

4/20

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 PS8

#### Dates & Deadlines

issued: Apr-13-2011 at 00:00

due: Apr-21-2011 at 06:00

checkoff due: Apr-26-2011 at 06:00

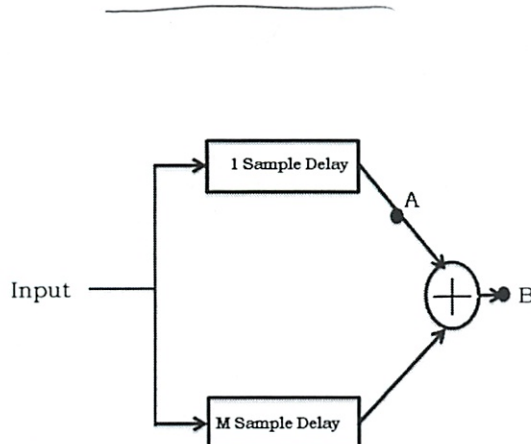
Help is available from the staff in the 6.02 lab (38-530) during lab hours -- for the staffing schedule please see the [Lab Hours](#) page on the course website. We recommend coming to the lab if you want help debugging your code.

For other questions, please try the 6.02 on-line Q&A forum at [Piazza](#).

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 the multiple delay system diagrammed below.

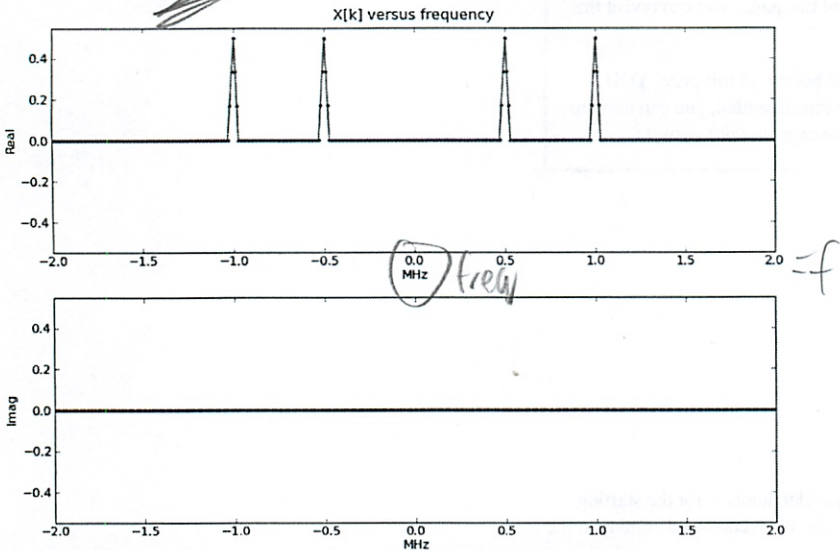


more modulated

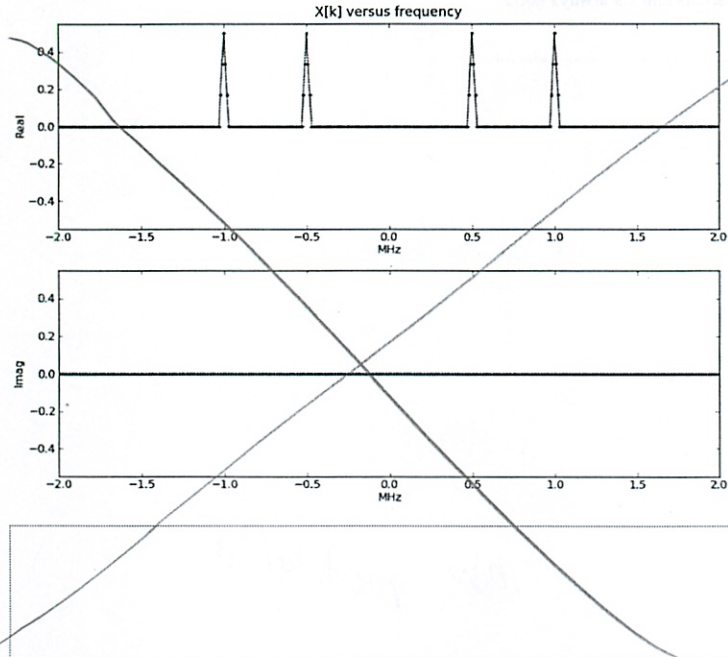
The input to the multiple delay system is a modulated signal that is periodic with period  $N = 400$ . In the plot below, the Fourier Series coefficients for this modulated signal are plotted versus frequency. The peak values in the plot are all 0.5, and the frequencies in the plot range from -2 megahertz to 2 megahertz (corresponds to a system with a sampling rate of 4 million samples per second).

Input

Fourier Coeff



A. Below are plots of the real and imaginary parts of the Fourier coefficients for point A in the multiple delay system. Determine the numerical values for the peaks at 0.5 and 1.0 megahertz.



Correction chart changed

(points: 0.5)

B. Use the following plot of the Fourier series coefficients for the sum of the delayed signals, point B in the multiple delay diagram, to determine the smallest integer value for M, the number of samples in the second delay.

how to do?  
 must add to 1?  
 or math formula

$f \rightarrow a_k \rightarrow \frac{2\pi k}{F_s} = 2\pi \frac{k_2}{N}$   
 correct

$$Y[n] = \frac{1}{2} \sum_{k=-k_1}^{k_1} a_k e^{j(k-k_c) \frac{2\pi}{N} n}$$

$k_c = 15$  and  $1$

but don't want



#1. in what at On Own all  $\pi$

$$A, 5 \quad \text{---} \quad 15 + \frac{1}{3}i$$

$$1 \quad \text{---} \quad 0 + \frac{2}{3}i$$

↑  
cos R    sin R

? multiply

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

So ~~15~~ is actually 1 or not

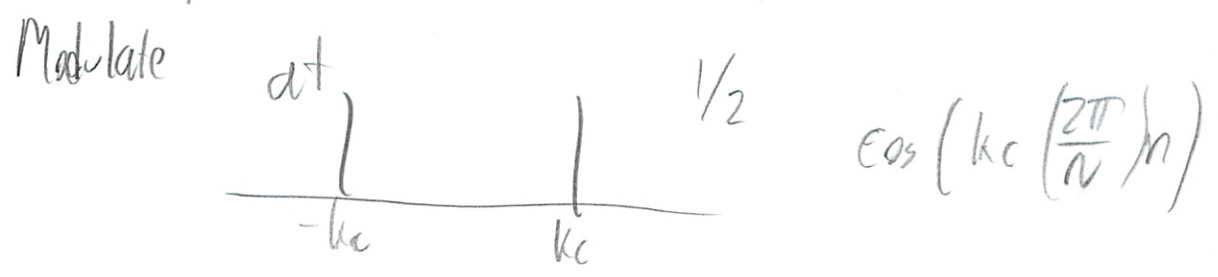
So  $\cos R = \frac{1}{2}$     roughly  $\frac{\pi}{3}$     also  $\frac{1}{2}$

$\sin R = \frac{2}{3} \approx \frac{1}{3}$      $\frac{\sqrt{2}}{2}$

$\cos R = 0$     is this  $\frac{\pi}{2}$   $90^\circ$     0

$\sin R = \frac{2}{3}$     1

but why is R diff - is that ok.



②

Read notes

So  $\cos(2\pi f_1 t)$   $\cdot$   $\cos(2\pi f_2 t)$

baseband

? modulation

QoS  $\frac{1}{t} \cdot \text{---} = 0 + \frac{2}{3}$

15  $\cdot \text{---} = 15 + \frac{1}{3}$

? something

Where this is 1 or 0 depending

$\text{---} = \cos(2\pi f_2 t)$

? ignore?

So  ~~$\cos^{-1}(2\pi f_2) = \text{---}$~~

Or new freq =  $f_1 + f_2$  and  $f_1 - f_2$

? but what?

$\cos^{-1}(\text{---}) = 2\pi f_2$

$\frac{\cos^{-1}(\text{---})}{2\pi}$

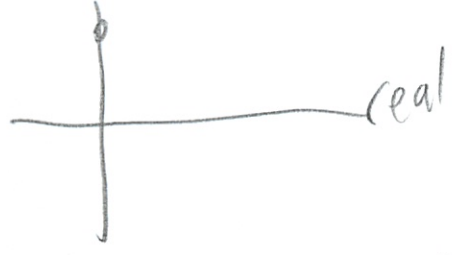
When  $\frac{\pi}{2}$   
 $\hookrightarrow f_2 = 90^\circ$

When  $\frac{\pi}{3}$   
 $f_2 = 90^\circ$

So good



③ imag



? So how to find the values?

Give up + go to lab

1. Not modulation of  $\omega$  - just phase shift  
- Just a delay

$$x[n] =$$

↑ at each time step, a real #

$$= \cos\left(2\pi \frac{kn}{N}\right) \begin{array}{l} \text{freq of signal} \\ \text{spectral coefficients} \end{array}$$

$k$  is int s.t.  $-\frac{N}{2} \leq k \leq \frac{N}{2}$

↑ just care 1 timestep

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

how much  
energy  
at freq  
 $\frac{2\pi k}{N}$

w/ cos - energy is all at 1 freq  
- only non 0 at freq

Only 2 freq where non 0

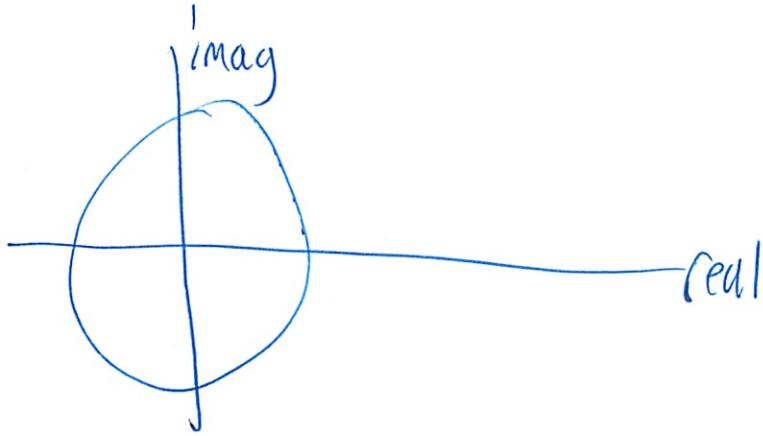




③

This is  $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$  if doing modulation

So phase factor is complex - has imag



$e^{jd}$   
? goes around circle

Just plug into calc  
 $e^{-j \frac{2\pi k n}{N}}$

- have  $k, N$



4 his notes from 1-3

if  $x[n] = \cos\left(\frac{2\pi k'n}{N}\right)$  |  $k$  is int s.t.  ~~$-\frac{N}{2} < k < \frac{N}{2}$~~

$-\frac{N}{2} \leq k < \frac{N}{2}$

Then  $a_k = \begin{cases} \frac{1}{2} & \text{if } k = \pm k' \\ 0 & \text{o.w.} \end{cases}$

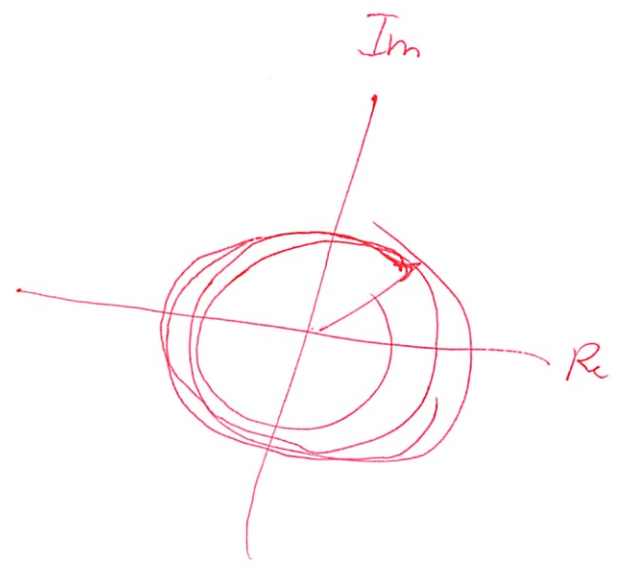
at  $f = \frac{2\pi k}{N}$

$\cos\left(\frac{2\pi kn}{N}\right) = \cos\left(\frac{2\pi f}{f_s} \cdot n\right)$

$x[n-1]$

$\cos\left(\frac{2\pi k(n-1)}{N}\right) = \frac{1}{2} e^{j \frac{2\pi k(n-1)}{N}} + \frac{1}{2} e^{-j \frac{2\pi k(n-1)}{N}}$   
 $= \frac{1}{2} \underbrace{e^{-j \frac{2\pi k}{N}}}_{\text{circle}} e^{j \frac{2\pi kn}{N}}$

$e^{j\alpha}$



#1 So  $\frac{k}{N} = f \leftarrow .5 \text{ and } 1$   
 $\frac{N}{7400} f_s \approx 4 \text{ MHz}$

for .5

$$\frac{k}{400} = \frac{.5 \text{ MHz}}{4 \text{ MHz}}$$

Can cancel units

$$\frac{k}{400} = \frac{.5}{4}$$

$$200 = 4k$$

$$k = \cancel{400} 50$$

1

$$\frac{k}{400} = \frac{1}{4}$$

$$400 = 4k$$

$$k = 100$$

So now plug in

$$e^{-j \frac{2\pi 50}{400}}$$

$$e^{-\frac{\pi j}{4}}$$

b/c this is the phase shift!

Or have to plug into whole thing - n also  
 - no n is just 1 - can ignore?

$$e^{-\frac{\pi j}{4}} + e^{\frac{\pi j}{4}}$$



① What is the form of ans we want?

$\cos(\Omega)$ ?

Or I should give real + complex forms

Oh - WA was using  $j$  as variable, not imag

5 Its  $\sqrt{2}$

- but should also be imag

well  $\frac{1}{2}e + \frac{1}{2}e$

That is  $.707 + 0j$

Why no imag component?

$$\frac{1}{\sqrt{2}} e^{-j \frac{2\pi}{400} \cdot 100} + e^{j \frac{2\pi}{400} \cdot 100}$$

gives 0

Oh right its  $e + e^j$

---

No its

$$\cos\left(-\frac{2\pi}{400} \cdot 100\right) + \sin\left(-\frac{2\pi}{400} \cdot 100\right) j$$

(3) So

$$\frac{1}{2} \left[ \cos\left(-\frac{2\pi}{400} \cdot 100\right) + \sin\left(-\frac{2\pi}{400} \cdot 100\right)j \right] + \frac{1}{2} \left[ \cos\left(\frac{2\pi}{400} \cdot 100\right) + \sin\left(\frac{2\pi}{400} \cdot 100\right)j \right]$$

$$\frac{2\pi}{400} = \frac{\pi}{200}$$

$\frac{\pi}{2}$

$$\frac{1}{2} [0 + -1j] + \frac{1}{2} [0 + 1j]$$

$$-\frac{1}{2}j + \frac{1}{2}j$$

0

So got 0 again

15

$$\frac{\pi}{4}$$
$$\frac{1}{2} \left[ \cos\left(-\frac{\pi}{4}\right) + \sin\left(-\frac{\pi}{4}\right)j \right] + \frac{1}{2} \left[ \cos\left(\frac{\pi}{4}\right) + \sin\left(\frac{\pi}{4}\right)j \right]$$

$$\frac{1}{2} \left[ \frac{\sqrt{2}}{2} + -\frac{\sqrt{2}}{2}j \right] + \frac{1}{2} \left[ \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}j \right]$$

$$\frac{\sqrt{2}}{2} - 0j$$

is supposed to be  $\ominus$

but that's not really right! according to picture

Unless you forget the  $\frac{1}{2}$

Since we just want phase shift, not more!

④

On right + left



- don't go back to cos term

(5)

B) Now B for delays

The  $\sqrt{.5}$  fr The  $.5$  freq killed out

When - is it 0

sin and delay kill it  
cos

If know freq can do it

Can figure out phase

Or just go backward in time

for .5

$$0 + 0i = \cos\left(-\frac{2\pi}{400} \cdot 50 \cdot t\right) + \sin\left(-\frac{2\pi}{400} \cdot 50 \cdot t\right)i$$

$$0 + 0i = \cos\left(\frac{\pi}{4} \cdot t\right) + \sin\left(\frac{\pi}{4} \cdot t\right)i$$

$$0 = \cos\left(-\frac{\pi}{4} \cdot t\right) \quad \rightsquigarrow \quad 0 = \sin\left(-\frac{\pi}{4} \cdot t\right)$$

$t$ s correspond

$$t=2$$

$$t=4$$

? but  $t$ s must = 1

Or are  $t$ s (+) not (x)  
↑ no are



6

For 1 MHz

↓ should really be - (previous one too)

$$0 - 1i = \cos\left(\frac{\pi}{2} t\right) + \sin\left(-\frac{\pi}{2} t\right) i$$

$$0 = \cos\left(\frac{\pi}{2} t\right) \quad -1 = \sin\left(-\frac{\pi}{2} t\right) i$$

$$t = 1, 3, 5, 7$$

$$t = \cancel{1, 3, 5, 7}, 1, 5, 9, 13$$


Try multiple  $t$ s

So try multiple  $t$ s for .5

$$t = 2, \cancel{4}, 6, 10, 14$$

$$t \text{ is } -\frac{2\pi}{4} \quad -\frac{6\pi}{4}$$

good at

$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$-\frac{3\pi}{2}$
	$\frac{3\pi}{2}$	$\frac{\pi}{2}$
$\frac{3}{2}$		

is 0 at



$$t = 4, 8, 12, 16$$

BA will never line up

$$1, 3, 5, 7, 9, \quad +2$$

$$1, 5, 9, 13, \quad +4$$

$$2, 6, 10, 14, \quad +4$$

$$4, 8, 12, 16, \quad +4$$

7

Don't need to match up

Just will .5

What he would do

Chris wrote both are cos

$$x[n] = \cos\left(\frac{2\pi 50}{400} n\right) + \sin\left(\frac{2\pi 50}{400} n\right)$$

Add delay 1 sec

2 peaks

$$x[n-1] = \cos\left(\frac{2\pi 50}{400} (n-1)\right) + \sin\left(\frac{2\pi 50}{400} (n-1)\right)$$

Oh its addition of another system!

$$x[n-1] = \frac{1}{2} e^{+j\frac{\pi}{4} n} e^{-j\frac{\pi}{4}} + \frac{1}{2} e^{-j\frac{\pi}{4} n} e^{j\frac{\pi}{4}} + \frac{1}{2} e^{-j\frac{\pi}{2}} e^{j\frac{\pi}{2} n} + \frac{1}{2} e^{j\frac{\pi}{2}} e^{-j\frac{\pi}{2} n}$$

our 4 peaks  
right outer

left outer

Missed  $\frac{1}{2}$

add to ans a



9

think it through

$$e^{j \frac{5\pi}{4}} = e^{j \frac{11\pi}{4}}$$

$$D = 5$$

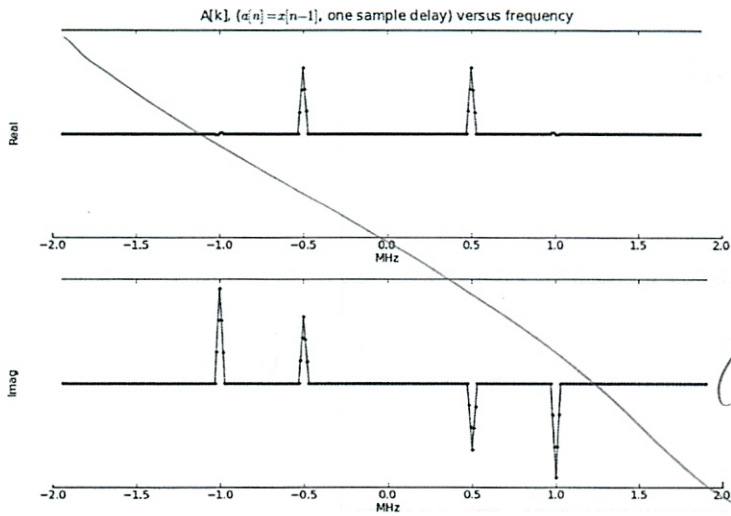
TA: ~~Don't~~ keep in complex exponentials!

$e^j$  corresponds to mapping

Stay there!

---

Or 1 sample delay is clock



the output  
is modulated

Correction on pic  
as well

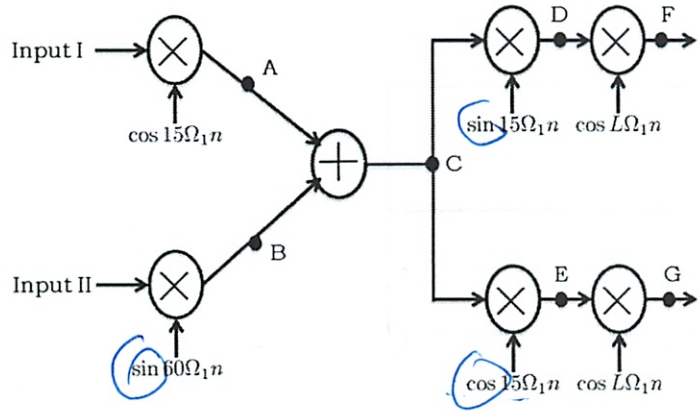
think in terms of cos  
- how long is 1 sec delay  
this is  $\frac{T}{2}$  - but how many sec?  
Or they cancel on output  
- is there an animation of this?



(points: 0.5)

**Problem 2.**

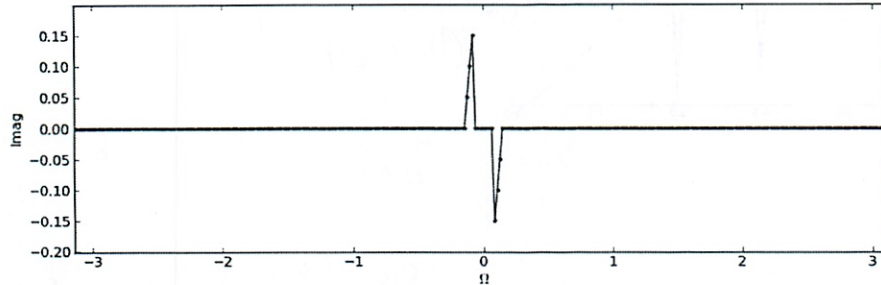
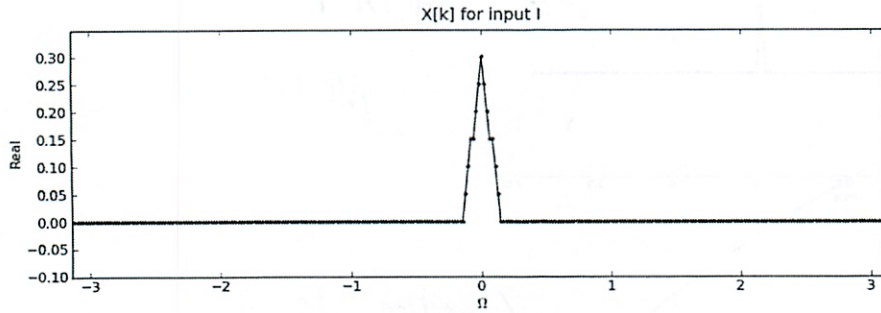
Consider the two-input modulation-demodulation system diagrammed below. The frequencies of the modulating sines and cosines are all integer multiples of  $\Omega_1 = 2\pi/300$ . Note that  $L$  in the diagram is an integer whose value is greater than 15.



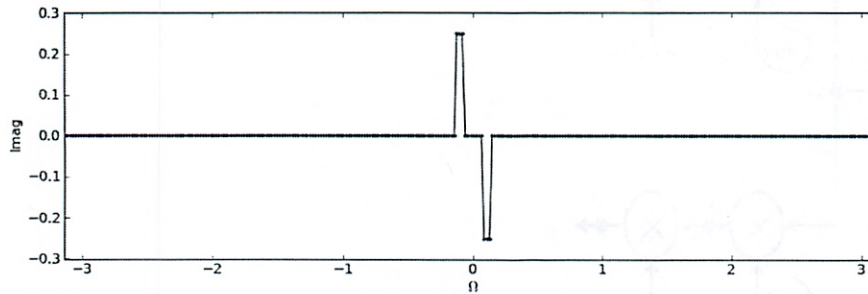
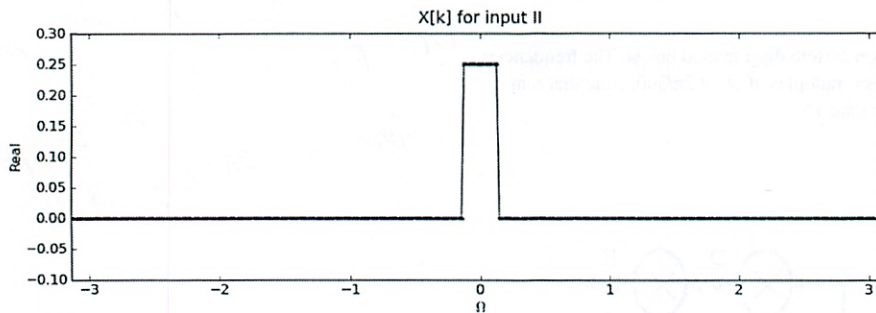
The two inputs to the modulation-demodulation system are periodic with period  $N=300$  samples -- their Fourier coefficients are plotted below. Note that the Fourier coefficients are nonzero only for  $-6 \leq k \leq 6$  or equivalently, for frequencies between  $-6\Omega_1$  and  $6\Omega_1$ .



input 1

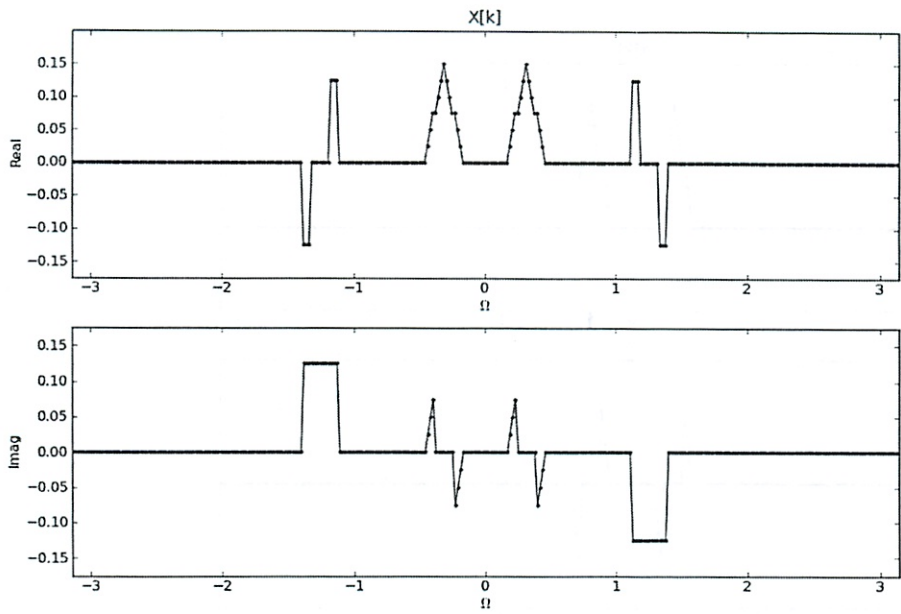


input 2



A. For each of the following Fourier coefficient plots, determine which signal (A-G) on the modulation-demodulation system diagram corresponds to the plot.

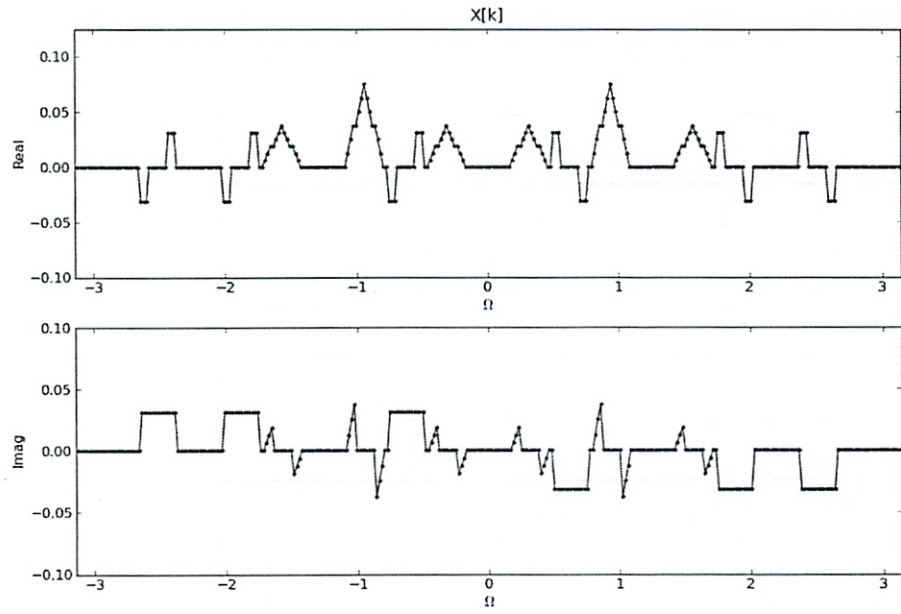
So how do you tell



Why flipped?  
sin!

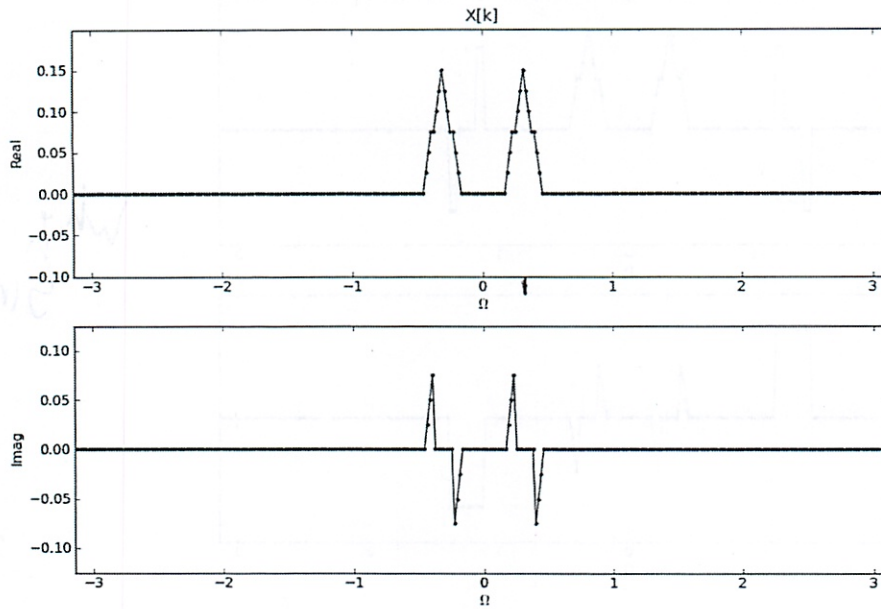
C

(points: .1)



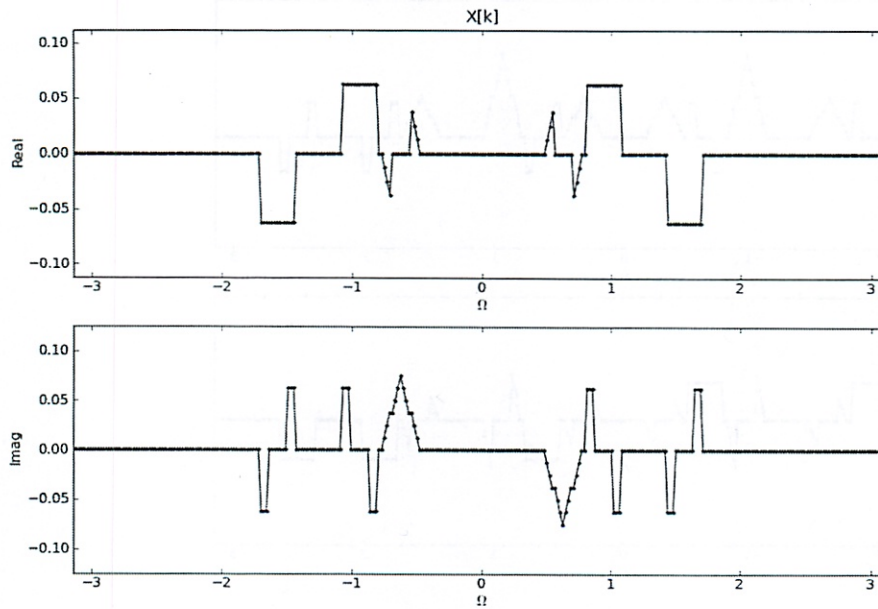
G

(points: .1)



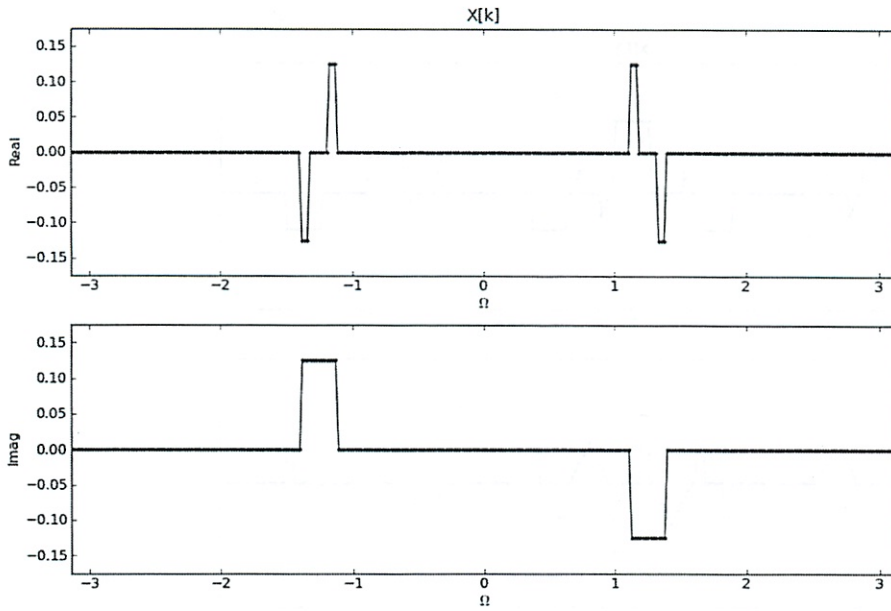
A

(points: .1)



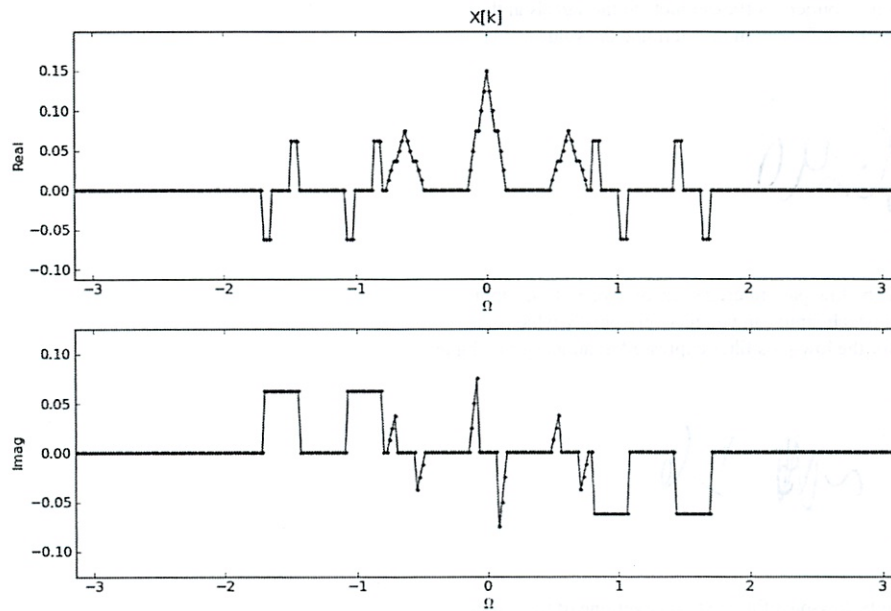
D

(points: .1)



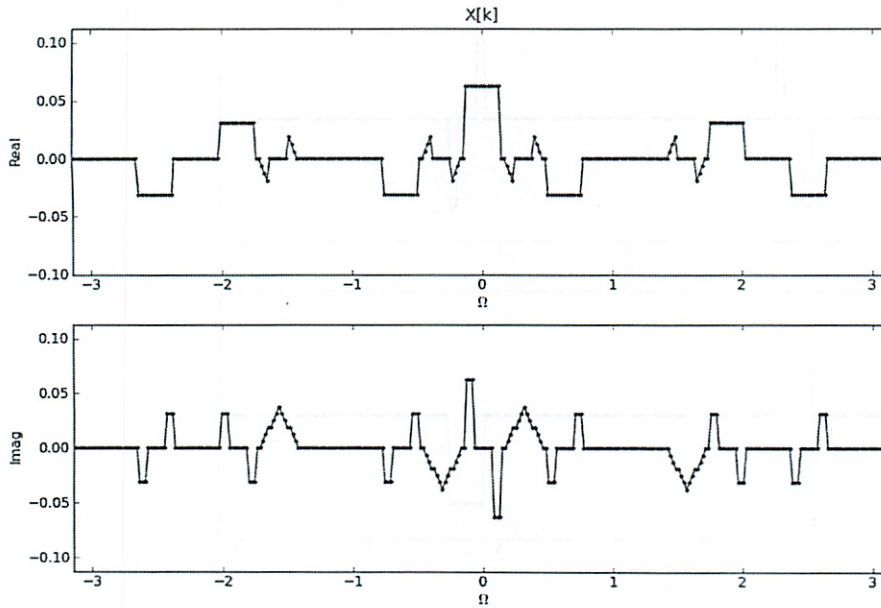
B

(points: .1)



E ✓

(points: .1)



F

(points: .1)

B. Once you have correctly matched the Fourier coefficient plots to the signals in the modulator-demodulator, please use that information to determine the value of the integer  $L$  in the diagram.

$L = 40$

(points: .3)

C. One of the two signals, E or G, can be low-pass filtered to recover one of the inputs. Which signal should you filter and which input can you recover? Finally what is the lowest possible cutoff frequency for the low-pass filter expressed as an integer multiple of  $\Omega_1$ ?

$L \sim \pm 6$

(points: .5)

D. One of the two signals, D or F, can be low-pass filtered to recover one of the inputs. Which signal should you filter and which input can you recover? Finally, what is the lowest possible cutoff frequency for the low-pass filter expressed as an integer multiple of  $\Omega_1$  (just enter the multiple)?

$L \sim \pm 6$

(points: .5)



In Lab

4/22

#2

So freq <sup>freq</sup> coefficients  $a_k$  <sup>no freq</sup>  
 $\cos\left(\frac{2\pi}{300} k\right)$   $\frac{k}{N} = \frac{f}{f_s}$

So 15 is the  $k$  for that one  
 $L$  is some unknown one  $> 15$

Only care  $-6 < k < 6$  so this means  $f_s = 12 \text{ mHz}$   
-learned from last prob

$x[n]$  are inputted

- can find from chart

- at 0  $\frac{k_n}{300} = \frac{0}{12}$

$$k_n = 0$$

- but how to read peaks?

- Or does that level of detail not matter here?

- just going visually

②

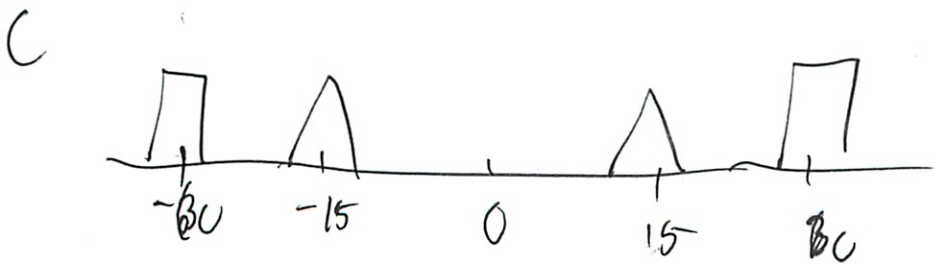
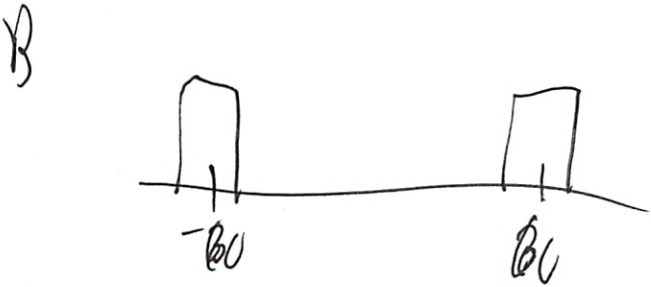
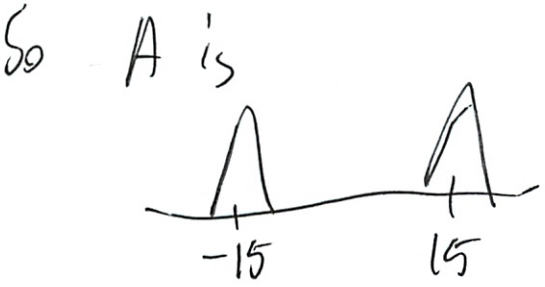
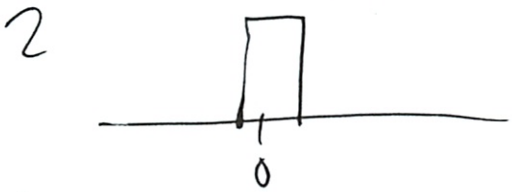
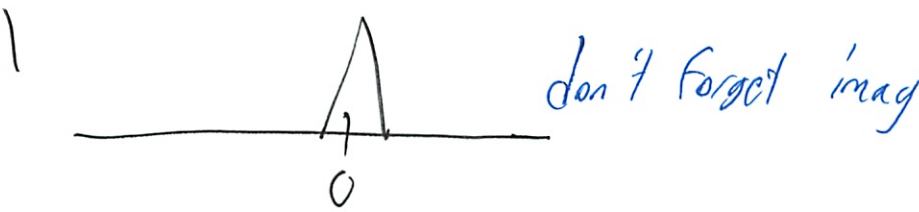
Where does the 15 fit in?

those are phase shifts?

↑ k<sub>c</sub> the target freq shift

Remember prev qv no modulation

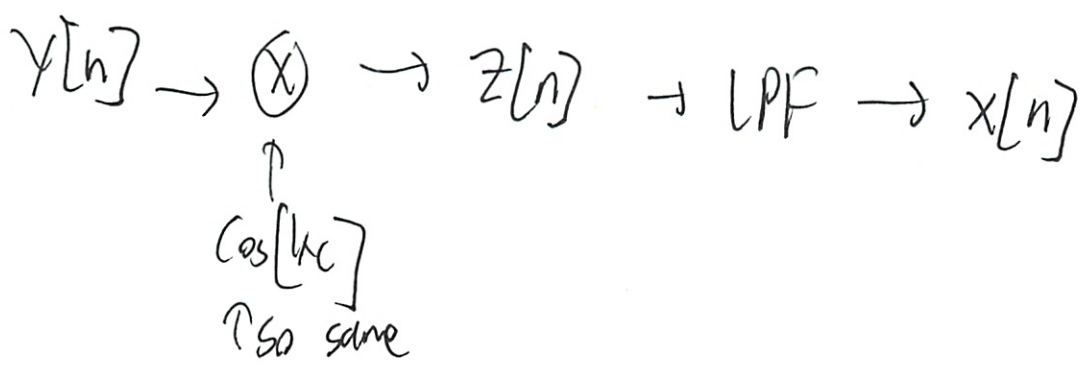
~~but~~



③ But when is real/imag flipped?

(It seems I am doing this by guessing - not understanding)

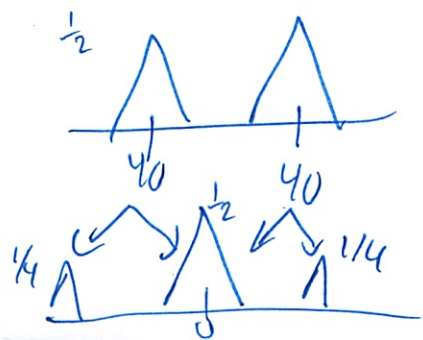
What do the  $\otimes$  do?  
 $\uparrow$  mod or demod



$D = E$  ? No sin/cos  
 - same thing?

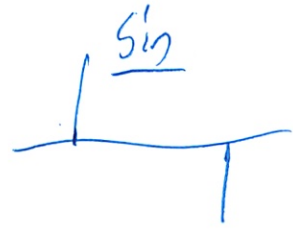
So flipped b/c sin  
 $\hookrightarrow$  flips real, imag  
 $\curvearrowright$   
 just switch

A if same - adds peak in middle demodulation they split



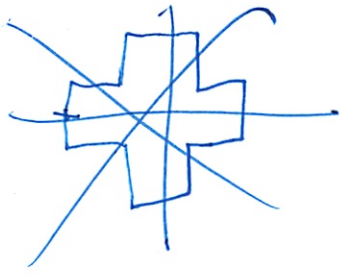
④

If ~~the~~ different - see slide!



demod, mod kinda same thing  
 demod then needs LPF

you don't get

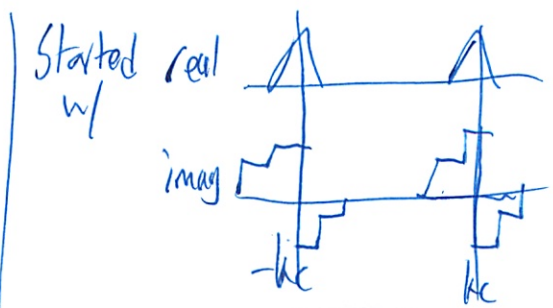
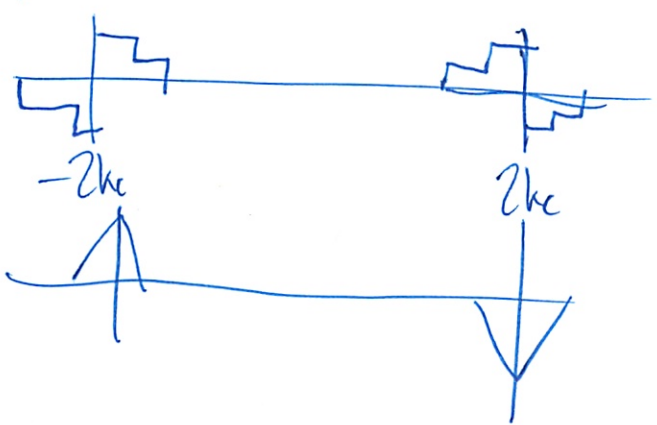


The low freq components are gone become 0

since add ↻ at each step →

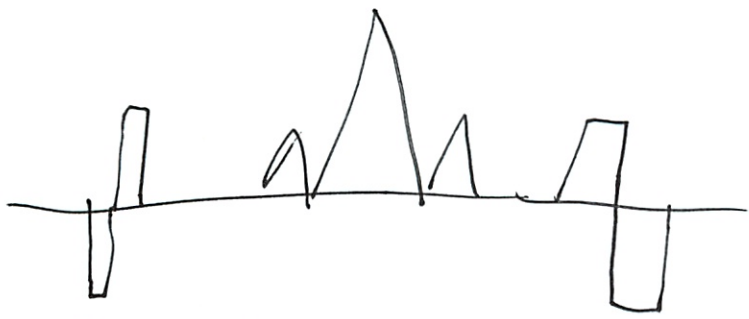
so left w/

real  
imag



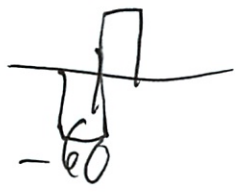
5

So then  $E_1$  is

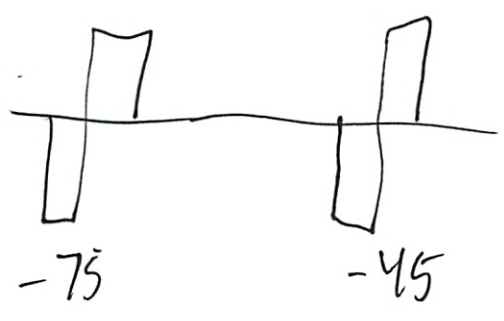


But why does this appear to split

Also modulated on 15



modulated 15



Yes

So  $D$  is

LF ~~imp~~ no center part



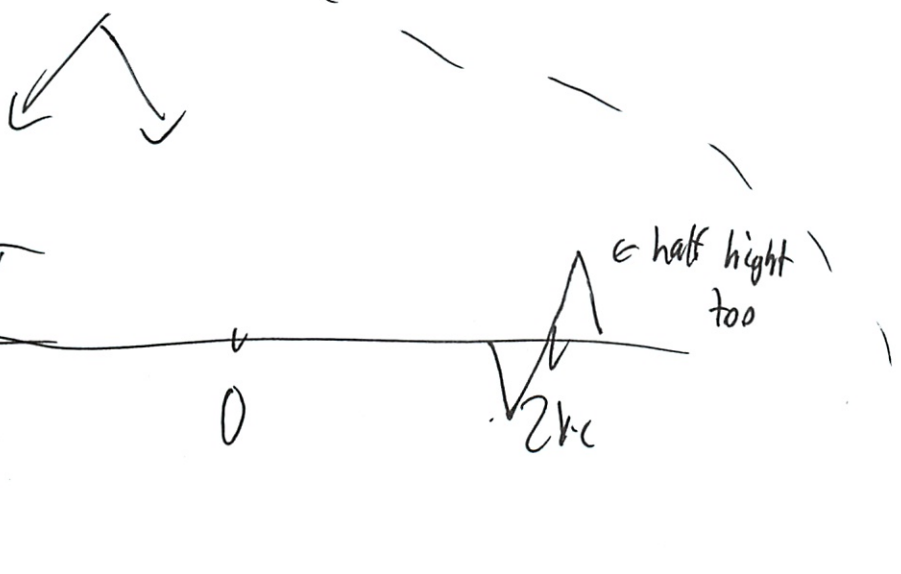
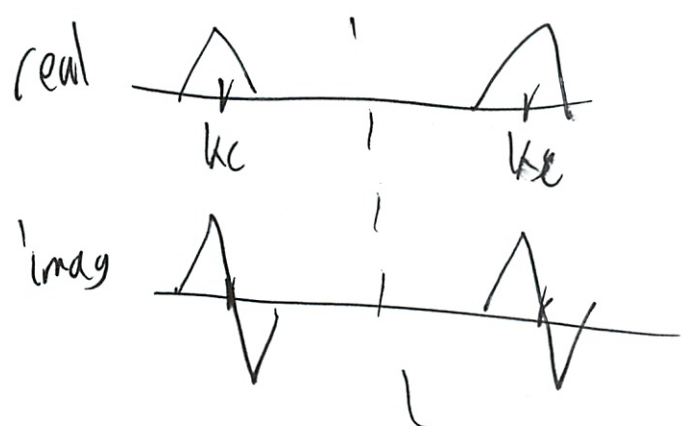
flip + modulate



6

Why ~~the~~ is part stretched - not replicated

- Oh that's sin



so that is right

⑦

Now F

Take D and cos ~~L~~

get stuff in middle

G remainder

~~flip of E~~

No its cos from E

so split more

B. Now need to find L

Do visually to get to 0

D - ~~was~~ from -1 to middle

- slightly more than 60  
less

40?

C) which to filter

E since in middle

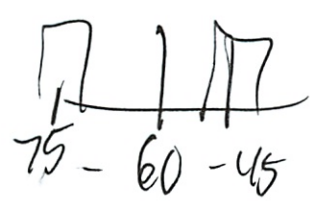
0 or (F)

25-10

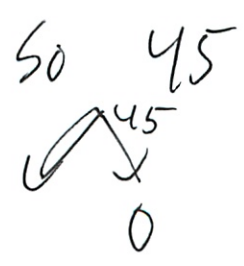
I think my visual estimation is bad  
How to do better

Individual signals limited to 6  
so cutoffs  $\pm 6$

So we got to ~~60~~ 6  
by 60  
then ~~15~~ to 0



then 45 over to 0

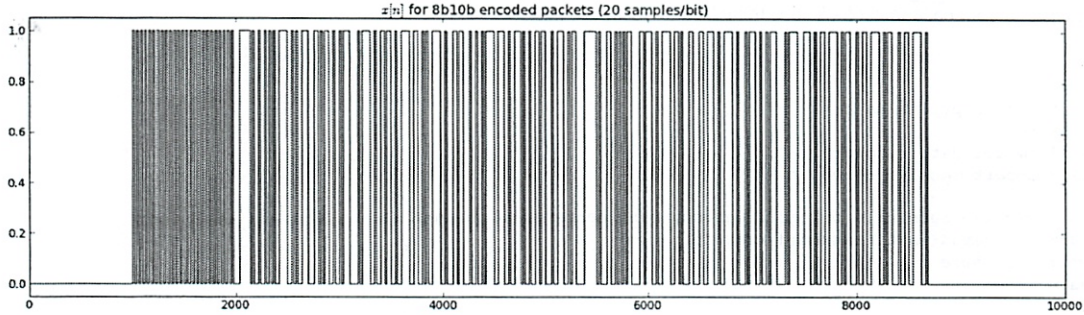


That's better - matches previous ans

Useful download links:

- [PS8\\_tests.py](#) -- test jigs for this assignment
- [PS8\\_rf.wav](#) -- samples from shared channel

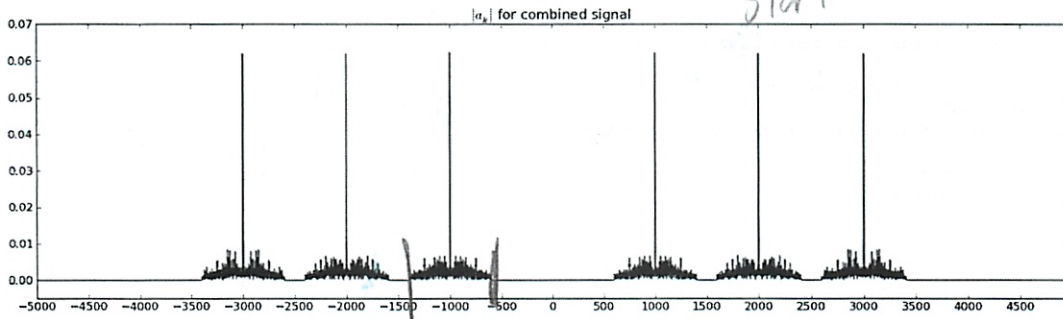
The goal for this task is to write a program to decode messages from a shared channel. The messages are short text sequences that have been encoded into 8b10b packets (see Lab 2!) using 20 samples/bit. The plot below shows the type of sample sequence produced by the encoding:



After some zero samples, there's a clock-recovery training sequence of alternating 0 and 1 bits, followed by several 8b10b packets as described in Lab 2 (SYNC followed by sixteen 8b10b bytes). There are two packets in the sequences you'll be decoding.

*done already*

Each of encoded messages is band-limited and then modulated up to a specific center frequency and all the modulated signals are combined to form the frequency-division multiplexed shared channel. Here's the spectrum of the channel signal:



*Start*

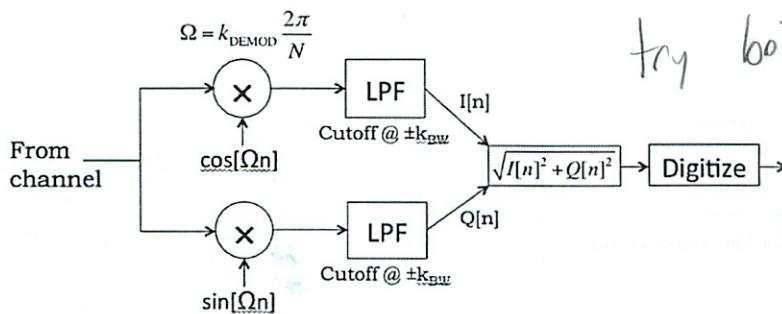
*3 messages or message 3x*

A little examination of this plot should reveal the number of messages, the  $k$  associated with the specific center frequencies and the approximate  $k$  used as the cutoff frequency for the band limiting. Note that phase of the modulating waveforms is unknown.

*Just read off chart*

*Cutoff is what width each dir, we can figure out*

Since we don't know the phase of modulating waveform, you'll have to write a Python function that implements the quadrature demodulator discussed in lecture and shown in the figure below:



*try both ±400*

*but what is k\_demod? the quadr demod is done as well! no - we need to implement*

Your function should take  $k_{demod}$  ( $k_{demod} (2\pi/N)$  is the center frequency of the carrier to be demodulated) and  $k_{BW}$  (the spectral coefficient corresponding to the cutoff frequency of the low-pass filter), returning a digitized sequence of samples. The digitizer should determine the appropriate threshold by looking at the magnitude of the samples it needs to process.

*for each 3 messages?*



`band_limited_samples = PS8_tests.band_limit(samples, kcuttoff)` is a utility routine that implements a low-pass filter with a cutoff at the specified spectral coefficient, i.e., it removes all frequencies with  $k > kcuttoff$  from `samples` and returns the result as a new sample sequence.

`channel_samples = PS8_tests.read_wave_file('PS8_rf.wav')` can be used to read in the channel samples from the supplied .wav file.

You can then use the `data_recovery` and `receive_8b10b` routines you wrote for Lab 2 to recover the message. So your code might look something like:

```
import math
import matplotlib.pyplot as p
import PS8_tests
from PS2_1 import data_recovery
from PS2_2 import receive_8b10b

# return digitized samples from the frequency band centered at kdemod.
def quadrature_demod(samples, kdemod, kcuttoff):
    # your code here
    pass

# read in samples for the shared channel
samples = PS8_tests.read_wav_file('PS8_rf.wav')

# take a look at the spectrum we received
#p.figure()
#PS8_tests.plot_spectrum_magnitude(samples)
#p.title('$|a_k|$ for received signal')
#p.show()

# recover one of the messages by demodulating the appropriate
# piece of the spectrum down to baseband.
kdemod = ???
kcuttoff = ???
digitized_samples = quadrature_demod(samples, kdemod, kcuttoff)

# recover bits from digitized samples
samples_per_bit = 20
bits = data_recovery(digitized_samples, samples_per_bit)

# decode 8b10b-encoded packets
msg = receive_8b10b(bits)

# print result!
print msg
```

most has  
already been done

Good luck! When you're done, please enter the received messages below and then upload your code.

Enter received message(s):

(points: 0)

Upload code for Task 1:

(points: 5)

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.

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.

So now how to do this  
- same as my hang up earlier

$$z[n] = \underset{\substack{\uparrow \\ \text{input}}}{y[n]} \cdot \cos[\Omega n]$$

$$= y[n] \left[ \frac{1}{2} e^{jk_c \frac{2\pi}{N} n} + \frac{1}{2} e^{-jk_c \frac{2\pi}{N} n} \right]$$
$$= \frac{1}{2} \sum_{k=-k_x}^{k_x} a_k e^{j(k+k_c) \frac{2\pi}{N} n} + \dots \text{etc}$$

$$= \frac{1}{4} \dots + \frac{1}{2} \sum_{k=-k_x}^{k_x} a_k e^{jk \frac{2\pi}{N} n} + \frac{1}{4} \dots$$

↑  
get rid

↑  
get rid

↑

So

sum up the  $k$ 's

What is  $k_x$ ?

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

Where did it say  $N$ ?

- # spectral coeffs  $800$ ?



②

First need to limit samples

Nice

Still going really slow!

$a_k$  is what we have?

Now what is  $n$  - the  $n$  we have now!

But what  $n$  do we have

what is  $y[n]$ ?

Is it supposed to be 'in there'?

Thought those were  $a_k$ 's!

Doesn't it change w/ time

So here  $n=1$ ?

Synthesis eq

- look at

- for  $n < N$

- try that

for  $\delta$  -4 -3 -2 -1 0 1 2 3 ~~4~~ ?

3

So do for both sin cases

How to tell which is which?

How to do sin i

minus instead

$$= \gamma[n] \left[ \frac{1}{2} e^{...} - \frac{1}{2} e^{...} \right]$$

but then how to get to next step?

Oh that is y

that is same

$$\left[ \frac{1}{2} \sum + \frac{1}{2} \sum \right] \left[ \frac{1}{2} e - \frac{1}{2} e \right]$$

$$\frac{1}{4} e^{2+} + \frac{1}{2} e^0 + \frac{1}{4} e^{-2}$$

Now  $\frac{1}{4} e^{2+} - \frac{1}{4} e^{-} + \frac{1}{4} e^{-} - \frac{1}{4} e^{2-}$

So  $\frac{1}{4} e^{2+} - \frac{1}{4} e^{2-}$

no middle!

Oh was in slides!

4)

So how to demodulate?

Just try?

---

Give up to lab hrs!

# Coding In Lab

4/22

3 diff messages  
- find all 3

Given  $x[n]$  samples over time domain

- the chart converts to Fourier coeff (freq domain)

When plotting - not given as freq coefficients! \*

\*  $N$ 's period of signal

- have signal don't know if periodic

- repeat

- So take # of time domain samples

\* Modulation is in time domain \*

- when you do the multiplication

---

\* Don't need to cut to only the freq you want first - then demodulate - since multiplying by a certain freq will only bring certain values to center LPTF the rest and you are good!

②

Is  $N$  whole signal?

- Seemed like it

- but why?


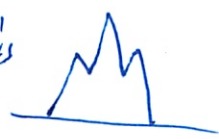
- I think I am visualizing wrong way

# fourier coeffs = period

The  $\sqrt{()^2 + ()^2}$  adds because 0 in some places

want 

not

 → which is 

if phase error

get series of voltages over time

define some digitization threshold

- say .5

- but could also not be

lab closed - kicking me out



3

# At hand

multiply  
~~constant~~  $\sin, \cos$

convert to freq  
what we want will be in middle - always  
- filter



but array from 0 to N

So

$$\frac{N}{2} \pm \text{k cutoff}$$

filter fn?

- yeah

Oh it does cutoff for you!

What are the digitized values b/w?

Oh forgot sq it

But now all seem to be almost 0!

That's not right!

Oh says gain

(4)

Samples are very 0 early on  
- try gain

---

Oh was error in my 8b10 code  
- replaced w/ solutions

How to know digitization limit  
? arg  
- no did not help

---

Oh don't ~~do~~ cos ~~of~~ (sample)!!!  
- Oh I wasn't!

---

Oh convert back to time?  
Much nicer

---

Need new N

Also changed cutoff to 500 helped

---

Emailed in

4/23

Got reply from email 6.02 Coding

band-limit wants fine input

- Oh it converts! for you

- I don't have to

Oh that was easy fix!

5 min

Submit

(2 min late)

## Routing Algorithms

At each intermediate node, choose the one which makes you reach "fastest"

Need to decide what you want to prioritize?

1. Shortest distance

2. Fastest time

3. Less gas

4. Lowest ~~foreign~~ carbon emissions

So ~~pick~~ find the shortest weighted path

## Algorithm

- distributed

- constant

- greedy-like

- at each stage just do the best right now

Weights statically assigned

- both endpoints know them

Called distance vector protocol (DV)

- maintain distance to every destination



2

Cost is  $\infty$  if don't know how to reach

Every node tells its neighbors

- I'm alive
- My current vector dictionary

(This stuff is easy again)

(Hard to write how ~~the~~ rating spreads)

There are no cycles - since that would be pointless

- if there is a cycle, then your path is wrong

Kinda like Viterbi Trellis

- shortest path finding
- since assigned prob to each edge
- "stochastic dynamic programming"
  - any optimization program
  - fin firm w/ some uncertainty
- want in parallel
- aka Belief propagation decoder
- Synchronous/A-synchronous does not matter

Updated once every few hrs

(3)

Weights keep getting updating

- like if a link went down
- but if all the links to one section fail
- then costs grow stupid as both nodes just send the packet back + forth to each other
- so just cut it off after some value and mark it as  $\infty$
- called route flapping
- takes a while to update
- esp if costs change freq

Can ya do something to make it faster?

- w/o maintaining whole path history
- ...

### Modulation Delay

$x[n]$  is signal of freq  $f$

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

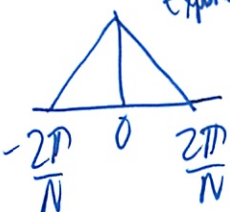
Going to modulate at freq  $f$

$$f = \frac{2\pi}{N} \text{ km}$$

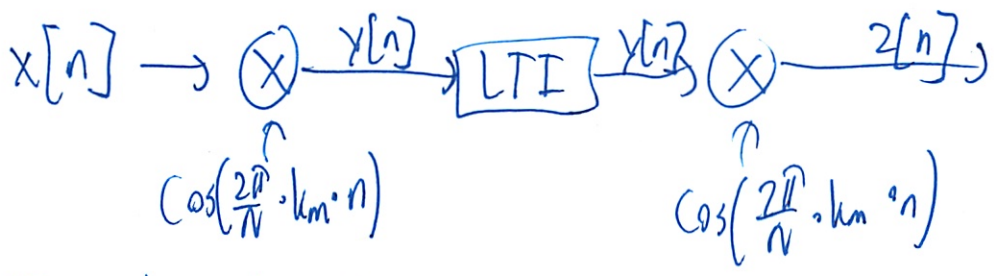
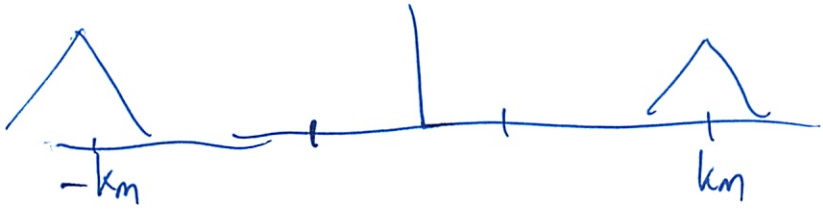
④ 

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

$$= \frac{1}{2} \left( \exp\left(j \cdot \frac{2\pi}{N} \cdot n\right) + \exp\left(-j \cdot \frac{2\pi}{N} \cdot n\right) \right) \cdot \frac{1}{2} \left( \exp\left(j \cdot \frac{2\pi}{N} \cdot k_m \cdot n\right) + \exp\left(-j \cdot \frac{2\pi}{N} \cdot k_m \cdot n\right) \right)$$

$x[n]$  is    
 ↑ having freq - this exponent non trivial

Want to transmit around  $k_m$

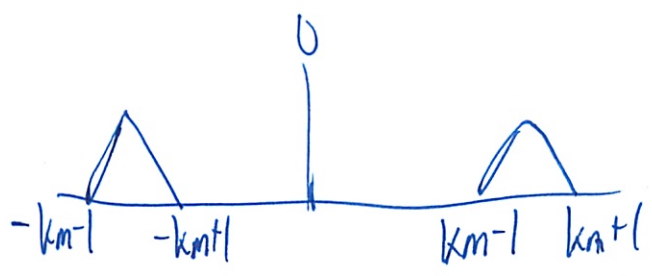


For sin - doesn't matter

For what sin, cos - just shifted

$$= \frac{1}{4} \left( \exp\left(-j(k_m+1) \cdot \frac{2\pi}{N} \cdot n\right) + \exp\left(-j(k_m-1) \cdot \frac{2\pi}{N} \cdot n\right) + \exp\left(j(k_m-1) \cdot \frac{2\pi}{N} \cdot n\right) + \exp\left(j(k_m+1) \cdot \frac{2\pi}{N} \cdot n\right) \right)$$

5



What if clock is slow? Delay  
 $n$  is smaller for receiver than transmitter

$$z[n] = y[n+3]$$

So what is demodulated output?

$$y(n) = z[n] \cdot \cos\left(\frac{2\pi}{N} \cdot k_m \cdot n\right) = \text{above signal except } n+3 \text{ instead of } n$$

So write in complex exponential

$$= \frac{1}{4} \left[ \exp\left(-j \left(k_m+1\right) \cdot \frac{2\pi}{N} \cdot n - \underbrace{\left(k_m+1\right) \cdot \frac{2\pi}{N} \cdot 3}_{\phi_1}\right) \right]$$

$\phi_1 \leftarrow$  constant  
 - does not vary w/  $n$

$$+ \exp\left(-j \left(k_m-1\right) \frac{2\pi}{N} n - \phi_2\right)$$

$$+ \exp\left(+j \left(k_m-1\right) \frac{2\pi}{N} n + \phi_2\right)$$

$$+ \exp\left(+j \left(k_m+1\right) \frac{2\pi}{N} n + \phi_1\right)$$

$$\frac{1}{2} \left( \exp\left(-j \cdot \frac{2\pi}{N} \cdot k_m \cdot n\right) + \exp\left(+j \cdot \frac{2\pi}{N} \cdot k_m \cdot n\right) \right)$$

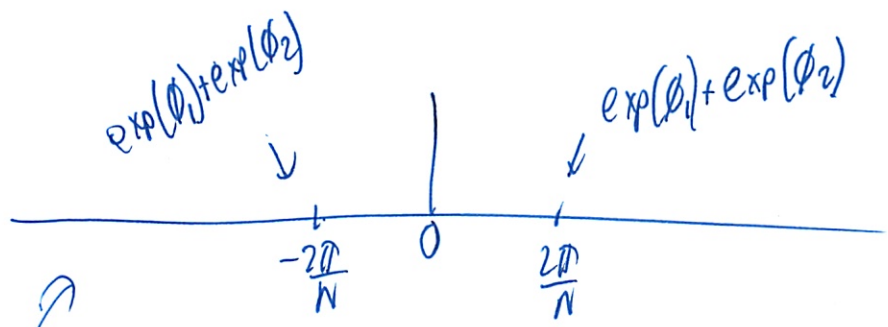
Care about what comes around 0



6

Statt candles

Look at freq response



would be 0 if no delay

If  $\frac{D}{2}$  then only in complex domain

What does this mean?

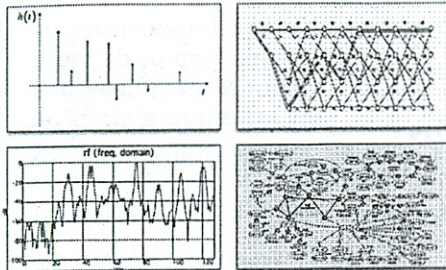
$$(k\pi + 1) \cdot 0 \cdot \frac{2\pi}{N} = \frac{D}{2}$$

Solve for  $D$

Here is just x component

Also capture y to do quadrature demodulation





INTRODUCTION TO BECS II  
**DIGITAL  
 COMMUNICATION  
 SYSTEMS**

*Rating 2 of 2*

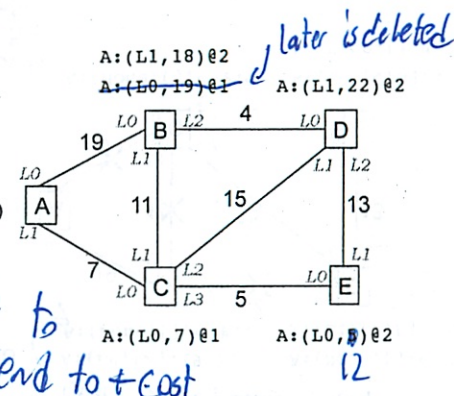
**6.02 Spring 2011  
 Lecture #20**

- distance vector review
- path-vector routing
- link-vector routing
- Dijkstra's shortest path algorithm
- hierarchical routing

**Distance-vector (DV) review**

At each ADVERT interval, nodes tell neighbors (dest, cost) for all routes in their routing table.

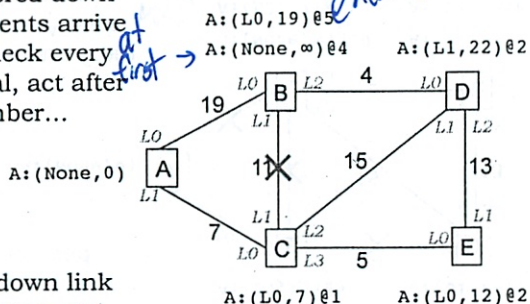
*Each node has own  
 has list of every node in network and a table first place to send to + cost  
 Sends its entire table to its immediate neighbor*



*What happens when break a link?*  
**Link is Down at Time 4?**

*also heart A, 26 - but more expensive  
 next time step - next ad*

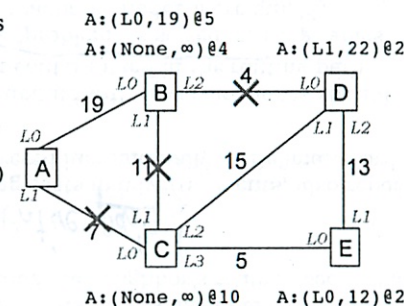
A link is considered down if no advertisements arrive over the link; check every ADVERT interval, act after some small number...



Routes using a down link are changed to have cost  $\infty$ , which will propagate to neighbors who then update their cost if they used you for their route.

**Partitioned Network at Time 10**

An unfortunate combination of down links might partition the network.



Routes using a down link are changed to have cost  $\infty$ , which will propagate to neighbors who then update their cost if they used you for their route.

*Only thing that happened up front*

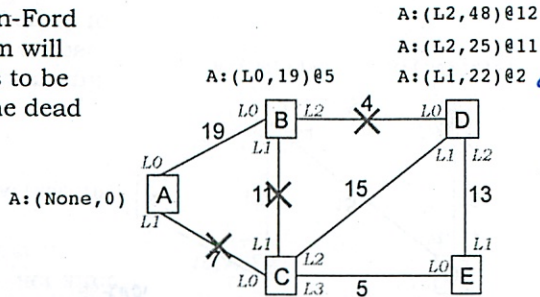
*4/25*



*E's route to A actually goes to C*

## Count to Infinity *Why is C sending to E*

Now the Bellman-Ford update algorithm will cause new costs to be calculated for the dead routes.



*hears from C*

*Everyone assuming other has magic path*

For example, C hears from E about a route to A with total cost 17. Since only costs are kept, C can't tell that E was relying on it for its route of cost 12!

*hears from E*

The costs spiral higher, eventually passing some bound, at which point they are recognized as  $\infty$ .

## Fixing "Count to Infinity"

- Problem
  - Node C's route to A breaks, C sets cost to  $\infty$
  - But at next round of advertisements, hears of lower-cost routes from neighbors, not know the neighbor's routes used C itself to get to A.
- Solution
  - In addition to reporting costs in advertisements, also report routing path as discovered incrementally by Bellman-Ford
  - Called "path-vector"
  - Modify Bellman-Ford update with new rule: nodes should ignore advertised routes that contain itself in the routing path
  - Pros: count-to-infinity "problem" is solved (routing tables eliminate routes to unreachable nodes more quickly)
  - Cons: advertisement overhead is larger

*send entire path!*

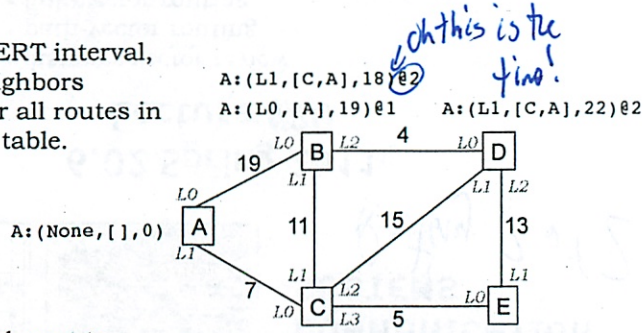
*No protocol has no routing loops - needs fine to update*

*so put TTL on packet*

*but after some time*

## Path-vector (PV) routing

At each ADVERT interval, nodes tell neighbors (path, cost) for all routes in their routing table.



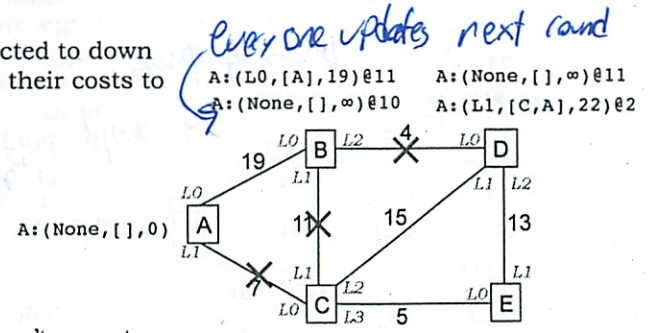
*oh this is the find!*

*keeping full path*

Nodes add link cost to neighbor's routing costs and keep their routing table up-to-date with shortest-path route.

## Partitioned Network at Time 10: PV

Nodes connected to down links change their costs to  $\infty$ .



*every one updates next round*

Using PV, C won't accept routes to A from either D or E since C appears on the path they advertise. Unreachable nodes are quickly removed from tables.

Pros: simple, works well for small networks  
Cons: only works for small networks

*only works for small networks*



# Different Link-State Routing

- Advertisement step
  - Send information about its links to its neighbors (aka *link state advertisement* or LSA):

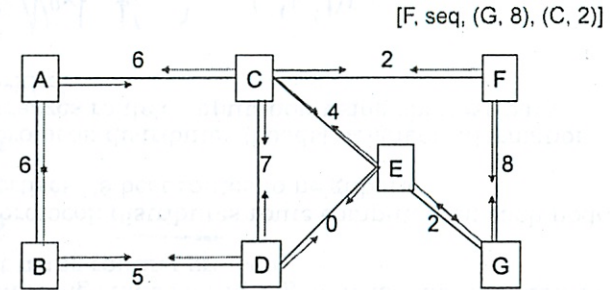
[seq#, [(nbhr1, linkcost1), (nbhr2, linkcost2), ...]]

neighbors forward incoming ads on further! - increase link heard on  
- don't change keep up to date list

diff than what I saw on the then

- Do it periodically (liveness, recover from lost LSAs)
- Integration
  - If seq# in incoming LSA > seq# in saved LSA for source node: update LSA for node with new seq#, neighbor list rebroadcast LSA to neighbors (→ **flooding**)
  - Remove saved LSAs if seq# is too far out-of-date
  - Result: Each node discovers current map of the network
- Build routing table
  - Periodically each node runs the same *shortest path algorithm* over its map
  - If each node implements computation correctly and each node has the same map, then routing tables will be correct

# LSA Flooding



- LSA travels each link in each direction
  - Don't bother with figuring out which link LSA came from
- Termination: each node rebroadcasts LSA exactly once
- All reachable nodes eventually hear every LSA
  - Time required: number of links to cross network

don't rebroadcast again

travels each link exactly twice

# Dijkstra's Shortest Path Algorithm

nodeset ABCDEF

spcost routes  
A B C D E F  
∞ ∞ ∞ ∞ ∞ ∞  
- - - - - -

- Initially
  - nodeset = [all nodes] = set of nodes we haven't processed
  - spcost = {me:0, all other nodes: ∞} # shortest path cost
  - routes = {me:--, all other nodes: ?} # routing table
- while nodeset isn't empty:
  - find u, the node in nodeset with smallest spcost
  - remove u from nodeset
  - for v in [u's neighbors]:
    - d = spcost(u) + cost(u,v) # distance to v via u
    - if d < spcost(v): # we found a shorter path!
      - spcost[v] = d
      - routes[v] = routes[u] (or if u == me, enter link from me to v)
- Complexity: N = number of nodes, L = number of links
  - Finding u (N times): linear search=O(N), using heapq=O(log N)
  - Updating spcost: O(L) since each link appears twice in neighbors

every other node

node next closest

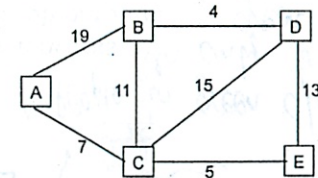
enter first starting

$O(N^2 + L)$  or  $O(N \log N + L)$   
depends on method

# Dijkstra Example

Finding shortest paths from A:

- LSAs:  
 A: [(B,19), (C, 7)]  
 B: [(A,19), (C,11), (D, 4)]  
 C: [(A, 7), (B,11), (D,15), (E, 5)]  
 D: [(B, 4), (C,15), (E,13)]  
 E: [(C, 5), (D,13)]



Step	u	Nodeset	spcost					route				
			A	B	C	D	E	A	B	C	D	E
0		[A,B,C,D,E]	0	∞	∞	∞	∞	--	?	?	?	?
1	A	[B,C,D,E]	0	19	7	∞	∞	--	L0	L1	?	?
2	C	[B,D,E]	0	18	7	22	12	--	L1	L1	L1	L1
3	E	[B,D]	0	18	7	22	12	--	L1	L1	L1	L1
4	B	[D]	0	18	7	22	12	--	L1	L1	L1	L1
5	D	[]	0	18	7	22	12	--	L1	L1	L1	L1

← sends all to C  
just next node for each



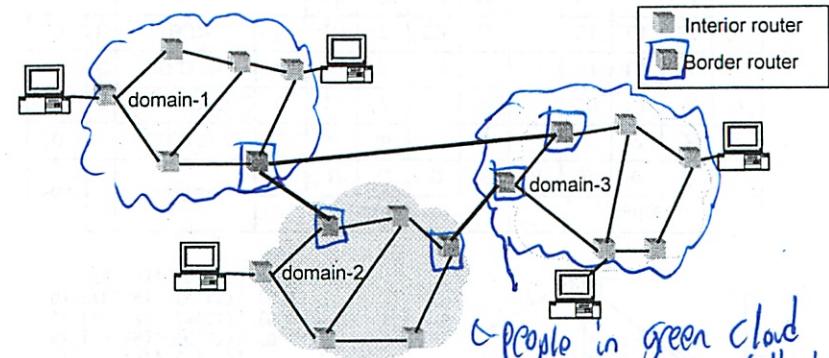
# Why is Network Routing Hard?

- Inherently distributed problem
  - Information about links and neighbors is local to each node, but we want global reach
- Efficiency: want reasonably good paths, and must find them without huge overhead
- Handling failures and "churn"
  - Must tolerate link, switch, and network faults
  - Failures and recovery could be arbitrarily timed, messages could be lost, etc.
- Scaling to large size very hard (later courses)
  - And on the Internet, many independent, competing organizations must cooperate
  - Mobility makes the problem harder

*need to know what dir to head or using long paths*

*more than need to know*

# Hierarchical Routing



- Internet: collection of domains/networks
- Inside a domain: Route over a graph of routers
- Between domains: Route over a graph of domains
- Address: concatenation of "Domain Id", "Node Id"

*people in green cloud can only talk in green cloud*

*hierarchical*

*hierarchy of much smaller graphs - can use DV*

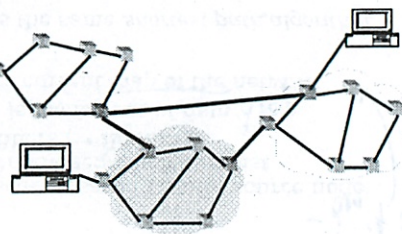
*bandwidth packets to talk to outside world*

# Pros and Cons

## Advantages

- Scalable
  - Smaller tables
  - Smaller messages
- Delegation
  - Each domain can run its own routing protocol

*100-200*



## Disadvantages

- Mobility is difficult
  - Address depends on geographic location
- Sup-optimal paths
  - E.g., in the figure, the shortest path between the two machines should traverse the yellow domain. But hierarchical routing goes directly between the green and blue domains, then finds the local destination → path traverses more routers.

# Summary

- The network layer implements the "glue" that achieves connectivity
  - Does addressing, forwarding, and routing
- Forwarding entails a routing table lookup; the table is built using routing protocol
- DV protocol: distributes route computation; each node advertises its best routes to neighbors
- LS protocol: distributes (floods) neighbor information; centralizes route computation using shortest-path algorithm

*Next time: reliability*

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 PS9

#### Dates & Deadlines

issued: Apr-20-2011 at 00:00

due: Apr-28-2011 at 06:00

checkoff due: May-03-2011 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 [Piazza](#).

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.

Over many months, you and your friends have painstakingly collected a 1,000 Gigabytes (aka 1 Terabyte) worth of movies on computers in your dorm (we won't ask where the movies came from). To avoid losing it, you'd like to back the data up on to a computer belonging to one of your friends in New York.

#### Terminology:

1 kilobyte =  $10^3$  bytes

1 megabyte = 1000 kilobytes =  $10^6$  bytes

1 gigabyte = 1000 megabytes =  $10^9$  bytes

1 terabyte = 1000 gigabytes =  $10^{12}$  bytes

You have two options:



- A. Send the data over the Internet to the computer in New York. The data rate for transmitting information across the Internet from your dorm to New York is 2 Megabytes per second. How many days will it take to transfer 1 Terabyte using this method?  $10^6$

$$\frac{10^{12}}{2 \cdot 10^6} = \frac{10^6}{2} \text{ seconds} \quad \frac{10^6/2}{60 \cdot 60 \cdot 24} = 5.78 \text{ days}$$

(points: 0.5)

- B. Copy the data over to a set of disks, which you can do at 100 Megabytes per second. Then rely on the US Postal Service to send the disks by mail, which takes 4 days. How many days will it take to transfer 1 Terabyte using this method? *Shepherd*

*Assume  $\infty$  disks*

$$\frac{10^{12}}{100 \cdot 10^6} = 10,000 \text{ seconds to load} \quad \frac{10,000}{60 \cdot 60 \cdot 24} = 11 \text{ days} + 4 = 15 \text{ days}$$

(points: 0.5)

**Problem 2.**

It is sometimes beneficial to communicate using radio between terrestrial computers via a switch on a geostationary satellite, positioned 36,000 kilometers above the surface of the earth.

- A. What is the minimum round-trip time, in seconds, for any network communication between two computers so connected? (The round-trip time from A to B is defined as the time to send a small packet from A to B and to receive a response or acknowledgment from B at A.) Radio signals travel at the speed of light ( $3 \cdot 10^8$  meters per second in air and vacuum).

*Assume vacuum*

$$\frac{36,000,000 \cdot 2}{3 \cdot 10^8} = 24 \text{ seconds}$$

(points: 0.5)

- B. Suppose the one-way delay between the two computers is D seconds and that the bit rate of data transmission between them is B bytes/second. How many bytes would the sending computer have sent (en route to the receiver) at the time when the receiver gets its first byte? Assume that there are no errors or losses. Enter your answer as an



oh I did direct link - I think that is right as a base  
 will just go w/ that?

(points: 1)

To handle failures in this network, each node implements the following rule in addition to the three mentioned earlier:

**Rule 4.** Immediately upon detecting the inability to reach a directly connected neighbor, update the cost to each destination in the routing table that uses the corresponding link to INFINITY (set to 100).

Suppose every node has found a correct route to every other node. The link between C and D fails and C detects the failure at some time  $T$ . There are no other failures and the link doesn't come back up. Each packet takes zero time to be sent along a link and be processed at a node. Packets don't carry a "hop limit" ("time to live") field.

Which of the following statements is correct?

- C. Given enough time, the routing protocol converges, with every node having the correct cost for every destination.

$t=T$        $t=T+100$   

A	ABCD
B	ABCD
C	ABC
D	D

 No True if possible cost is  $\infty$

(points: 0.25)

- D. There exists some time  $T$  such that a packet sent from A to D at some time after  $T$  bounces back-and-forth between B and C.  $\uparrow 2 T_s$  different wording

emailed in: just asking if is finite time where bounces back and forth  
 No - B not on route after C  
 $A \rightarrow B \rightarrow C \rightarrow D$

(points: 0.25)

- E. There exists some time  $T$  such that a packet sent from A to D at some time after  $T$  bounces back-and-forth between A and B.  $\uparrow$  same

expression. *abstract!*

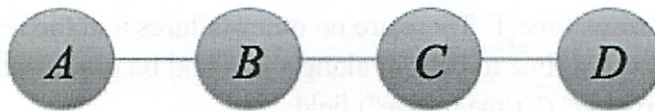
In  $D$  sec  $\underbrace{\hspace{2cm}}$ , how many bytes  
 $\text{bytes} \cdot d \quad \text{bytes} \cdot \frac{1}{\text{sec}} = \text{bytes}$

(points: 0.5)

**Problem 3. ChainNet**

*DV*

Each node in the chain-like network topology shown below runs a distance vector protocol, sending routing information to its neighbors, according to these rules:



**Rule 1.** Send the first distance-vector advertisement immediately after power up.

**Rule 2.** Every 100 seconds thereafter, send a distance-vector advertisement containing the best route and cost for each known destination. That is, if a node powers on at time  $t$ , it sends advertisements at times  $t, t+100, t+200, t+300, \dots$

**Rule 3.** As soon as a route advertisement is heard, update the route and corresponding cost if, and only if: (a) the new cost is smaller than the current one, OR (b) the cost of the current route changes.

Once sent, the delay before a packet is received at a neighbor is negligible. Unless mentioned otherwise, assume that no route advertisements are lost between nodes that are on. Each link has cost 1.

*knows possible neighbors*

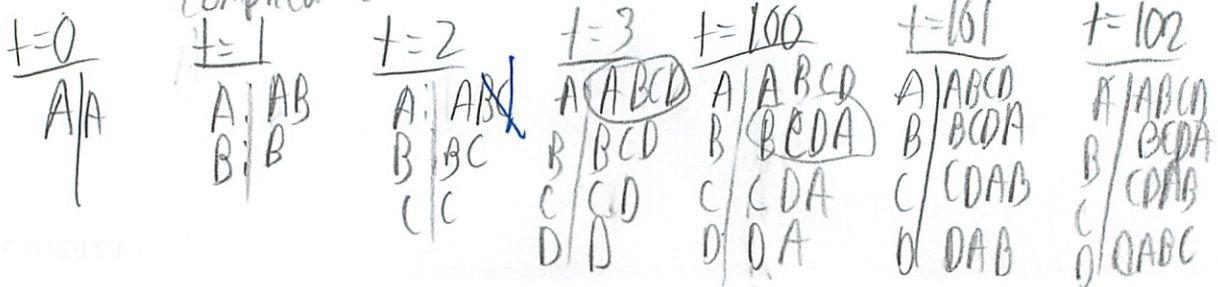
In this network, there is no HELLO protocol in place. Each node knows the identity of each of its neighbors, but **does not know if the neighbor is alive or not**. Each node learns whether a neighbor is alive or not only by receiving an appropriate distance-vector advertisement.

Initially, all nodes are off. Node A powers on at time  $t=0$ , node B at  $t=1$ , node C at  $t=2$ , and node D at  $t=3$  seconds.

*assume message rec instantly*

A. At what time will node A have correct routing information about all the other nodes in the network? And at what time will node D have correct routing information for all the other nodes in the network?

*Complicated*





$A \rightarrow B \rightarrow C$   
 No

(points: 0.25)

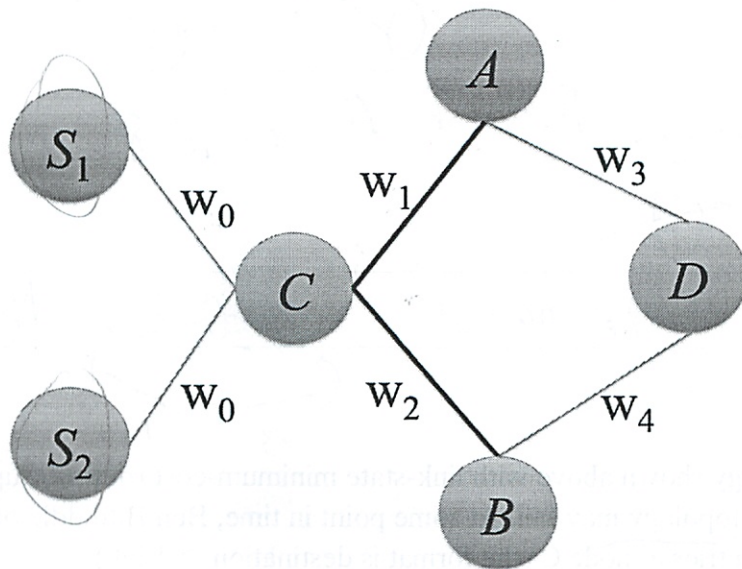
F. There exists some time T such that a packet sent from A to D at some time after T traverses the path A - B - C - B - A - B - C - B - A - B - C - B ...

No - won't go back  
 Done at C

(points: 0.25)

**Problem 4. FishNet**

Ben Bitdiddle is responsible for routing in FishNet, shown below. He gets to pick the costs for the different links ( $w_0, w_1, w_2, w_3,$  and  $w_4$  shown near the links). All the costs are non-negative integers.



Goal: To ensure that the links connecting C to A and C to B, shown as darker lines, carry equal traffic load. All the traffic is generated by S1 and S2, in some unknown proportion. The rate (offered load) at which S1 and S2 together generate traffic for destinations A, B and D are  $R_A, R_B,$  and  $R_D,$  respectively. Each network link has a bandwidth higher than  $R_A + R_B + R_D$ . There are no failures.

Protocol: FishNet uses link-state routing; each node runs Dijkstra's algorithm to pick

Bellman Ford  
 DV - just direction + distance to destination  
 everyone has a map  
 not constrained  
 have not learned yet, what LS uses to pick shortest

minimum-cost routes. If there two nodes at any step that have the same min cost, assume the choice is made arbitrarily.

*So assume knows whole map*

*-this is how*

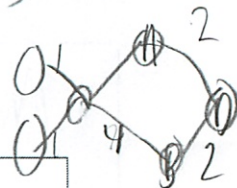
Suppose  $R_A + R_D = R_B$ . For the following questions, choose weights that guarantee that there will be equal traffic on LinkCA and LinkCB, regardless of how ties are resolved when picking the minimum cost routes. (Note that all link costs are non-negative integers.)

*answered last set problem*

A. Given the following link costs, what is the largest possible value for  $w_1$ ?

- $w_0 = 1$
- $w_2 = 4$
- $w_3 = 2$
- $w_4 = 2$

*So = traffic - w1 extra traffic on B  
-or just merge A,D*

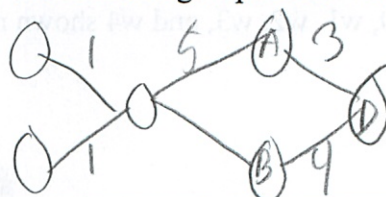


Need to send D through A, must be  $< 6 = 5 - 2 = 3$

(points: 0.5)

B. Given the following link costs, what are the smallest and largest possible values for  $w_2$ ?

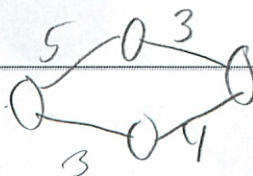
- $w_0 = 1$
- $w_1 = 5$
- $w_3 = 3$
- $w_4 = 3$



Same - must route D through A, so  $< 8$   
 ~~$7 - 4 = 3$~~

(points: 0.5)

*Think about it*

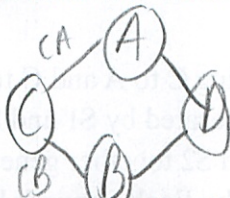


*No needs bigger! (5)*

**Problem 5. More FishNet**

Consider the FishNet topology shown above with link-state minimum-cost routing. Suppose that some of the links in the topology may fail. At some point in time, Ben Bitdiddle observes the following routing table entries at node C (the format is destination -> Link):

- A -> link CA
- D -> link CA
- B -> link CB



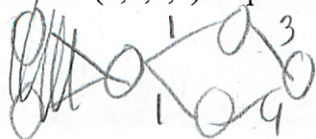
*What scenario?*

Which of the scenarios below is consistent with observing the above routing table? Once again, just guessing the answers wouldn't be a good strategy...

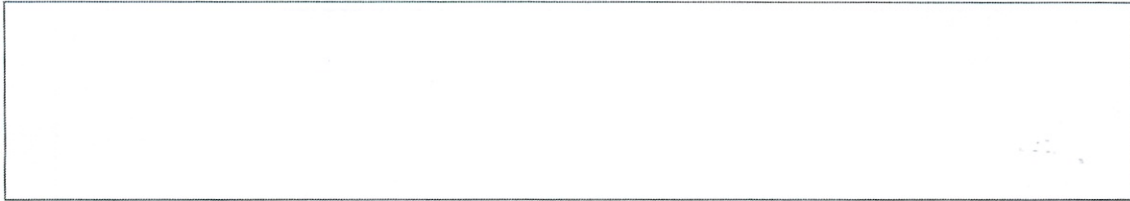
*still want it to be even*

A. Weights  $w_1$  through  $w_4$  are (1,1,3,4) respectively, and no link has failed.

*Ans is 1 or 0*

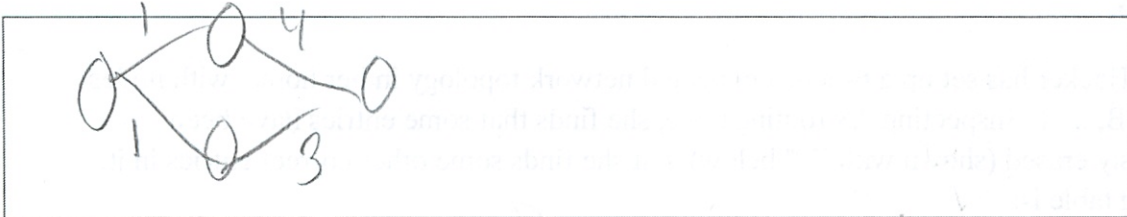






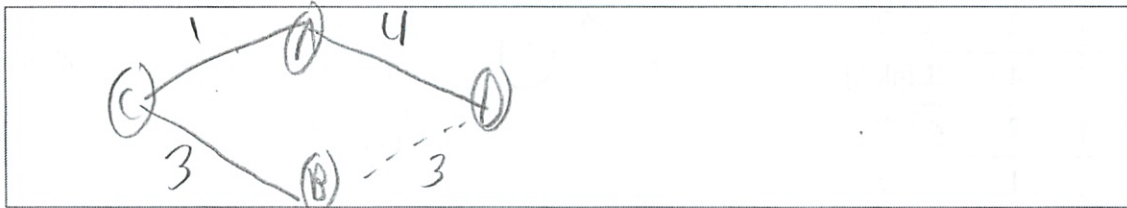
(points: .15)

B. Weights  $w_1$  through  $w_4$  are (1,1,4,3) respectively, and no link is down



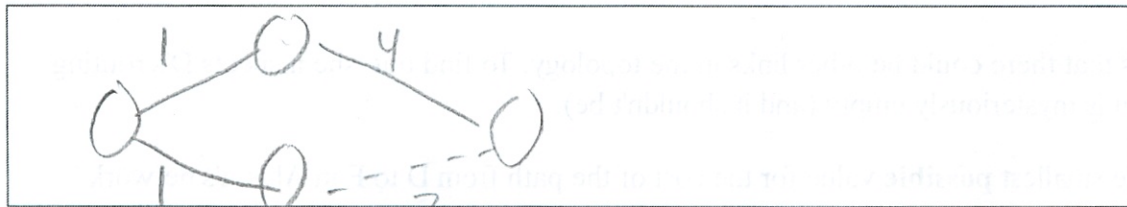
(points: .15)

C. Weights  $w_1$  through  $w_4$  are (1,3,4,3) respectively, and link BD is down



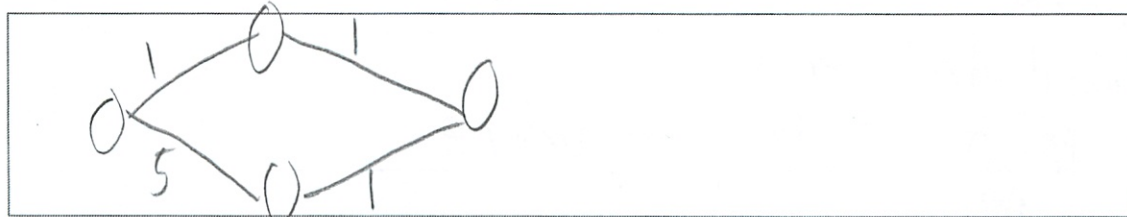
(points: .15)

D. Weights  $w_1$  through  $w_4$  are (1,1,4,3) respectively, and link BD is down



(points: .15)

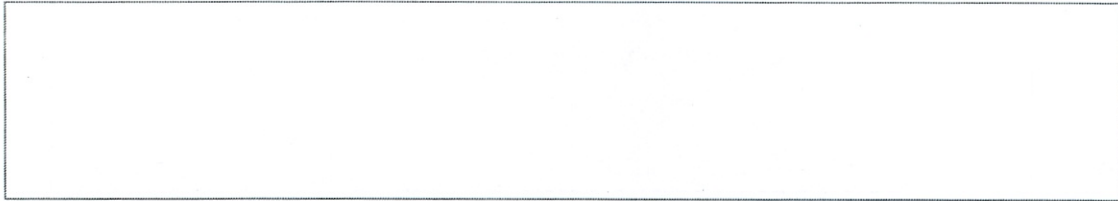
E. Weights  $w_1$  through  $w_4$  are (1,5,1,1) respectively, and no link is down



(points: .2)

F. Weights  $w_1$  through  $w_4$  are (1,5,1,1) respectively, and link BD is down



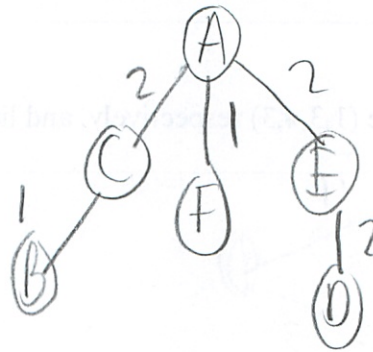


(points: .2)

**Problem 6.**

Alyssa P. Hacker has set up a 6-node connected network topology in her home, with nodes named A, B, ..., F. Inspecting A's routing table, she finds that some entries have been mysteriously erased (shown with ``?'' below), but she finds some other correct entries in it. A's routing table is:

Destination	Path cost	Link
B	3	LinkAC
C	2	AC?
D	4	LinkAE
E	2	AE?
F	1	?



← have to draw or walk never see

Each link has a cost of either 1 or 2 and link costs are symmetric (the cost from link X to link Y is the same as the cost from Y to X). The routing table entries correspond to minimum-cost routes.

She knows that there could be other links in the topology. To find out, she inspects D's routing table, but it is mysteriously empty (and it shouldn't be).

What is the smallest **possible** value for the cost of the path from D to F in Alyssa's network topology? Assume that any two nodes may possibly be directly connected to answer this question. (Hint: It will help to construct the topology on paper using the information given.)

So now 5  
 Can it direct connect?  
 - No d can't be 4 from Link AE

(points: 1)

Does not seem like it

**Problem 7.**

Anette Worker wants to calculate the bandwidth consumed by distance vector routing advertisements in her network, which has  $n=200$  nodes and  $m=500$  links. Each node address is 4 bytes long and each cost is a 2-byte integer. Each advertisement has a 4-byte sequence

DL



number, the sending node's address, and then the vector of [address,cost] tuples (4+2 = 6 bytes each). Each advertisement is sent by a node once every 30 seconds to its neighbors. Ignore the bandwidth consumed by HELLO messages.

What is the total bandwidth consumed by routing advertisements in bytes/second? (Round to nearest integer.)

So DV set every 30 sec - so don't need time to propagate

Each node sends 1 to each link

So every 30 sec  $200 \times (4\text{bit} + 4\text{bit}) + 500 \text{ of } 6 \text{ each}$  2 dir 1000

(points: 1)

$$\frac{200 \cdot 8 + 1000 \cdot 6}{30} = 253 \frac{1}{3} \rightarrow 253$$

### Introduction to the Python tasks

This lab uses NetSim, a simple packet-level network simulator. You will write the code for the main components of distance-vector and link-state routing protocols.

NetSim executes a set of steps every time slot; time increments by 1 each slot. During each time slot a link can deliver one packet from the node at one end of the link to the node at the other end of the link.

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

To understand the different parameters one can set in NetSim, go to a shell (terminal window) and enter:

```
python PS9_1.py -h
```

This command prints out the various options:

-g, --gui	show GUI
<del>-c, --checkoff</del>	<del>checkoff the lab task</del> <i>no longer here</i>
-n NUMNODES, --numnodes=NUMNODES	number of nodes
-t SIMTIME, --simtime=SIMTIME	simulation time
-r, --rand	use randomly generated topology

The -r option generates a random topology with the specified number of nodes. The default number of nodes is 12 and the default simulation time is 2000 time slots; you can change both using the corresponding command-line options.

This lab has two main tasks, each with a few sub-tasks. The first task is to implement a distance-vector (DV) routing protocol. The second task is to implement a link-state (LS) routing protocol. You may find it useful to debug your code using the -r option with randomly generated topologies, and it may be useful for you to use the -g option during the debugging process.

If you do not use the -g option, the code will run against a set of test topologies, which



include static and changing costs as well as link failures. Each test carries a certain number of points.

Both routing protocols should construct the minimum-cost path from the Router to all the other destination addresses that are currently reachable in the network. The destination is derived from the Router class and has an `address` field that will be used as an index into the routing table and the cost table. Unless explicitly mentioned otherwise, we will use the term "minimum-cost path" and "shortest path" interchangeably.

### Useful classes and data structures

*The Router class:* The logic for DV resides in the `DVRouter` class; the logic for LS is in the `LSRouter` class. Both these classes are derived from the Router class, defined in `lab8_router.py`. The main goal of your software is to construct and maintain two key pieces of information in the `DVRouter` and `LSRouter` classes: *the routing table* and *the cost of the minimum-cost path* from the Router to the various destination addresses.

That is, your code should correctly produce and maintain:

- `self.routes`, the routing table: a dictionary that maps from a destination address to a `Link`. This link is the link that the Router will use to forward any packet destined for the corresponding destination address.
- `self.spcost`, the table of costs of the shortest (i.e., minimum cost) paths: A dictionary that maps from a destination address to the current estimate of the cost of the path to get there.

*The Link class:* This class is defined in `PS9_netsim.py`; you don't need to modify it. But you should know that you can obtain the cost of a link, `L`, using `L.cost` (a variable in class `Link`). The other place you'll use the `Link` class is to populate the routing table, which (by definition) stores the `Link` to be used to reach any destination. We have provided a useful function in the Router class, `getlink(n)`, which takes a neighboring Router, `n`, and returns the `Link` connecting `self` to `n`. This function is often useful in constructing the routing table.

### The HELLO protocol and maintaining live neighbors

We have already implemented the HELLO protocol for you in `lab8_router.py` (you should not need to modify this file); each Router sends a HELLO packet every `HELLO_INTERVAL` time slots (10 by default in NetSim). Whenever a node hears a HELLO packet along a link, it adds the current time, the address of the Router at the other end of the link, and the cost of that link, to `self.neighbors`, which is a dictionary mapping a `Link` to a (timestamp, address, linkcost) tuple.

If a node does not hear a HELLO for more than  $2 * \text{HELLO\_INTERVAL}$  time slots on a link, it removes that node from `self.neighbors` and calls the Router's `link_failed(link)` function, giving the "failed" link as argument. In response, your routing protocol implementation may take suitable actions.

**Debugging and Testing Procedures** Writing distributed protocols, even in simulation, can be



a challenge. To help you a bit, we have some utilities that are accessible via the GUI (`-g` command-line option). By clicking on any node while the simulation is running, you can look at its routes and shortest path costs to every destination. For link-state routing, clicking on a node also prints out the last LSA information available at that node from each of the other nodes.

Please note that clicking on any link toggles the state of the link between "working" and "failed" states, letting you test your protocol under link failures. We will want to ensure that your protocol works properly in the face of failures and recoveries.

The "Step 1", "Step 10", and "Step 100" buttons are the way in which you should step through the operation of your protocol and see what is happening by clicking on various nodes.

In any given time-slot, colored squares may appear on a link. These are packets. The packets are color-coded: green ones are HELLO packets, red are advertisements (type ADVERT), and blue are data packets. (The blue data packets aren't relevant to this lab.)

For both the routing protocols you will implement in this lab, you should be prepared to demonstrate the correctness of the routing tables at the various nodes in the following scenarios listed below. Test your protocols with the `-r` option, which will generate random topologies. And make sure to test it on the test topologies and conditions we have provided (accessible when you don't use the `-g` option).

1. *No link failures*. All routes computed at the various nodes must actually correspond to the shortest paths in the graph. In addition, every node must have a routing table entry for all other nodes in the topology.
  - *Metric topology*: The link costs are for a topology where the costs satisfy the metric property, i.e., one that satisfies the triangle inequality ( $\text{cost}(AB) + \text{cost}(BC) > \text{cost}(AC)$ ) for all triples of links AB, BC, AC.
  - *Non-metric topology*. The topology does not satisfy the triangle inequality.
2. *One or more link failures, but a connected network*. The routing protocol should be able to adjust its routes to route packets "around" the failed link. You should be able to give a rough estimate of how long link state and distance vector protocols take to find the new routes. You should also bring the failed link back up and verify that the routes go back to what they looked like in the base case.
3. *Dynamic cost changes and failures*. After the previous test has converged, change some link costs and introduce new failures to see if the protocol computes routes correctly.
4. *Convergence time*. This test configures some changes to link costs (both increase and decrease of costs) and sees whether the protocol converges fast enough.
5. *Disconnected topology*. We break multiple links in the topology such that some subset of the nodes cannot reach the other subset. You must show that your routing protocol eventually converges to the correct routing in this case -- that is, a node must have a



routing table entry to every node it is connected to by a path in the underlying topology, and must not have a routing table entry for any node it cannot reach. You should also be able to provide a rough estimate of how long this convergence process takes for the two routing protocols by thinking about it and from your observations of the protocol in simulation. For distance vector routing, you should also demonstrate the "count to infinity" problem during convergence. Finally, when you "heal" one or more links to produce a connected topology, the protocol should eventually ensure that all nodes find the correct routes to all destinations.

## Python Task #1: Distance vector (DV) routing

*just do what lab says*

Useful download links:

PS9\_netsim.py -- network simulator

PS9\_tests.py -- testing functions

PS9\_1.py -- template file for this task

The file you will have to extend is PS9\_1.py.

This file contains the `DVRouter` class, which is derived from the `Router` class (which in turn derives from the `Node` class defined in PS9\_netsim.py). Your first task for this lab is to write the following three functions, which are the core of any DV protocol:

1. `make_dv_advertisement()`: Scan the `self.routes` and `self.spcost` tables and construct a list of `[(dest1, cost1), (dest2, cost2) ...]`. Return this list.

As explained in lecture and the lecture notes, each router in a DV protocol periodically exchanges *routing advertisements* with its neighbors in the network topology, containing the information about destinations and their shortest-path costs `[(dest1, cost1), (dest2, cost2), ...]`. This function will be called every `ADVERT_INTERVAL` time slots (50 by default in NetSim) by `send_advertisement()`, which will take care of constructing packets with this list as its contents, and sending one such packet to each neighbor in the topology.

*Sends to its neighbors its entire list of known dest - but only next step packet!*

2. `link_failed(link)`: Called when the HELLO protocol determines a failure. When called, your code needs to take suitable action to recognize that the link is now "dead". For example, depending on how you design your DV protocol, you may: set `self.spcost` for all destinations whose routing table entries currently use that link to `self.INFINITY` (note: `self.INFINITY`, not just `INFINITY`), delete that route from your table, or anything else.

We are intentionally not specifying the precise behavior, leaving it to you to design it. As long as what you do in this step is consistent with what you do in `make_dv_advertisement()`, your protocol will work correctly. Conversely, an inconsistency will likely cause the protocol to be incorrect.

3. `integrate(link, adv)`: This function is where the actual distributed computation



occurs. It takes as input two arguments: the link which delivered the advertisement (`link`), and the advertisement itself (`adv`), which you constructed as a list in `make_dv_advertisement`. (You can ignore the marshalling of the advertisement into a packet and the corresponding unmarshalling back to the list, but if you're curious, you can see how `send_advertisement()` and `process_advertisement()` do these tasks.) You can use `link.cost` to determine the cost associated with the link.

The result of `integrate()` is the current `self.routes` and `self.spcost` tables, which as mentioned before, are the routing table and table of shortest path costs. The underlying rule you should use is the Bellman-Ford update rule, as described in lecture: remember to add the link cost to the cost reported by the advertisement that comes from the neighbor at the other end of the link.

The `integrate()` step should take care to update the current route for a destination under the following conditions:

- a. If the cost in an advertisement for the destination plus the link cost is smaller than the cost of the current route.
- b. If the cost in an advertisement for the destination changes, and the advertisement comes on the link corresponding to the current-best route.
- c. Depending on how you design your DV protocol, you may have to take care of a subtle (but important) issue in `integrate()`: if you find that a previous advertisement that came from `fromnode` contained a destination, and you are using the corresponding link as the route to the destination, and the current advertisement does not mention the destination, then you have to assume that the destination is no longer reachable via fromnode. Otherwise, it is likely that your protocol may not be correct. You should also note that not every design requires this check; it all boils down to how you send your routing advertisements.

Please note that if your protocol decides that there is no route to a destination, then the route to that destination must be set to `None` and the `spcost` for that destination should be set to `self.INFINITY`. Failure to do so may cause some of the tests to fail.

You are NOT required to implement mechanisms to alleviate or eliminate the "counting-to-infinity" problem, such as split-horizon, poison-reverse, or even path vector, but are welcome to do so if you like!

When first debugging your code, it's useful to use the `-g` option so you can run the simulation for a small number of timesteps and then click on nodes to see their routing tables. When you're ready to see if your code passes the tests, run the program without the `-g` option. Please note that the verification is not exhaustive: passing these tests does not necessarily imply that your code is 100% correct!



Upload code for Task 1:

Browse...

(points: 10)

## Python Task #2: Link-state (LS) routing

Useful download link:

[PS9 2.py](#) -- template file for this task

The LS protocol uses the `LSRouter` class, which is derived from the `Router` class. The `self.routes` and `self.spcost` tables are identical to the DV case. The `LSRouter` class adds two new variables to `Router`:

- `self.LSA`, a dictionary that maps a Router address to a list `[seqnum, (n1,c1), (n2,c2), (n3,c3), ...]`, where the Router address is the *originator* of the link-state advertisement (LSA), seqnum is the current sequence number of the LSA from that node, and the `(n_i, c_i)` tuples are the currently live neighbor address and link cost from the `LSRouter` sending the LSA. Note that this LSA does not have a "origin\_address" field; we simply get that from the source field in the packet, and you don't need to worry about it (in practice, implementations will explicitly include such a field in the LSA). Note that this dictionary contains the set of nodes and links in the graph known to the node.

So here, it maintains the whole list to get somewhere

`self.LSA.get(u)` returns the last LSA update originating from Router `u` that this Router (`self`) knows about. It has the format `[seqnum, (n1,c1), (n2,c2), ...]` where `seqnum` is the sequence number at `u` when it originated the LSA and each `n_i` is a neighbor of `u` (that `u`'s HELLO protocol considered to be "live" when it generated the LSA numbered `seqnum`), and `c_i` is the cost of the link from `u` to `n_i`.

- `self.LSA_seqnum`, which is the sequence number for the LSA generated by the Router. It increments by 1 on each successive advertisement.

See the lecture notes for the details of how an LS protocol works. Each Router periodically sends its currently live links (which we maintain in the `self.neighbors` dictionary, as explained earlier when we discussed the HELLO protocol). Each Router also *re-broadcasts*, along all its links, an LSA packet that it receives via a neighboring Router, containing an LSA originating at some other Router. This re-broadcast is done only once per LSA; to ensure this property, the Router checks the sequence number of the incoming LSA to make sure that it is larger than the last LSA heard from that originating Router. These periodic LSA broadcasts are done every `ADVERT_INTERVAL` time slots (50 by default in NetSim); the re-broadcasts are done when a Router receives a new LSA (using the sequence number check).

So a packet is re-broadcast for + wide

For this task, the file you have to extend is [PS9 2.py](#):

Your task is to write the following two functions, which form the core of any LS protocol:



V<sub>o</sub>E WP: |E|+V

(points: 1)

If the link failure causes the network topology to become disconnected, will your answer to the question above change? Explain.

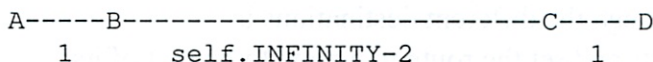
I have no clue

like a new network?

(points: 1)

In testing a network path with very high cost:

↑ what are these?



a correct implementation of the distance-vector implementation says that there is no route at node A for destination D, and vice versa, but in fact there is a path between these two nodes. Why does the protocol say that?

since its so large  
 its too easy

(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

1. `make_ls_advertisement()`: consult the list of neighbors, `self.neighbors`, that are currently "live" and return a list of `[(neighbor1, linkcost1), (neighbor2, linkcost2), (neighbor3, linkcost3), ...]`. This function is called by `send_lsa`, which marshalls this LSA into the packets, and then sends one packet each along each link.

Note that the `self.neighbors` dictionary mapping a Link to a `(timestamp, address, linkcost)` tuple will be useful in constructing the LSA; as mentioned above in the discussion of the HELLO protocol, `neighbors` keeps track of currently "live" neighboring Routers.

2. `run_dijkstra(nodes)`: Use Dijkstra's algorithm to produce `self.routes[dest]` for all `dest` in `nodes`, as well as `self.spcost[dest]`. The topology information for the network (graph) is available in the `self.LSA`, whose format was described above. The set of `nodes` currently reachable from the Router is passed as the argument to `run_dijkstra()`.

*to find shortest path*

There is one important issue that you need to watch out for in the steps of `run_dijkstra()` that will set the routing table entries, `self.routes`, for various destinations. At the Router, as you go through the different destinations in non-decreasing order of shortest-path costs and set the route to a node to be that of its parent in the shortest path tree. If the parent is `self.address` (i.e., the Router running the algorithm), then you should remember to set the route to the link connecting the Router (`self`) to the destination. You can use the `getlink()` method for this purpose.

Please note that if your protocol decides that there is no route to a destination, then the route to that destination must be set to `None` and the `spcost` for that destination should be set to `self.INFINITY`. Failure to do so may cause some of the tests to fail.

When first debugging your code, it's useful to use the `-g` option so you can run the simulation for a small number of timesteps and then click on nodes to see their routing tables. When you're ready to see if your code passes the tests, run the program without the `-g` option. Please note that the verification is not exhaustive: passing these tests does not necessarily imply that your code is 100% correct!

Upload code for Task 2:

Browse...

(points: 10)

Suppose that a link failure has just occurred in the network. Give a rough estimate for how long it will take for the distance vector (Task #1) and link state (Task #2) protocols to correctly update the routing tables in *all* the nodes. Briefly explain your answer. Give your answer in terms of the HELLO and ADVERT intervals, not a numeric value.

*depends on width of graphs  
2x width / takes longer*



## 6.02 coding

4/16

Should be kinda straightforward

Main part is interfacing w/ project

DV

1. What to collect:

- its whole table

- integrate adds link cost

Should  
be in  
black

What is sp in sp cost?

(can't print stuff out in GUI mode?)

- can do text

Costs

{ 'A' : 'setf' }

{ 'A1' : 0 }

2 So ~~remove~~ set cost to  $\infty$  and remove costs

how deal w/ named arrays:  
↑ associative

also dictionary

② No that is link - not route!  
will deal w/ later

3. get ad

- integrate seems like one at a time

[ ('B' : 'self'), ('B' : 0) ]

So add 0 + link cost to dict

~~send~~ set B to that cate

if smaller

Or if cost change

See instructions

adv [0][0] is source

should make get cost function

(otes is keyed by dest

(costs is too)



③

2. So redo 2 that keys are dest - not links

Sp cost - singular!

Oh in get\_cost if not there - return  $\infty$

kinda working in 30 min

now look at c)

in routes save what as values: links?

1. For sending just dest + cost

- not links

- so diff format

↑ this prob breaking

Nice now that c issue

— but how do we just know on DV

Oh must drop all old routes

But I overwrite them

I don't get it

(4)

Do graphical

Why does step 1 turn red?

green = hello  
red = advertisements  
blue = data

It never learns about further than neighbors

So make ads is wrong

---

So after that why does not send ad

- only in 50 times

Not sending more

Sends ~~an~~ costs

Oh I am only integrating first one!

How to get key?

- can't

---

Ran a bunch more now  
but still wrong

⑤ starting to see more

Seems ~~not~~ right - but why do tests fail

had F - dest 6 expected C got 6  
but that's right, unless test does not match



Coding

Thinks I have too many print <sup>In lab</sup> statements  
Must take failures into account in integrate

I remember seeing that - but not understanding

Ok I get what they want now

So, now if got None

- but only if got that route before

- over that link

Actually a ~~More~~ Infinity

But if it is same link should update!

Failure never received!  
- only ~~on~~ on qvi  
Took out print statements to see better

dests is empty

(wish could run functions - debugger)

Ok dest - on link never worked

Did not need extra section like John thought (c)  
Done!

② Going to dinner

would I have seen that bug w/o  
- failure on link not working

Prob not, but hungry





2

1. Make Ls - ad  
- package up ads

Where does it store ads?

No need to process

Self.LSA.get(v)

iterate through

only sends ~~for~~ neighbors  
its well links

So each node will have list of everyone's neighbor  
as

Do this first

What is neighbors list?

its link

but for what neighbor

Or link  $\rightarrow$  timestamp, address, cost tuple

↑  
is this neighbor

do I have to add seq # - no

③

Now clear for 2 - Dijkstra I think  
produce routes for all dest in nodes  
spcost

Use LSA data

Go through diff dest in shortest path  
cost and set root as parent

Or set  $\infty$

Where to start:

Check slides

- node in nodeset with smallest path
- which is current on list and since 0

Self, spcost + routes auto generated

so start somewhere then follow min path



so stop at every cost

How to find v's neighbors

9

in self.LSA get

what format is this?

how remove a certain item from list?

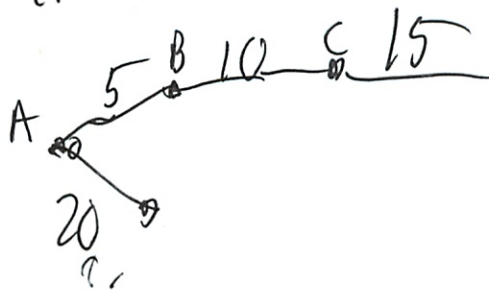
a.remove()

(Just following spec - not really thinking)

How get link cost?

~~Oh~~ Oh duh given

What if



Set what as route?

a link?

Sans get link()

route = node that its parent

- Oh so just 0 or routes [0]
- the whole path or 1 node?
- a link?



⑤ Reread  
~~Shank~~ sounds like I node that is its parent  
- ~~node~~ so  $v$ , not  $cates[v]$

Yeah! On wp it says  $v$   
Email in?

---

Still does not work

why self address seem wrong

Oh or it  $v$  not  $v$

No  $v$  since parent - then use that node

---

Or is problem something else?

They say ~~get link~~  $\uparrow$  not defined?  
- yes it is

---

Response: is a link - the next link to use

- so from LSA?

get link(neighbor)

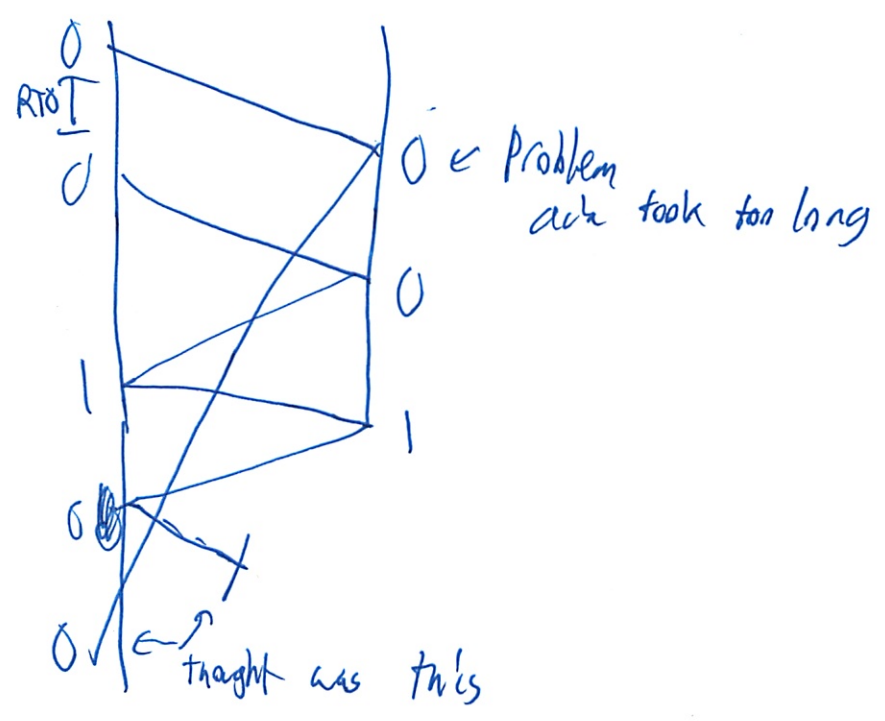
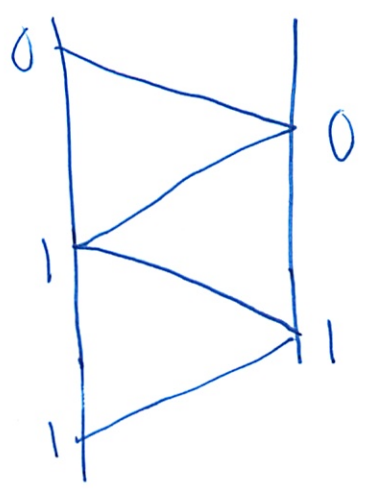
So link used to get to  $v$  from LSA

So use that for root node

But what about other times?

6

b)  $k$  (packet #) replaced by 0 or 1  
- Receiver receives this



---

13) Adaptively estimate RTT