Discussion 8:

A3 Help, Routing Protocol Questions

By Joey Buiteweg

(some slide idea credits to Yiwen Zhang)

Discussion Outline

By the end of this discussion we will:

- Know what to be careful of in assignment 3
- Be able to reason about scenarios involving Distance-Vector routing

A3 Hints and Help

What You Should Test (i.e what we will test)

WTP-base

Your sender and receiver should work given:

- Timeouts/Large amounts of latency (think 100s of ms)
- Packet loss
- Packet corruption
- Multiple file transfers for 1 receiver lifespan
- Large binary or text file transfers (War and Piece, The Dictionary, etc.)

Script your tests and don’t hand inspect files

- Use things like cmp, sdiff, etc.

What You Should Test (i.e what we will test)

WTP-opt

Your sender and receiver should work given:

- Same conditions as last slide
- ACKs should have precisely the same seq as what was sent
- Packets that have been ACK’d should not be retransmitted

i.e make sure the expected “optimizations” are observable

Do not use TCP sockets, the AG knows when you are doing this!

Mininet Tips

Mininet has a Python API, it’s how we setup topologies

```
class Lossy(Topo):

    def __init__(self, **opts):
        Topo.__init__(self, **opts)
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')

        __bw = opts["bw"]
        __delay = opts["delay"]
        __loss = opts["loss"]

        self.addLink(h1, s1, bw=__bw, delay=__delay,  loss=__loss)
        self.addLink(h2, s1, bw=__bw, delay=__delay,  loss=__loss)
```

Mininet Tips

Mininet allows us to run commands on hosts

Lets us automate starting different processes

```
# get host from topology, can also use net[‘h1’]
h1 = net.get(‘h1’) # net is from calling Mininet(**kwargs)
h1_cmd = “echo hello from h1”

# Two options for running command on host
# Option 1: host.cmd function, wrapper around host.popen
output = h1.cmd(h1_cmd) # output == “hello from h1”

import subprocess
# Option 2: host.popen, more fine grained control, very useful!
h1_proc = h1.popen(h1_cmd, stdout=subprocess.PIPE,
                   stderr=subprocess.STDOUT)
h1_done = h1_proc.poll() # Can see if process is done or not
h1_stdout, h1_stderr = h1_proc.communicate() # h1_stderr may be None, see
docs
# Note: same interface that’s provided by subprocess.Popen(), in this case
we # are running on specific mininet host!
```

Python Subprocess Example

```
from subprocess import call
h1 = net.get('h1') # net = Mininet(topo=Lossy(**kwargs), …)
h2 = net.get('h2')
tmp = ["./wSender", <other flags here>]
h1_cmd = " ".join(tmp)
tmp = ["./wReceiver", <other flags here>]
h2_cmd = " ".join(tmp)
net.start()
h2.cmd(h2_cmd)  # start receiver

h1.cmd(h1_cmd)  # start sender
cmd = ["cmp", self.infile, self.outdir + "/" + "FILE-" + str(i + 1)]
# compare outputs
if call(cmd) != 0:
     print "Input and output file differ!"
     net.stop()
     exit(1)
```

Demo

Lecture Based Questions

Q1 Link-State vs Distance-Vector

True/False:

Link-State (LS) routing involves broadcasting its local knowledge of the network to everyone

True, uses Dijkstra's for computation (OSPF). Why not Floyd-Warshall?

Conversely, Distance-Vector routing involves telling only neighbors about its global view True, uses Bellman-Ford for computation (RIP)

Both routing methods involve finding least-cost paths to all other nodes

True, allows easy metric to avoid loops

Q2 Distance Vector Properties

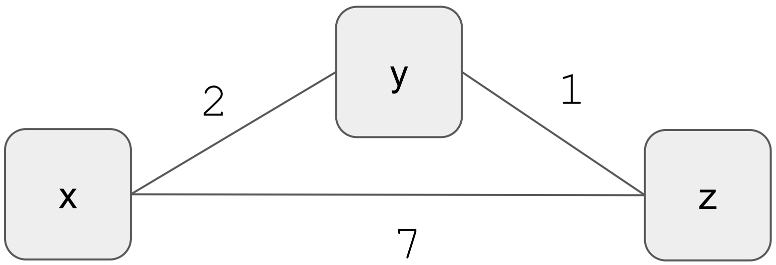
Yes/No:

For DV routing, will the count-to-infinity problem occur if we **decrease a link's cost?**

No. Loops aren't caused by decreasing link cost

What about if we **connect two previously unconnected nodes?**

No. Loops potentially result from a removing a link



Q3 Distance Vector situations

Consider this network fragment

w’s least-cost path to **u** (not shown) of 5

y has least cost path to **u** of **6**

Complete paths from **w** and **y** to **u** not shown

All links have strictly positive costs

What is **x**’s distance vector for **w**, **y**, and **u**?

$d_x(w) = 2$

$d_x(y) = 4\ (w)$

$d_x(u) = 7$

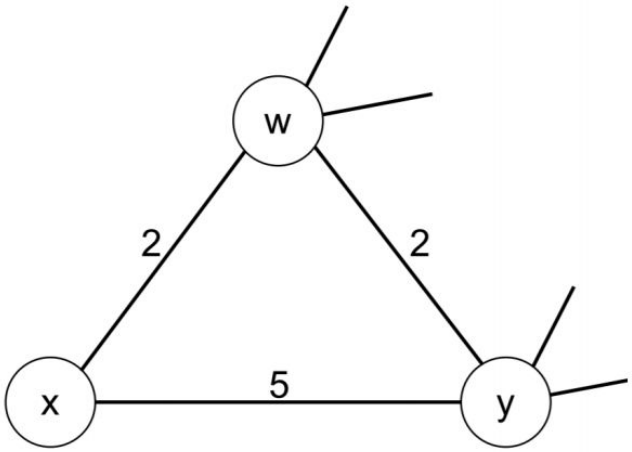
via w

- Let
  - $d_x(y)$:= cost of least-cost path from x to y
- Then
  - $d_x(y) = \min_v \{c(x, v) + d_v(y)\}$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

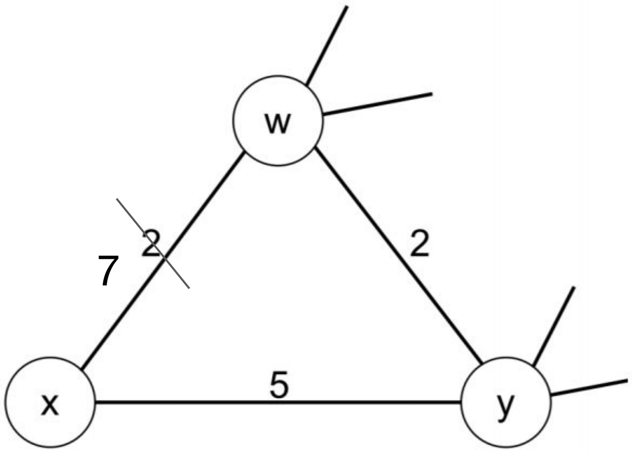


$$D_w(u) = 5$$

$$D_y(u) = 6$$

Q4.1 Distance Vector situations

Give a link-cost change for either **c(x, w)** or **c(x, y)**
such that **x** will inform its neighbors
of a new least-cost path to **u**



$$D_w(u) = 5$$
$$D_y(u) = 6$$

Change c(x, w) to be larger than 6, so that we go through y
or
Make c(x, y) < 1 (no link cost)

$d_x(w) = \cancel{2}\ 7$
$d_x(y) = \cancel{4}\ 5\,(y)$
$d_x(u) = \cancel{7}\ 11$
via $\cancel{w}$
y

- Let
  - $d_x(y)$:= cost of least-cost path from x to y
- Then
  - $d_x(y) = \min_v \{c(x, v) + d_v(y)\}$

cost from neighbor v to destination y
cost to neighbor v
min taken over all neighbors v of x

Q4.2 Distance Vector situations

Give a link-cost change for either **c(x, w)** or **c(x, y)** such that **x** will **not** inform its neighbors of a new least-cost path to **u**

Make c(x, y) > 0

Make c(x, w) < 7

$d_x(w) = 2$

$d_x(y) = 4$

$d_x(u) = 7$

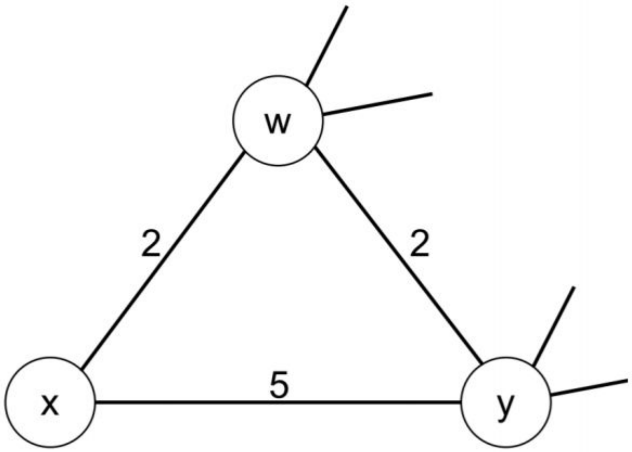
via w

- Let
  - $d_x(y)$:= cost of least-cost path from x to y
- Then
  - $d_x(y) = \min_v \{c(x, v) + d_v(y)\}$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x



$$D_w(u) = 5$$

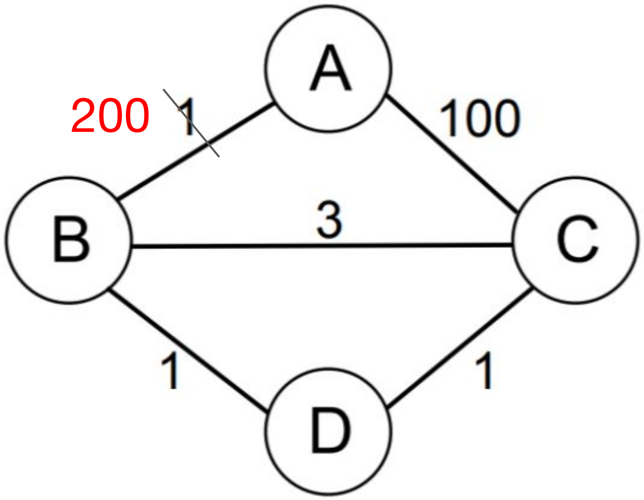
$$D_y(u) = 6$$

Q5 Poisoned Reverse

Consider this network fragment

Assume the following events:

- DV is used with **poisoned reverse**
- Routing state has stabilized
- **c(A, B)** goes from 1 to 200 very suddenly



Will count to infinity occur?

No:

In general, if **z** routes to **x** through **y**, then **z** will advertise to **y** that $\mathbf{d_z(x)}$ is infinite.

In this case, if **D** routes to **A** through **B**, then **D** will advertise to **B** that $\mathbf{d_D(A)}$ is infinite

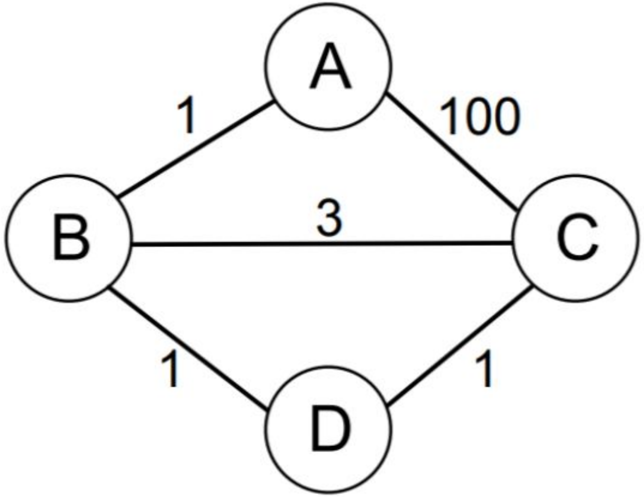
The following slides are not ground truth (just my best guess as to how things would go)

Q5 Poisoned Reverse

At D	A	B	C	D
B	1(A)	0	2(D)	1(D)
C	inf	inf	0	1(D)
D	**2(B)**	1(B)	1(C)	0

At B	A	B	C	D
A	0	1(B)	inf	inf
B	1(A)	0	2(D)	1(D)
C	3(D)	2(D)	0	1(D)
D	**inf**	1(B)	1(C)	0

Read table as from (row) to (col)

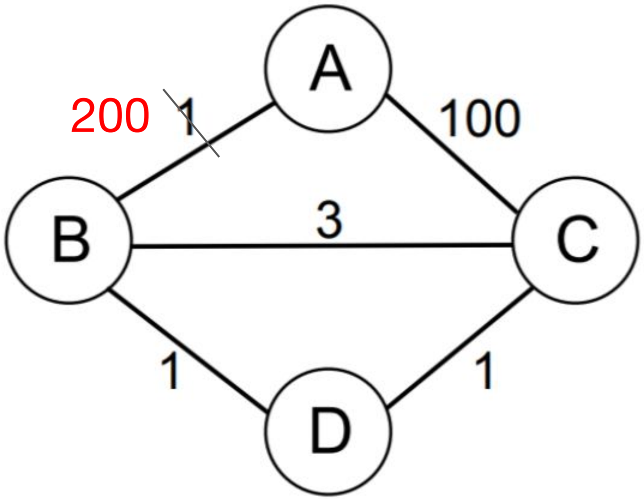


Q5 Poisoned Reverse - some time later

At D	A	B	C	D
B	**103(C)**	0	3(C)	4(C)
C	100(A)	3(B)	0	1(D)
D	**101(C)**	1(B)	1(C)	0

B still thinks $d_D(A)$ = inf

Eventually B will go to A through D though



At B	A	B	C	D
A	0	103(C)	inf	inf
B	**103(C)**	0	3(C)	4(C)
C	100(A)	3(B)	0	1(D)
D	**inf**	1(B)	1(C)	0