k)$



use superposition to solve

Spectrum of Digital Transmissions

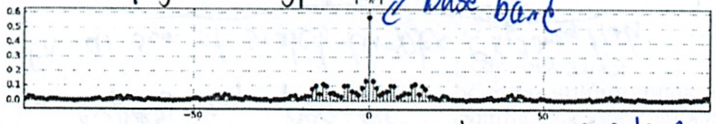
transmit @ 7 samples/bit



but using lots of spetrum!

$|a_k|$

base band

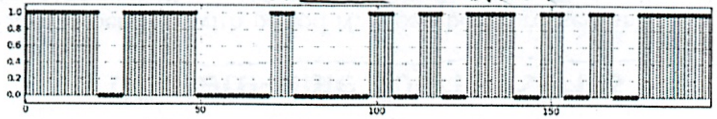


the spectral coefficents for the seq

↑ since slightly more

spectrum

x[n] synthesized from a_k

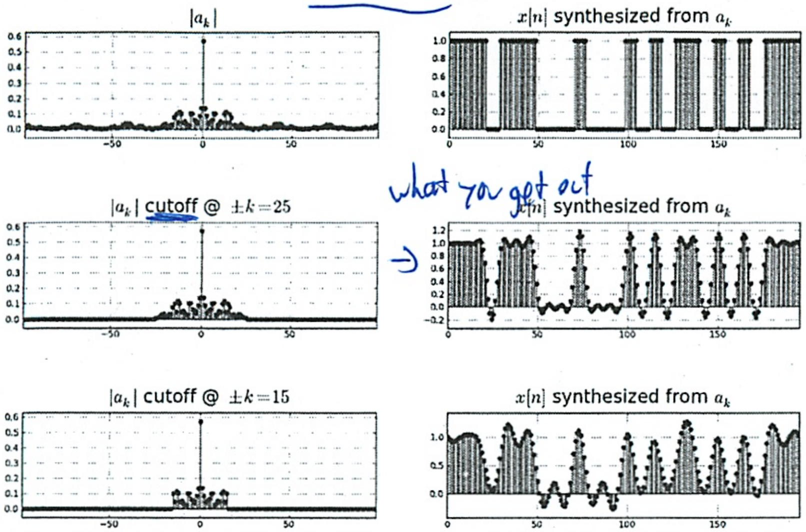


behaving the way we want - getting energy, at freq we want.

try to limit freqs

- to stay on spectrum

Effect of Band-limiting a Transmission



"magically" cheating here - just 0 out above a cutoff

what you get out

→

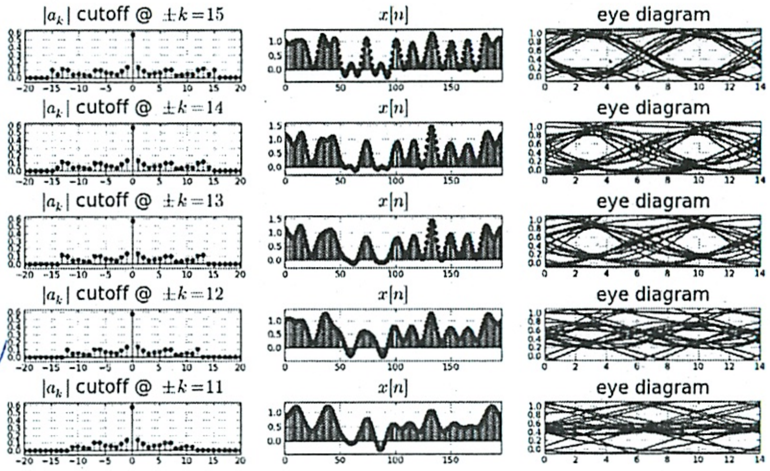
~~thus~~ limit freq used

square wave turning to sin wave - like if sin

since high freq responsible for the limited transitions

- like when it was slow channel

How Low Can We Go?



throw away more + more

7 samples/bit → 14 samples/period → k=(N/14)=(196/14)=14

Where do we expect eye to disappear?

$$\underbrace{|||||||......|}_{14 = \text{period}}||||||||$$

How many 14s in 196

$$k = \frac{196^{\leftarrow N}}{14} = 14$$

↑ coincidence

6.02 Recitation

3/29

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

$j = i$

↑ the CS notation

$$\underbrace{x}_{\text{real}} + \underbrace{jy}_{\text{complex}}$$

modulus $|x+jy| = x^2+y^2$

Remember $\cos^2\theta + \sin^2\theta = 1$

Can write cos, sin in form of complex #

$$e^{j\theta} = \cos\theta + j\sin\theta \qquad (a)$$

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

$$= \cos\theta - j\sin\theta \qquad (b)$$

If add - will cancle - get formula for cos

$$\frac{e^{j\theta} + e^{-j\theta}}{2} = \cos\theta \qquad \frac{(a)+(b)}{2}$$

$$\frac{e^{j\theta} - e^{-j\theta}}{2j} = \sin\theta \qquad \frac{(a)-(b)}{2j}$$

Represent time series in summation of complex exponentials

Look at <u>periodic</u> → it repeats

$x[n]$ period N

$x[n] = x[n+N]$ for all n

If want can represent as

$$x[n] = \sum_{k\in\langle N\rangle} a_k \cdot e^{j\cdot\frac{2\pi}{N}\cdot k\cdot n}$$

↑ summation of N functions, each function is nice

(c)
Distributed
Furier ← spelling
Transform

Can think of

$-\frac{N}{2}$ $\frac{N}{2}-1$

or

0 $N-1$

Choose N consecutive things

In class defined

$$a_k = \frac{1}{N}\sum_{n\in\langle N\rangle} x[n]\, e^{-j\cdot\frac{2\pi}{N}\cdot k\cdot n}$$

(d)
Inverse
DFT

Lots of devices use

Say get signal w/ period N

- get $\grave{x}[n]$

3

-Calculate a_k

How many calculations?

Each a_k has $x[n]$ terms

- each term 2 multiplications + a few additions

If have to do N things then

$$N\cdot N \underbrace{+c}_{\text{forget}}$$

Fast Forier Transform

- does $c + N\log N$
- is very, very important

To get (d) out of (c)

$$(c) \times e^{j\cdot r\cdot\frac{2\pi}{N}\cdot n}$$

$$\sum_{n\in\langle N\rangle} x[n]\, e^{-j\cdot r\cdot\frac{2\pi}{N}\cdot n} = \sum_{n\in\langle N\rangle}\sum_{k\in\langle N\rangle} a_k\, e^{j\cdot\frac{2\pi}{N}\cdot n\cdot(k-r)}$$

Exchange index k, n

$$= \sum_{k\in\langle N\rangle} a_k \left(\sum_{n\in\langle N\rangle} \Big(\underbrace{e^{j\cdot\frac{2\pi}{N}\cdot(k-r)}}_{\alpha}\Big)^n\right)$$

$$= \sum_{n\in\langle N\rangle} \alpha^n$$

4

$$= \sum_{n=0}^{N-1} \alpha^n$$

$$= \begin{cases} N & \text{if } \alpha = 1 \\ 0 & \text{if } \alpha \neq 1 \end{cases}$$

α will be 1

when $k-r = 0$

$k = r$

term $= N$

entire thing reduces to

$$= a_r \cdot N$$

What we were looking for to find (d)

When $\alpha \neq 1$ - $k-r \neq 0$ - some complex #

-modulus $=1$

but does ~~not~~ $\neq 1$

$$\sum_{n=0}^{N-1} \frac{(1-\alpha)\alpha^n}{(1-\alpha)} = \frac{1}{1-\alpha} \sum_{n=0}^{N-1} (\alpha^n - \alpha^{n+1})$$

$$= \frac{1}{1-\alpha} \left(1 - \alpha + \alpha - \alpha^2 + \alpha^2 - \alpha^3 + \alpha^3 , \ldots + \alpha^{N-1} - \alpha^N \right.$$

$\textcircled{5}$

Cancles so

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

$$\alpha^N = \left(e^{j\cdot l\cdot\left(\frac{2\pi}{N}\right)}\right)^N$$

$$= e^{j\cdot l(2\pi)}$$

$$= \cos(2\pi l) + j\sin(2\pi l)$$

$$= 1$$

$$x[n] = \cos\left(\frac{2\pi}{N}n\cdot r\right)$$

$0 \leq r \leq N-1$

Using first formula

$$= \frac{1}{2}e^{j\cdot\frac{2\pi}{N}\cdot r\cdot n} + \frac{1}{2}e^{j\cdot\frac{2\pi}{N}\cdot(-r)\cdot n}$$

Implies that

$$a_k = \begin{cases} \frac{1}{2} & \text{if } k=r,\ k=-r \\ 0 & \text{otherwise} \end{cases}$$

↑ cos only has this real

~~Same~~

⑥

Same for sin

$$x[n] = \sin\left(\frac{2\pi}{N} \cdot r \cdot n\right)$$

$$= \frac{1}{2j} \cdot e^{j \cdot \frac{2\pi}{N} \cdot r \cdot n} - \frac{1}{2j} e^{j \frac{2\pi}{N} \cdot (-r) \cdot n}$$

$$\hookrightarrow \quad a_k = \begin{cases} -\frac{j}{2} & \text{if } k = r \\ \frac{j}{2} & k = -r \\ 0 & \text{otherwise} \end{cases}$$

Compositions

$x[n] = x_1[n] + x_2[n]$ both period n

Super imposition

Say did FT of x_1 and x_2

∟got $(a^1_k)_{k \in \langle N \rangle}$ ∟ got $(a^2_k)_{k \in \langle N \rangle}$

So $x[n]$

- could just compute
- $(a_k)_{k \in \langle N \rangle}$

Or just add them

$$a^k = a^1_k + a^2_k$$

$$= \sum_k a^1_k e^{j\frac{2\pi}{N}\cdot k\cdot n} + \sum_k a^2_k \cdot e^{j\cdot\frac{2\pi}{N}\cdot k\cdot n}$$

$$= \sum_k \underset{\overset{\shortparallel}{a_k}}{(a^1 + a^2_k)} e^{j\frac{2\pi}{N}\cdot k\cdot n}$$

by def

Now have a toolkit for FT

Find each w/ formula

Then add them up

Other tutorial problems

#3 $x[n] = 1 + \sin\left(\frac{2\pi}{N}\cdot n\right) + 3\cdot\cos\left(\frac{2\pi}{N}\cdot n\right) + \cos\left(2\cdot\frac{2\pi}{N}\cdot n + \frac{\pi}{2}\right)$

signal fn

What FT.

8)

First - Calc F. coefficents for each term

Then -add them up

Constant Term

-exactly cos(0)

$x[n] = 3\,y[n]$

$a_k = 3\,a_k$

So write 1 as

$$\cos\left(\frac{2\pi}{N}\cdot 0 \cdot n\right)$$

So from above

$a_0 = 1$

$a_{other} = 0$ ← Technically $a_k = 0$ for all $k \neq 0$

2nd Term

F. Coefficents

$r = 1$

So $a_k = \begin{cases} -\frac{j}{2} & \text{if } k=1 \\ \frac{j}{2} & k=-1 \\ 0 & \text{otherwise} \end{cases}$

Third Term

$$a_k = \begin{cases} \overset{\swarrow \text{multiplied by 3}}{\frac{3}{2}} & k = \pm 1 \\ 0 & \text{otherwise} \end{cases}$$

Fourth Term

not standard term

So use trig to rejigger

$\cos\left(\theta + \frac{\pi}{2}\right) = -\sin\theta$

So $-\sin\left(2 \cdot \frac{2\pi}{N} \cdot n\right)$

So $$a_k = \begin{cases} +\frac{j}{2} & k=2 \\ -\frac{j}{2} & k=-2 \\ 0 & \text{otherwise} \end{cases}$$

Now add

each k $\downarrow$

$a_0 = 1 + 0 + 0 + 0 = 1$

$a_1 = 0 + -\frac{j}{2} + \frac{3}{2} + 0 = \frac{3}{2} - \frac{j}{2}$

$a_2 = 0 + 0 + 0 + \frac{j}{2}$

$a_{-1} = 0 + \frac{j}{2} + \frac{3}{2} + 0 = \frac{3}{2} + \frac{j}{2}$

$a_{-2} = 0 + 0 + 0 - \frac{j}{2} = -j/2$

a_k for $k \neq -2, -1, 0, 1, 2 = 0 + 0 + 0 + 0 = 0$

10

B, C, D pretty much same thing

E) x[n] has exactly 1 non 0 value

$$x[0] \cdots \underset{\underset{\neq 0}{\uparrow}}{x[m]} \cdots x[N-1]$$

Can you complete modulus of all F coefficents

$$a_k = \frac{1}{N} \sum_{n \in \langle N \rangle} \cdot x[n] \cdot e^{-j \cdot \frac{2\pi}{N} \cdot k \cdot n}$$

Only term that matters is when it is not 0

$$= \frac{1}{N} \cdot x[m] \cdot e^{-j \cdot \frac{2\pi}{N} \cdot k \cdot m}$$

Compute modulus of that thing

$$|a_k| = \frac{1}{N} \cdot |x[m]| \cdot \underbrace{|e^{j \cdots}|}_{= 1}$$

$$= \frac{|x[m]|}{N}$$

all ks are = modulus

Something sparse in time domain - becomes full in freq domain

---

Heisenberg uncertainity principle is essentially this

- You can not measure 2 qu exactly correctly at same time
  - moment
  - position

(11)

If you knew time exactly

- freq would be spread out

· Do 4, 5 on own.

(Don't really get, so should do)

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 PS6

Dates & Deadlines

issued:	Mar-16-2011 at 00:00
due:	Mar-31-2011 at 06:00
checkoff due:	Apr-05-2011 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.

Aloha

Problem 1.

Consider a shared medium with N backlogged nodes running the slotted Aloha MAC protocol without any backoffs. A "wasted slot" is one in which no node sends data. The fraction of time during which no node uses the medium is the "waste" of the protocol. Each packet is 1 time slot long.

A. If the sending probability is p, what is the waste?

Waste = 1 − Utilization

$U = Np(1-p)^{N-1}$

(points: 0.5)

B. Suppose N is large. If the Aloha sending probability, p, for each node is picked so as to maximize the utilization, what is the probability of a collision?

have waste $1-\frac{1}{e}$

but what is p(collision)? – don't see anywhere!

I assume it is all waste?

Or think fresh – though remember seeing

(points: 0.5) $P(\text{collission}) = 1 - P(\text{none}) - P(\text{[illegible] 1 ie sucess})$

$1-(1-p)^N - Np(1-p)^{N-1}$

then ~~fill in~~ $\lim N \to \infty$ $p = 1/N$ $= -\frac{1+e}{e} \approx .632$

Problem 2.

Alyssa and Ben are the only two users on a shared medium broadcast network running a variant of slotted Aloha. Their computers are configured such that Alyssa is 1.5 times as likely to send a packet as Ben. Assume that both computers are backlogged and that each packet is one slot long.

A. For Alyssa and Ben, what is their probability of transmission such that the utilization of their network is maximized? Please give numeric answers.

$U_{max} = \frac{1}{N}$

– but computers set wrong

~~$A = \frac{2}{3}$ $B = 1/3$~~ no not right!

(points: 0.5) $A = .6$ $B = .4$

B. What is the maximum utilization? Please give a numeric answer.

$U = Np(1-p)^{N-1}$ think about it – 6.042!

but w/ diff ps? $\underbrace{.6}_{\text{tran}}\underbrace{(1-.4)}_{\text{does not trans}} \underbrace{+}_{\text{add}} .4(1-.6) = .52$

(points: 0.5)

Problem 3.

You have two computers, A and B, sharing a wireless network in your room. The network runs the slotted Aloha protocol with equal-sized packets; each packet is 1 slot long. You want B to get twice the throughput over the wireless network as A whenever both nodes are backlogged. You configure A to send packets with probability p_a = 0.25. What should you set the transmission probability of B (p_b) to, in order to achieve your throughput goal? Assume that if exactly one node sends a packet in a time slot, it will be received successfully, but if both nodes do so, then neither packet will be received successfully.

Want B to get twice throughput
What does that mean? B is ~~more~~ sucessful more often? Just says X;
or b transmits most of the time? .75?

(points: 1) How calc throughput? sucessful transmissions over time

Problem 4. B sucessful when $p_B(1-p_a)$ So $2p_a(1-p_b) = p_b(1-p_a)$

$p_a = .25$

~~$p_a + p_b = 1$~~

no!

$.5 - .5b = .75b$

$.5 = 1.25b$

$b = .4$

Ben Bitdiddle sets up a shared medium wireless network with one access point and N client nodes. Assume that both the access point and the N client nodes are backlogged. Each of the N clients wants to send its packets to the access point; the access point's packets are destined to various clients. The network uses slotted Aloha with each packet fitting exactly in one slot. Ben sets the transmission probability, p, of each client node to 1/N and sets the transmission probability of the access point to a value p_a.

A. Determine the utilization of the network in terms of N and p_a. Hint: when N=4 and p_a=0.25, the utilization is approximately 40%.

$U = N \cdot p(1-p)^{N-1}$

$4 \cdot .25(1-.25)^{4-1} \approx 42\%$

? just access point?
— oh client $\frac{1}{N}$ ap p–a

$Np(1-p)^{N-1}(1-p_a) + p_a(1-p)^N \approx .395$

(points: 0.5)

B. What is the utilization of the network when N is large?

Lim as $n \to \infty$ of above

$= .367$

(points: 0.25)

C. Suppose N is large. What value of p_a ensures that the aggregate throughput of packets received successfully by the N clients is the same as the throughput of the packets received successfully by the access point?

— which is actually just utilization

So first find client sucess

AP sucess

$Np(1-p)^{N-1}(1-p_a) = p_a(1-p)^N$ $p = \frac{1}{N}$ solve for p_a

(points: 0.25) Complicated thing

$a = \left(\frac{N-1}{N}\right)^{-N}\left(-Na\left(\frac{N-1}{N}\right)^N + a\left(\frac{N-1}{N}\right)^N + \left(\frac{N-1}{N}\right)^N\right)$

take lim $n \to \infty$

— no result found!

? $a = \frac{1}{2}$

$\frac{1-a}{e} = \frac{a}{e}$

↑

try other side again w/ x sign

?

got $\frac{a}{e}$

↑

add • sign

?

0 = not found

Try one side at a time — but has a!

?

Try limit first

do clients no that?

From here on, only the client nodes are backlogged -- the access point has no packets to send. Each client node sends with probability p (don't assume it is 1/N).

Ben Bitdiddle comes up with a cool improvement to the receiver at the access point. If exactly one node transmits, then the receiver works as usual and is able to correctly decode the packet. If exactly two nodes transmit, he uses a method to cancel the interference caused by each packet on the other, and is (quite remarkably) able to decode both packets correctly.

C. What is the probability, P2, of *exactly* two of the N nodes transmitting in a slot? Note that we want the probability of *any* two nodes sending in a given slot.

with no AP

$N^2 \underbrace{p \cdot p}_{\text{2 tran}} (1-p)^{N-2}$

all except 2 transmit

N^2 possible combos

(points: 0.5)

? is N^2 right? – $\binom{N}{2} = \frac{1}{2}(N-1)N$ ← that might be more correct (ie correct instead)

D. What is the utilization of slotted Aloha with Ben's receiver modification? Write your answer in terms of N, p, and P2, where P2 is defined in the problem above.

am I mixing up p and U?

U= no its P1 + P2

(points: 0.5)

Problem 5.

what is this again

Media access control - guessing general term

Alyssa P. Hacker is designing a MAC protocol for a network used by people who: live on a large island, never sleep, never have guests, and are always on-line. Suppose the island's network has N nodes and the island dwellers always keep *exactly some four of these nodes backlogged*. The nodes communicate with each other by beaming their data to a satellite in the sky, which in turn broadcasts the data down. If two or more nodes transmit in the same slot, their transmissions collide; however, the satellite uplink doesn't interfere with the downlink. The nodes on the ground cannot hear each other and each node's packet transmission probability is non-zero. Alyssa uses a slotted protocol with all packets equal to one slot in length.

? just normal?

A. For the slotted Aloha protocol with a fixed per-node transmission probability, p, what is the maximum utilization of this network? (Note that although there are N nodes in all, only some four of them are constantly backlogged.)

(i just normal $\frac{1}{N} = \frac{1}{4}$
ignoring how we know which of the 4

(points: 0.5)

B. In this network, as mentioned above, four of the N nodes are constantly backlogged, but the set of backlogged nodes is not constant. Suppose Alyssa must decide between slotted Aloha with a transmission probability of 1/5 or time division multiple access (TDMA) among the N nodes. For what N does the expected utilization of this slotted Aloha protocol exceed that of TDMA?

why this

Strict TDMA

U w/ $p\ \frac{1}{5} = N \frac{1}{5}\left(1-\frac{1}{5}\right)^{N-1}$
TDMA U – ? depends how many want to transmit
N=4 does not matter – or no it does
Util $= \cancel{\#}\ 4\left(\frac{1}{N}\right) = \frac{4}{N}$

(points: 0.5)

Problem 6.

Token-passing is a variant of a TDMA MAC protocol. Here, the N nodes sharing the medium are numbered 0, 1, ..., (N-1). The token starts at node 0. A node can send a packet if, and only if, it has the token. When node *i* with the token has a packet to send, it sends the packet and then passes the token to node (*i*+1) mod N. If node *i* with the token does not have a packet to send, it passes the token to node (*i*+1) mod N. To pass the token, a node broadcasts a token packet on the channel and all other nodes hear it correctly.

A data packet occupies the channel for time T_d. A token packet occupies the channel for time T_k. If *s* of the N nodes in the network have data to send when they get the token, calculate the utilization of the channel in terms of the parameters above. Note that the bandwidth used to send tokens is pure overhead; the throughput we want corresponds to the rate at which data packets are sent.

2 10 10 1

Hint: When 20% of the nodes have data to send (i.e., *s*/N = 0.2) and T_d=10*T_k, the utilization is 2/3.

Stumper! I should really be able to figure out!

$U = \frac{T_d}{T_d + T_k}$ but how to include which nodes active?

$\cancel{\frac{1}{T_k = \left(1-\frac{s}{N}\right)}}$

Oh # to send = s
(N−s) don't send

(points: 1)

$\frac{T_d}{T_d + [N-s+1]T_k}$ $T_d = 10T_k$ $\frac{10T_k}{10T_k + (N-s+1)T_k}$

test $\frac{10}{10+9.1} = 52\%$ not clos

Introduction to this week's Python tasks.

This lab uses **WSim**, a simple packet-level network simulator for a shared medium network. You will be writing a small amount of code to develop various MAC protocols and measure how they perform under different conditions. Much of your work will be on experimenting with various parameters and explaining what you observe.

In each experiment, all the nodes run the same MAC protocol. The simulator executes a set of steps every time slot; time increments by 1 each slot.

You can run the python programs for this lab using python from the command line, e.g., `python PS6_.py`. **This lab does not work well in IDLE (you can use IDLE to edit files, but running them may not work as expected).**

To understand the different parameters one can set in WSim, go to a shell (i.e., command line prompt) and enter:

might need to go to lab

```
python PS6_1.py -h
```

This command prints out the various options; the important ones are:

1. **Retry:** If two or more nodes are actively sending a packet in the same time slot, they collide and both packets are considered lost. The `-r` option decides whether the node should *retry* the packet or not upon failure. In WSim, the feedback about whether a packet succeeded or not (i.e., collided) is instantaneous, with the sending node discovering it in the same time slot as the transmission. By default, the retry option is "off". When it is turned on, at an *offered* load of 100% (which is what we will use in this lab), the **actual** load presented to the system **exceeds** the channel's maximum rate. That is, we would expect most queues to be backlogged most of the time with these settings.

2. **Packet size:** In any given experiment, the size of a packet is fixed. It has to be an integral number of time slots in size (1 or more). To set the packet size, use the `-s` option; the default is 1. Notice that setting a large packet size (say, 10) emulates an "unslotted" network.

3. **Skew:** The -k option specifies whether the load is skewed or not. The load itself is generated according to a random process, whose details aren't important for this lab. By default, the skew is off, so all nodes generate the same load on average. When the `-k` option is set, then the total offered load, L, is divided in geometrically-spaced amounts. Node 0 presents a load of L/2, node 1 L/4, and so on. The last two nodes each present the same load, $L/2^{N-1}$, where N is the total number of nodes.

4. **Number of nodes:** The number of nodes in a run of WSim; default is 16. Set using `-n`.

5. **GUI:** The `-g` option turns on the graphical user interface, which may be of some use in debugging your code. **We recommend that you set the parameters for the simulation from the command line and NOT from the GUI, as the GUI's**

parameter setting code may not port well across different python installations.

Experimental method

WSim runs for a specified number of time slots (settable using the `-t` option, with a default of 10000) and prints out some performance numbers (and possibly displays some graphs) at the end. Of interest to us are the **utilization and fairness** numbers, which are defined in the lecture notes. The code reports a "weighted" fairness number as well, which is the fairness index calculated over the ratio of the observed throughputs to offered loads. The (unweighted) inter-node fairness is calculated over the throughputs alone, without regard to the offered load.

In this lab, you will implement the core of three MAC protocols---**TDMA, stabilized Aloha** (with backoffs), and **CSMA**---and understand their performance. Each node is an object, which has three methods that you can use to implement the core of the MAC protocol:

`boolean = node.channel_access(time,ptime,numnodes)`

> This method is called by WSim every time slot when the node has a packet waiting to be sent in its queue. This method should return `True` if the MAC protocol you're implementing would like a packet sent in the current time slot, and `False` otherwise. `time` is current time, `ptime` is the packet size in time slots, and (for TDMA) `numnodes` is the number of nodes in the network.

`node.on_collision(packet)`

> Called every time slot in which the node has experienced a collision.

`node.on_xmit_success(packet)`

> Called every time slot in which the node has successfully sent a packet (i.e., no collisions occurred during transmission).

You can modify any per-node state that you want to in these methods (e.g., the state maintained by the backoff scheme, statistics of interest, etc.). Make sure to add the code to initialize this state in the Node object's `__init__` function, whose body is included in the lab task files.

In many of our experiments, we will use the `-r` option, which cause the nodes to retry upon experiencing a collision. (Of course, the MAC protocol's `channel_access()` method will determine when the retry actually occurs.)

Python Task #1: TDMA

Useful download links:

PS6_wsim.py -- packet-level simulator
PS6_1.py -- template file for this task

Implement a simple TDMA scheme by suitably filling in the `channel_access()` method in the template file PS6_1.py. Recall that in a TDMA scheme, time is divided into `numnodes` equal-size slots, each long enough to accommodate the transmission of a single packet and that each node is allocated one of the slots to use when it has a packet to transmit. Note that a node can determine its unique node number (an integer between 0 and `numnodes-1`) by calling `self.get_id()`.

The slightly tricky part of this function is to correctly handle packet sizes that are larger than 1 time slot. We want the TDMA scheme to treat each packet as an atomic unit of transmission; when the protocol determines that a given node can send, that node should send the complete packet. Put another way, we want the effective size of a time slot in the scheme to be equal to the packet size. You will probably find it easier to first write the function and run it for a packet size of 1 slot, then modify your code to correctly handle larger packet sizes. Note that when the packet size is set to some value using the -s option, all nodes will use that value.

Run the following (after you test it for various packet sizes to ensure that there are no collisions):

- `python PS6_1.py -t 2000`
  (16 nodes, packet size = 1 slot, simulation time = 2000 slots)
- `python PS6_1.py -s 7 -t 14000`
  (16 nodes, packet size = 7 slots, simulation time = 14000 slots)
- `python PS6_1.py -k -n 20`
  (20 nodes, skewed load, packet size = 1 slot, simulation time = 10000, the default value)

When you're ready, please submit the file with your code using the field below.

File to upload for Task 1: Browse...

(points: 1)

Questions:

A. Please run the following experiments using a skewed load (`-k`) where the load offered by a node decreases with the node number, i.e., high-numbered nodes have a packet to transmit much less frequently than low-numbered nodes.

```
python PS6_1.py -k -n 10
python PS6_1.py -k -n 20
python PS6_1.py -k -n 40
python PS6_1.py -k -n 80
```

Please report the utilization from each experiment (look for `util` in the printout). With a skewed load, as one increases the number of nodes, what happens to the utilization? Why?

↓ b/c share slot

(points: 0.5)

B. What is the number of nodes at which the network utilization is smaller than 0.25 for the skewed workload? (Because each run is randomized, run it a few times to be confident of your answer.)

Grrr guess + check

(points: 0.5)

Python Task #2: Aloha with Fixed Probability of Sending

Useful download link:

PS6_2.py -- template file for this task

In this task, we will implement slotted Aloha with a fixed transmission probability and measure its performance under different conditions. The program allows you to set the transmission probability using the -p option. Looking at PS6_2.py, note that the `__init__` function of AlohaWirelessNetwork sets each node's "p" to the configured value of the transmission probability. Your task is to implement the Aloha MAC protocol by providing the correct code in `channel_access()`, so that the node will transmit packets with the configured probability. Your scheme should work when a packet is 1 slot long, but also for longer packet sizes (of course, the utilization may be different for different packet sizes).

AK **Please note: For this task as well as the subsequent ones, do not use numnodes in your code, even though it is accessible. The reason is that we want your algorithm to work for arbitrarily distributed offered loads, and using numnodes will not help achieve that.**

P given?

Test your code by running the following for different values of p and observe the resulting utilization. These tests use the `-r` option to force multiple nodes to usually be backlogged. With this option, each collision causes a retry. Because the (default) offered load is 100%, the retries ensure that the total offered load exceeds the channel rate. You can also observe this backlog if you run the following tests with the -g option and pay attention to the queue lengths at the nodes.

?what is this?
GUI

```
python PS6_2.py -n 16 -r -p value
```

When you're ready, please submit the file with your code using the field below.

File to upload for Task 2: Browse...

(points: 1)

Questions:

A. Experiment with different values of p and observe the resulting utilization. What value of `p` in the range (0,1) maximizes the utilization in your experiments? What is the maximum utilization?

1/16

(points: 0.33)

B. We will now explore a skewed load to see whether that affects the best choice of `p`. Run the following for a few values of `p`.

```
python PS6_2.py -n 16 -r -k -p value
```

What value(s) of `p` in the range (0,1) maximizes the utilization in your experiments in this case? What is the maximum utilization?

.0625 = .25 .2 = .40 .23 = .4
.1 = .32 .25 = .39 .21 = .41
.15 < .37 .22 = .41

(points: 0.33)

C. How do the optimal values of `p` and the corresponding utilization compare with the no-skew case? Why are they different?

(points: 0.34)

Python Task #3: Stabilizing Aloha (with Backoff)

Useful download link:

PS6_3.py -- template file for this task

In this task, we will develop a stablization method for Aloha using randomized backoffs to replace the fixed probability of Task #2. Our goal is to adaptively select the transmission attempt probability, p, used in the `channel_access` method. To do that, write your code in PS6_3.py to adjust p in the `on_collision` and `on_xmit_success` methods, which are called when a packet transmission fails and succeeds, respectively.

We will use two parameters, `pmax` and `pmin`. These correspond to the maximum and minimum values of the transmission attempt probability, p. The values of these parameters can be set from the command line when you run the program, and are available as `self.network.pmax` and `self.network.pmin` respectively (see the `__init__` function of AlohaWirelessNetwork). In your code, ensure that `pmin` ≤ `p` ≤ `pmax`.

You can use any algorithm you want to set p in these functions, including the ones discussed in lecture. Good schemes achieve high utilization, but ensure also that fairness is high (as close to 1 as possible -- lower than 0.9 is a sign that there is significant unfairness when the number of nodes is between 8 and 16) and avoid the capture effect. There is no absolute correct answer (though there are bad methods!), so feel free to be creative if you think you have a good idea. Note that you are not allowed to use the number of backlogged nodes in your scheme, because that information would not be available in practice.

p changes & have to find

Run your code as follows for a few different settings of pmin and pmax and observe the utilization and inter-node fairness values.

```
python PS6_3.py -r -n 8 --pmin=value --pmax=value
```

(Note the two dashes in front of the pmax and pmin options.)

lecture notes multiplicative but still U = .4 which is what I got

When you're ready, please submit the file with your code using the field below.

File to upload for Task 3: Browse...

(points: 1)

Questions:

A. Run `python PS6_3.py -r -n 8 --pmin=0 --pmax=1`. Would you recommend running a real network with these parameters for `pmin` and `pmax`? Briefly explain your answer.

No - some hit 1 + dominate

(points: 0.5)

B. Set `pmin` to 0.01 and pick `pmax` so that the fairness is as large as possible when the number of nodes is 8 and there is no load skew. What is the utilization of your protocol when the packet size is 10 slots? How does it compare to the utilization when the packet size is 1?

For the first case (packet size of 10), run

```
python PS6_3.py -s 10 -t 70000 -r -n 8 --pmin=0.01 --pmax=value
```

For the second case (packet size of 1), run

```
python PS6_3.py -r -n 8 --pmin=0.01 --pmax=value
```

Additive

~~max = .2 → U = .39 D=.01~~

~~.2 .39 Δ=.01~~ Same - wrong

(points: 0.5) M

Python Task #4: CSMA

Multi .8 max U=.55 f=.75

.8 max U=.56 U=.77

Useful download link:

PS6_4.py -- template file for this task

In this task, we will try to get the best utilization and fairness we can for a MAC protocol that uses CSMA. To check if the channel is idle, you can use the `self.network.channel_idle()` from inside `channel_access()`. This function returns `True` if there is no on-going transmission in the **current time slot**, and `False` otherwise.

Every carrier sensing mechanism has a *detection time*, defined as the time interval between the ending of a previous transmission and the detection of the channel as "idle" by a node. For the purposes of this lab, we will assume that the detection time is 0. Hence, a node can sense that the carrier is idle in the immediate next slot after the termination of the previous transmission, when it does the check for whether the channel is busy. However, it is still possible for collisions to occur, for multiple nodes could simultaneously sense the channel at the beginning of a slot, and conclude that the channel is "idle", and possibly attempt a transmission in that time slot (or in the same future time slot).

Write your code PS6_4.py for the `channel_access()` method assuming that the node has the ability to sense the carrier. Obviously, you should fill in the steps for the `on_collision()` and `on_xmit_success()` methods as well.

Test your code as follows. The `-s` option is important because it causes the packets to be longer than 1 slot, allowing a node to sense whether another transmission is in progress during a time slot.

,005 ,8 6 lecture

```
python PS6_4.py -r -n 8 -s 10 -t 100000 --pmin=value --pmax=value
```

Note that you should run the above for 100000 time slots, because we have scaled up the packet size to 10 (from 1).

When you're ready, please submit the file with your code using the field below.

File to upload for Task 4: Browse...

(points: 1)

Questions:

A. What is the utilization and fairness of your protocol when `pmin` = 0 and `pmax` = 1?

U=.79
f=.92

(points: 0.5)

B. How would you set `pmin` and `pmax` in your protocol to make the fairness number be over 0.95 consistently while still maintaining good utilization?

used in lecture?

(points: 0.5)

Python Task #5: Practical CSMA, as in WiFi (802.11) and Ethernet (802.3)

Useful download link:

PS6_5.py -- template file for this task

The ALOHA and CSMA schemes in the previous tasks pick a probability `p` for transmitting a packet, and adapt `p` to stabilize the protocol. The advantage of this method is that it is easy to analyze. In practice, however, real-world CSMA protocols like the popular 802.11 WiFi standard and the 802.3 Ethernet standard implement something a bit different, as explained below. Your task will be to write the code for the key parts of this scheme.

Rather than decide whether any given slot should have a transmission with probability `p`, each node maintains a **contention window**, which we denote by `cw`. `cw` is initially set to

; we should do token

cwmin, which is a small positive integer (say, 1). Denote the current time slot by C. If the sender is backlogged, it picks a random integer t in [1, cw] and decides to send a packet in time slot C + t.

wait some random time [1, cw]

Of course, with carrier sense in place, the sender should only send a packet if the channel is idle in time slot C + t. So, the sender senses the carrier in that time slot, and then sends a packet only if the channel is idle then. If the channel is not idle, it waits until the channel is idle, and then sends the packet.

Whenever a collision occurs, the node doubles cw, but makes sure cw never exceeds cwmax (say, 512). Whenever a transmission is successful, the node might reset cw to cwmin, or might halve its current value of cw. You will note that this scheme is similar in spirit to the probabilistic transmission scheme from Task #4, but a crucial difference is that here each backlogged node is **guaranteed** to send a packet within a finite time, unlike in the probabilistic case where there is always a small probability that the node cannot send within any given number of time slots. In mathematical terms, the probability distribution that governs whether a node transmits a packet in a given time slot is uniform in this scheme (Task #5), while it is geometrically distributed in the previous case (Task #4).

You will first implement the scheme described thus far. It will turn out not to do as well as we would like, and in Task 5.2, you will fix an important weakness.

Implement this scheme by writing the appropriate code for channel_access(), on_collision(), and on_xmit_success() in PS6_5.py. How well does it work? To answer this question, measure the utilization and fairness by running

```
python PS6_5.py -r -s 10 -n 16 -t 100000  -W 256
```

The -W option sets the maximum contention window size. The minimum contention window is 1 (you can change it using the -w option if you like (lower case "w"). Running the above, you will note that even with carrier sense being used, the utilization is quite a bit lower than in Task 4.

A. Report the utilization and fairness. Briefly explain why the utilization is low. Hint: Think about what happens if more than one node is backlogged and waiting for an on-going transmission to complete; what happens when the on-going one finishes?

U=.78
f=.93

(points: 0.5)

To fix this problem, each node needs to ignore the time slots when other nodes are transmitting data. That is, if a node picks a time slot t in [1, cw] to transmit, it should wait for that many idle slots before attempting its own transmission. Of course, before transmitting data, it should ensure that the channel is idle.

Modify your code to include the above suggestion and run the same command as before:

```
python PS6_5.py -r -s 10 -n 8 -t 100000  -W 256
```

B. Report the utilization and fairness for your scheme after including the suggestion and running the same command as before.

U=.83
f=.~~[illegible]~~ 92

(points: 0.5)

Please upload the final version of your Task #5 code:

File to upload for Task 5: 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 ✓ 8:41 PM

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

6 cont

$$N \cdot \frac{T_d}{T_d + [N-s+1]\,T_h}$$

↑

for each to transmit

- but only $\frac{s}{N}$ want to transmit

$$\frac{s}{N} \cdot N = s$$

$$\frac{s \cdot T_d}{T_d + [N-s+1]\,T_h}$$

$\frac{2 \cdot 10}{19}$ No!

S in denom?

$\frac{20}{29} = \frac{2}{3}$!

$$\frac{sT_d}{sT_d + [N-s+1]\,T_h}$$

Glad to figure it out!

3/30

PS 6_1.py TDMA

- fill in channel_acess()
- node gets # self.getId()
- So t mod Id == 1 -transmit
  when

- Have functions > 1 time slot
  - So is that given by s?
  - I guess
- What is skew?
  - unequal offered loaded
  - So not back logged

- Oh simulator runs for each node
- Instructions says use numnodes
  do time % numnodes == id
  b/c above way had modulo by 0
  - better
- do you have to implement on_colllision
  on_xmit_sucess?

Look at error lst

Everything is failling

Oh did not do == Id

Sceeded!

Now multy thing lenght

- will always be thing
- So take time % p time == 1 - Then try to trans
- Works for 7
- But fails for 1

Do time % ptime == 0

Finished 25 min

- slow, careful, relaxing in lab

(3)

#2 Slotted Aloha

- transmits probabilistic
- just generates p r#
- how to generate random #?
- decimal
- say draw decimal [0,1]
    if < p - ~~a~~ fire
- Worked!
    - 15 min
    ~~m~~ - 2 lines of code!
- ~~$~~ Sim GUI time un defined
- I should look Plazzza
- Just forget GUI
- done - Whole thing 30 min

(4)

#3 Need to fix 2 for multi length

- Seems to work pretty good

I wish it showed ending p

My system is always hitting wall - wrong way?

- no util=.48

#4 CSMA

? What is this again?

Sense

Self. network. channel_idle()

- True if current time slot

detection time = 0 here

- but multiple can still sense and hit

So send if idle

or 1 after idle

Just send if idle

Oh no still have to do p and pmin, max

⑤

No need to check time % ptime

- ↑ U from .4 to .6
- if collission restart next slot trying, not waiting time

do max

- put at cap - don't skip

do multiplicatiative

- fairness when up much faster .6 → .8
- and util .4 → .55

Since who is backlogged changes over time

#5 still sense, stable

where is cw?