Nathan Fox

CS 460

February 17, 2015

Network Layer Homework

1) P1

a. A datagram architecture would be better in a case where routers might fail fairly often. If a router in an existing VC fails during data transfer, the network would have to create a new VC to finish data transfer. Datagram architectures allow more flexibility because the path isn't decided beforehand, so if a router fails on one path then an earlier router will just send the data on a different path.

b. A VC architecture would be better because during the VC setup phase, the network layer can reserve resources such as bandwidth along the path of the VC.

c. A datagram architecture would have more control traffic overhead in this situation. A VC architecture would have very little control traffic because it would set up all of the VCs once, and the routes would never have to be changed or updated. A datagram architecture would force routers to frequently update their forwarding tables even though the routes don't need to be updated.

2) P9

The minimal number of time slots is 2. First, send X from the top input port, Y from the middle, and Z from the bottom (assuming any input queue scheduling order). Second, send X from the middle input port and Y from the bottom port.

The largest number of time slots needed is 3. This is the case if there is HOL blocking.

3) P10

a.

11100000 00 0

11100000 01000000 1

11100000 01000001 2

11100001 01 2

otherwise 3

b.

- The address starts with 110, which falls into "otherwise" and is forwarded to link 3.

- The address starts with 11100001 01000000, which is forwarded to link 1. (It also matches a prefix that forwards to link 2, but the prefix that forwards to 1 is longer.)

- The address starts with 11100001 1, which falls into "otherwise" and is forwarded to link 3.

4) P13

Subnet 1 - 223.1.17.192/26 (last 6 bits can support up to 64 interfaces)

Subnet 2 - 223.1.17.0/25 (last 7 bits can support up to 128 interfaces)

Subnet 3 - 223.1.17.128/28 (last 4 bits can support up to 16 interfaces)

5) P19

It would be broken up into 4 fragments:

#1 680 bytes of data identification = 422 offset = 0 flag = 1

#2 680 bytes of data identification = 422 offset = 85 flag = 1

#3 680 bytes of data identification = 422 offset = 170 flag = 1

#4 360 bytes of data identification = 422 offset = 255 flag = 0

6) P26

Step N' D(t) D(u) D(v) D(w) D(y) D(z)

\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_

0 x inf inf 3,x 6,x 6,x 8,x

1 xv 7,v 6,v 6,x 6,x 8,x

2 xvu 7,v 6,x 6,x 8,x

3 xvuw 7,v 6,x 8,x

4 xvuwy 7,v 8,x

5 xvuwyt 8,x

6 xvuwytz

7) P31

i = positive infinity

x y z x y z

x| 0 3 4 0 3 4

y| i i i 3 0 6

z| i i i 4 6 0

x| i i i 0 3 4

y| 3 0 6 3 0 6

z| i i i 4 6 0

x| i i i 0 3 4

y| i i i 3 0 6

z| 4 6 0 4 6 0

8) P37

a. eBGP

b. iBGP

c. eBGP

d. iBGP

9) P38

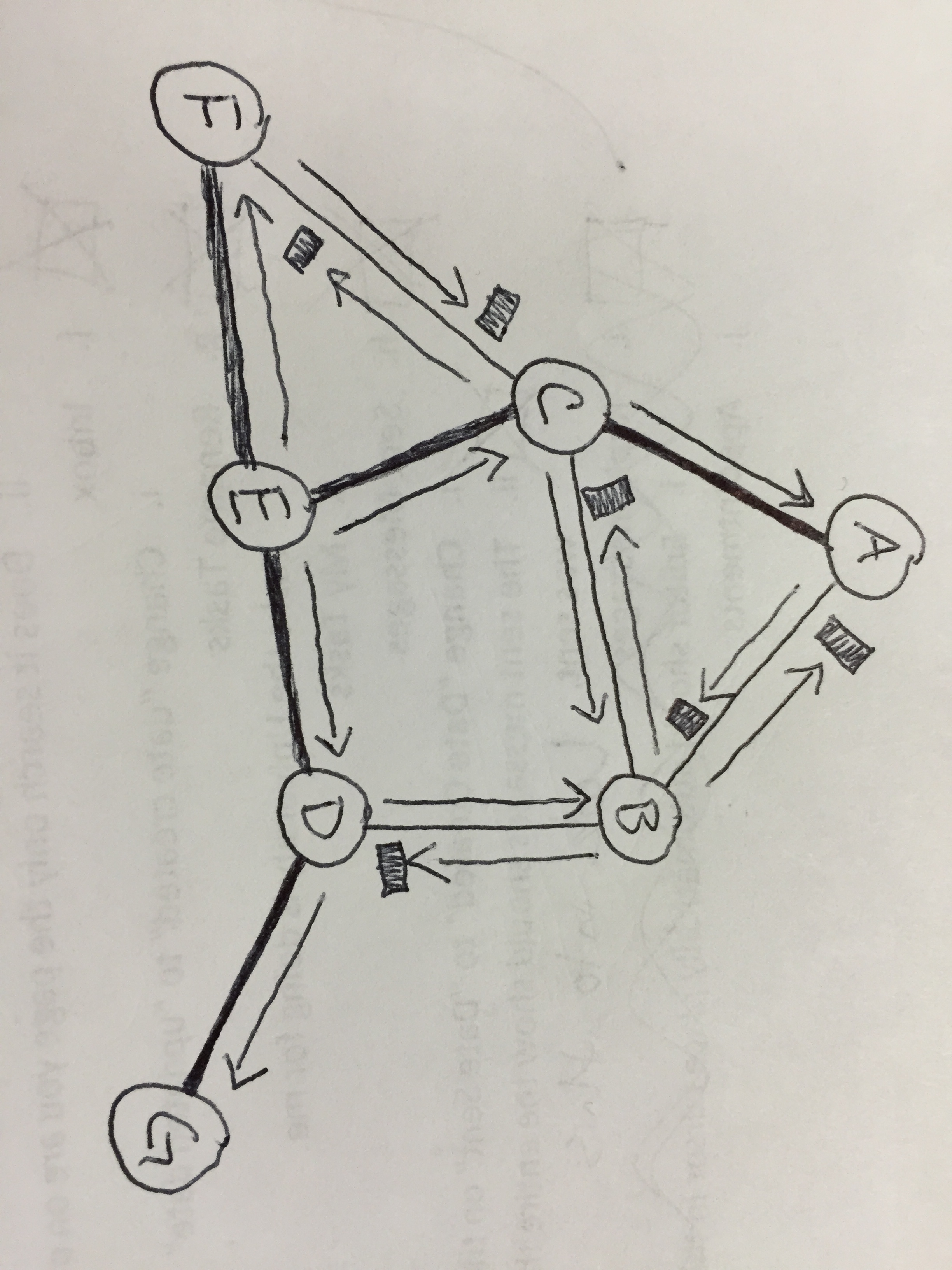
a. I will be I1, because the shortest path from 1d to x goes through I1

b. I will be I2, because the shortest path from 1d to x now goes through I2

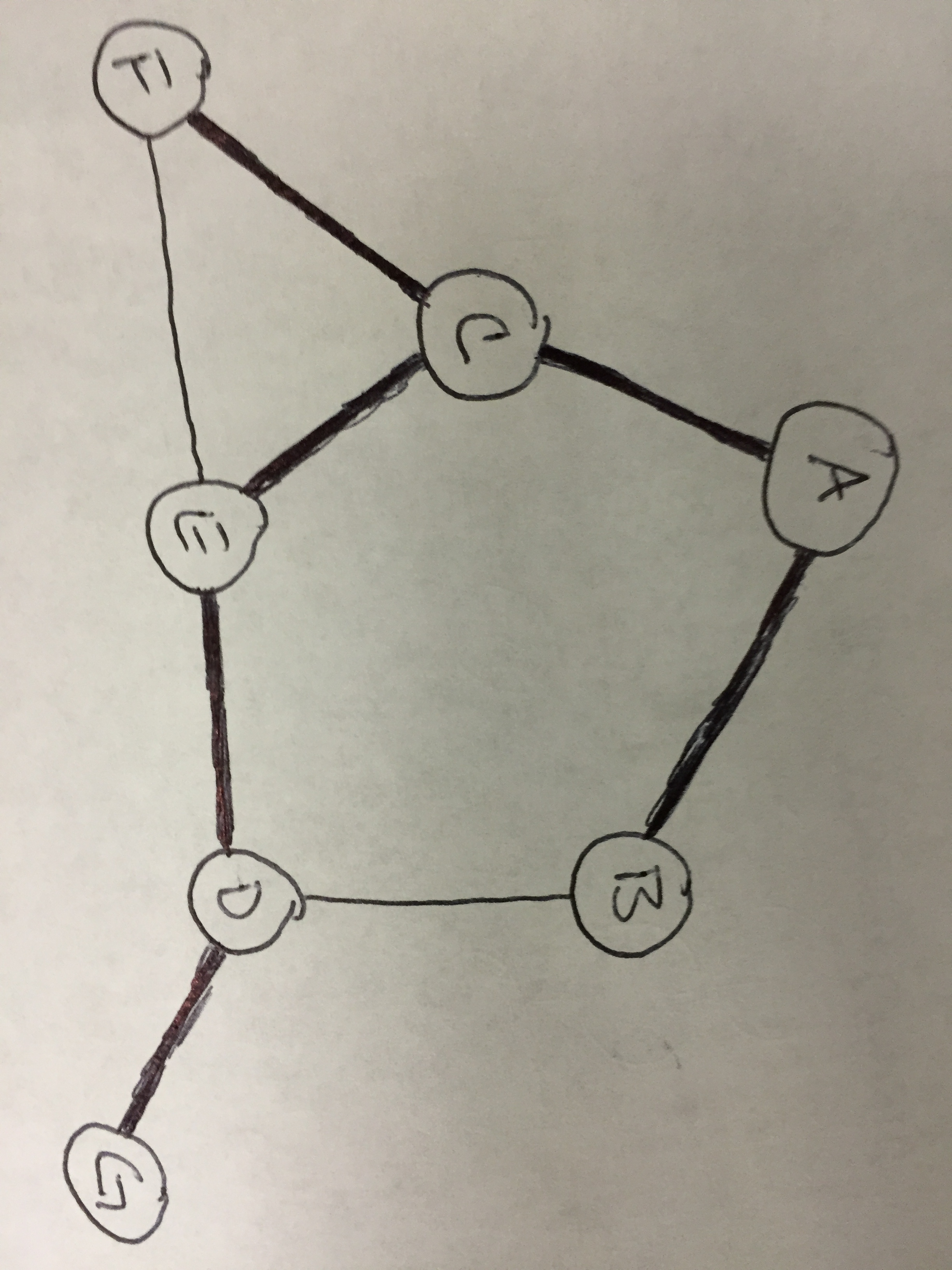
c. I will be I1, because in terms of AS hops, the shortest path from 1d to x now goes through I1

10) P39

The west-coast peering point in C's AS might not advertise C's coast-to-coast route to B's west-coast peering point. If C doesn't advertise the route, B won't know about the route, and B will have to use its own coast-to-coast route to get the data to the east coast.



11) P47

12) P49

The resulting tree, with a cost of 6, is a minimum spanning tree. The lowest possible cost to span a graph of 7 nodes is 6.

13) P55

Multicast addresses are 28 bits, so there are 2^28 possible multicast addresses. The chances of two groups randomly choosing the same address are 1/(2^28), or a .00000037% chance. If 1,000 users randomly select addresses, there is about a 0.1859% chance of collision. Here's the code I wrote to get these values:

import random

possible\_values = float(2\*\*28)

print 1/possible\_values

iterations = 1000

print sum([reduce(lambda x,y:x\*y,[(1.0-(m/possible\_values)) for m in xrange(1,n+1)],1)\*((n+1)/possible\_values) for n in xrange(0,iterations-1)])