Lecture 18: Network Layer
Link State and Distance Vector Routing

COMP 332, Spring 2018

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - homework 6 due today by 11:59p
   - homework 7 posted
     - but will likely make minor clarifications to programming part

2. Control plane
   - link state routing
   - distance vector routing
   - compare link state vs. distance vector

3. Network programming
   - raw sockets and byte packing

Control Plane

LINK STATE ROUTING

Dijkstra's algorithm

Link state: i.e., network topology, link costs

- known to all nodes, accomplished via link state broadcast
  - msg sent to every other node in network
- all nodes have same global info

Computes least cost paths

- from one "source" node to all other nodes
- obtain forwarding table for that node

Given path, put 1st hop router for each dst in forwarding table

Iterative

- after k iterations, know least cost path to k destinations
  - if n nodes, loop n times

Dijkstra’s algorithm

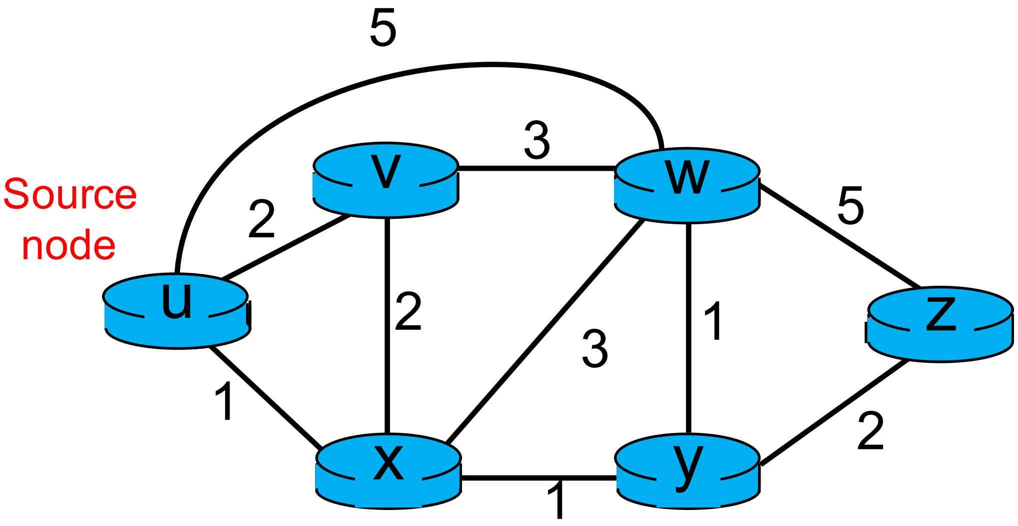
c(x,y): link cost from node x to y

D(v): current cost from source u to dst node v

p(v): predecessor node along path from source u to v

N': set of nodes whose least cost path definitively known

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1						
2						
3						
4						
5						



Initialization

```
N' = {u}
for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞
```

Dijkstra’s algorithm

c(x,y): link cost from node x to y

D(v): current cost from source u to dst node v

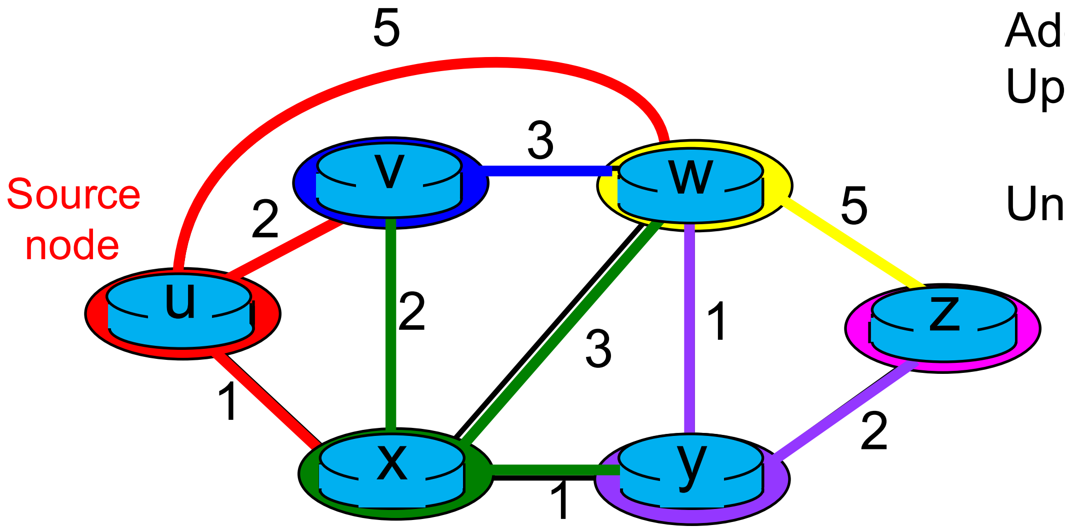
p(v): predecessor node along path from source u to v

N': set of nodes whose least cost path definitively known

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Loop

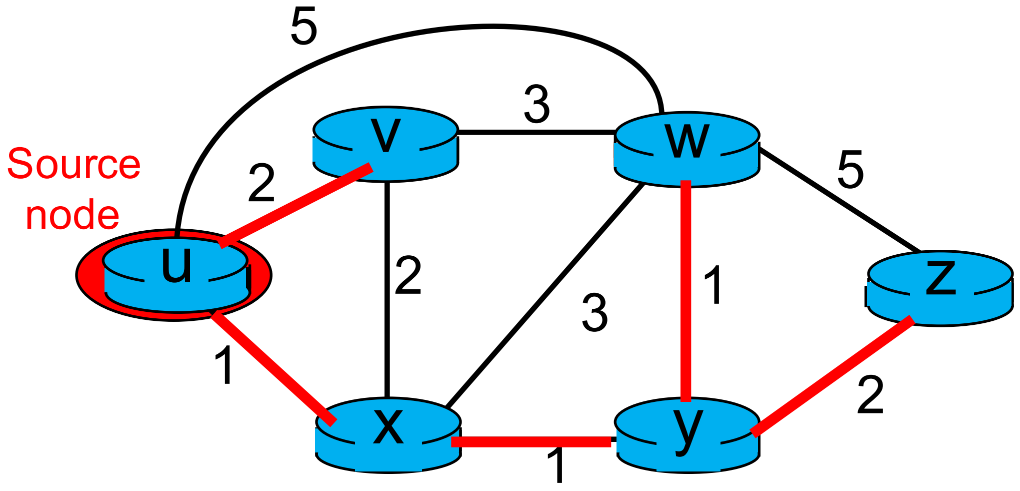
```
Find w ∉ N' s.t. D(w) is min
Add w to N'
Update D(v) for all neighbors v ∉ N' of w
    D(v) = min( D(v), D(w)+c(w,v) )
Until all nodes in N'
```



Dijkstra's algorithm

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Resulting shortest path tree



Forwarding table at u

dst	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Algorithm complexity with n nodes

Each iteration: need to check all nodes not in N’

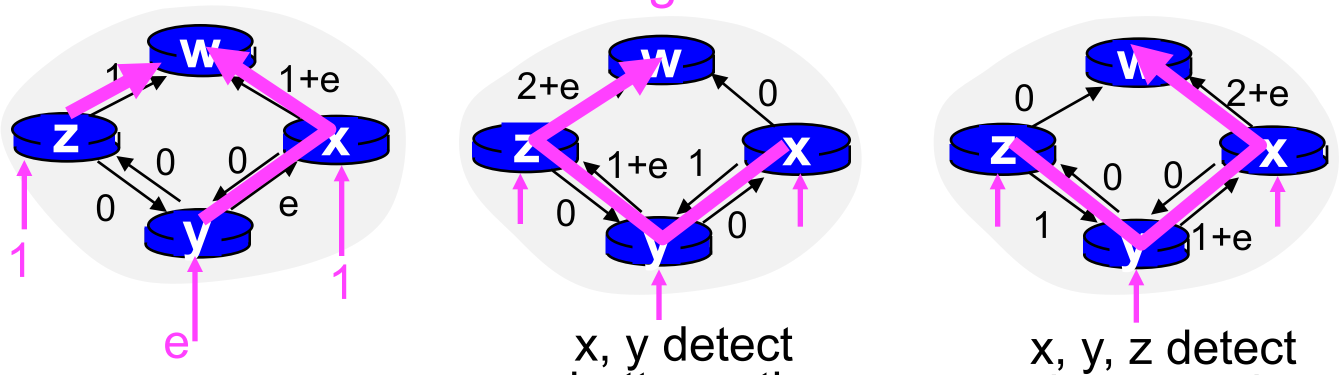
- n(n+1)/2 comparisons: $O(n^2)$, more efficient implementations possible

Network is dynamic

- link goes down: link state broadcast
- router goes down: remove link and all nodes recompute

Oscillations possible

- when congestion or delay-based link cost



initially

… recompute routing

… recompute

x, y detect better path

x, y, z detect better path

Need to prevent routers from synchronizing computations:

Have routers randomize when they send out link advertisements

Control Plane

DISTANCE VECTOR ROUTING

Distance vector routing

Distance vector (DV)

- vector of best known costs from router to each dst and link to use

Each node x maintains

- Link cost from x to each neighbor v
  - c(x,v)
- x’s own DV
  - $D_x(y)$: estimate of least cost path from x to node y
  - $D_x = [D_x(y): y \in N]$
- DV for each nbr v
  - $D_v(y)$: estimate of least cost path from neighbor v to node y
  - $D_v = [D_v(y): y \in N]$

Each node periodically sends its own DV to neighbors

- rather than link state costs

Bellman-Ford equation to update DV estimates

Uses dynamic programming

- break problem into simpler sub-problems
- solve each sub-problem once and store solution

Bellman-Ford equation

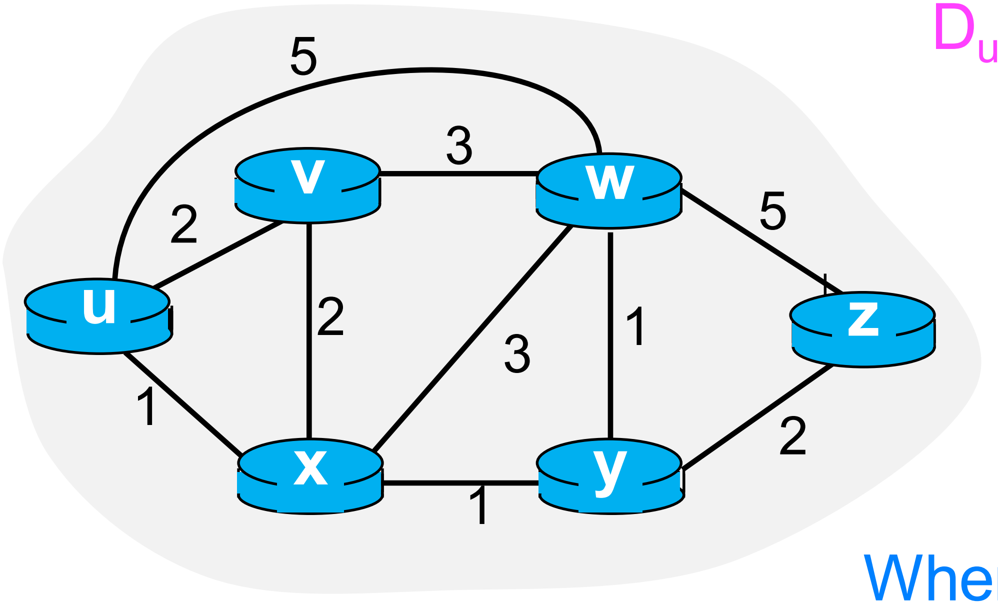
$D_x(y)$:= cost of least-cost path from x to y

$$D_x(y) = \underbrace{\mathit{min}_{v}}_{\mathit{min}\text{ taken over all neighbors v of x}} \{ \underbrace{c(x,v)}_{\text{cost to neighbor v}} + \underbrace{D_v(y)}_{\text{cost from neighbor v to destination y}} \} \text{ for each node } y \in N$$

When x receives new DV estimate from neighbor

- x updates its own DV using B-F equation

Example: compute min cost path from u to z



Bellman-Ford equation

$$
\begin{aligned}
D_u(z) = \min \{ & c(u,v) + D_v(z), \\
& c(u,x) + D_x(z), \\
& c(u,w) + D_w(z) \} \\
= \min \{ & 2 + 5, \\
& 1 + 3, \\
& 5 + 3 \} \\
= 4 &
\end{aligned}
$$

Where

$D_v(z) = 5$, $D_x(z) = 3$, $D_w(z) = 3$

Node achieving minimum is next hop in shortest path

- put in forwarding table

Distance vector algorithm run at each node x

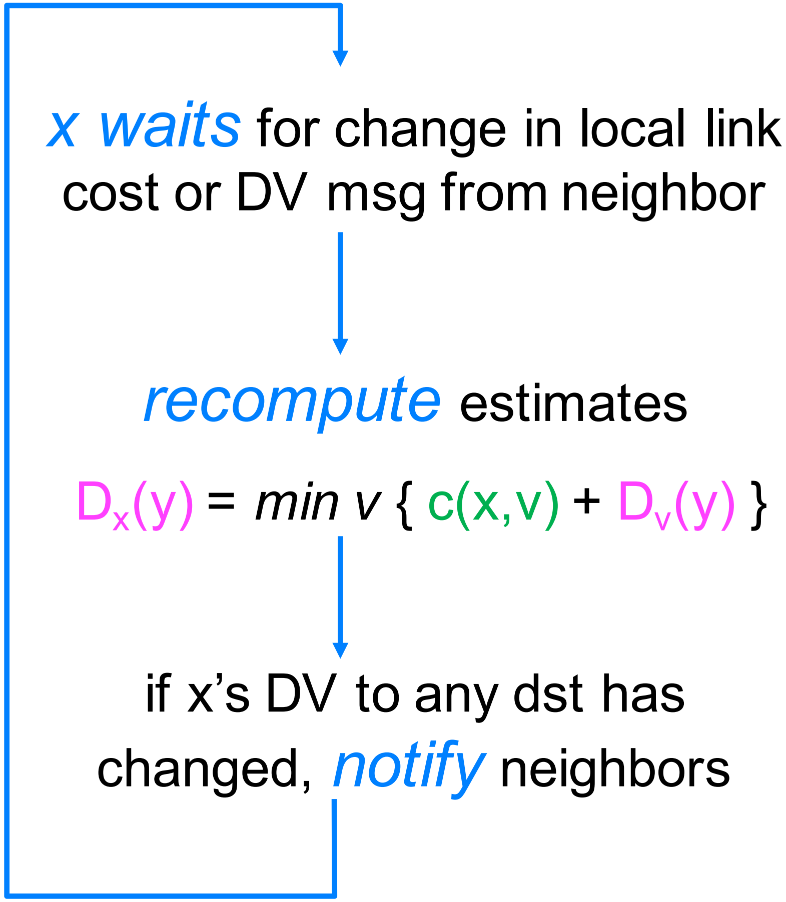
Initialization

```
For all dst $y \in N$
    if y is nbr of x
      $D_x(y) = c(x, y)$
    else
      $D_x(y) = \infty$

For each nbr w and dst $y \in N$:
    $D_w(y) = \infty$

Send x's DV to all nbrs w
    $D_x = [D_x(y) : y \in N]$
```

Loop



x waits for change in local link cost or DV msg from neighbor

recompute estimates

$D_x(y) = \mathit{min}\ v\ \{\ c(x,v) + D_v(y)\ \}$

if x's DV to any dst has changed, *notify* neighbors

Q: when does loop terminate?

When no more changes

$D_x(y) = \min\{c(x,y)+D_y(y), c(x,z)+D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y)+D_y(z), c(x,z)+D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

Node x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Node y

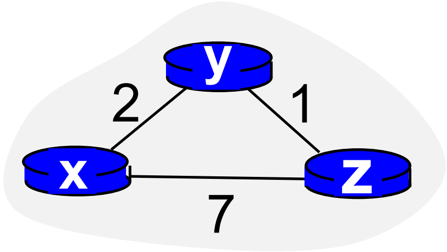
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

Node z

cost to

from	x	y	z
x	∞	∞	∞
y		∞∞	∞
z	7	1	0



time

$D_x(y) = \min\{c(x,y)+D_y(y), c(x,z)+D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y)+D_y(z), c(x,z)+D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

Node x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

Node y

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

Node z

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

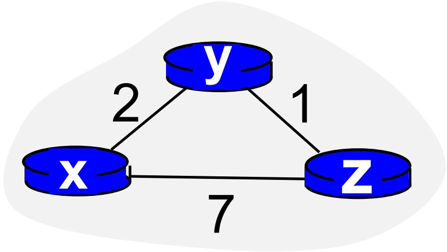
from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

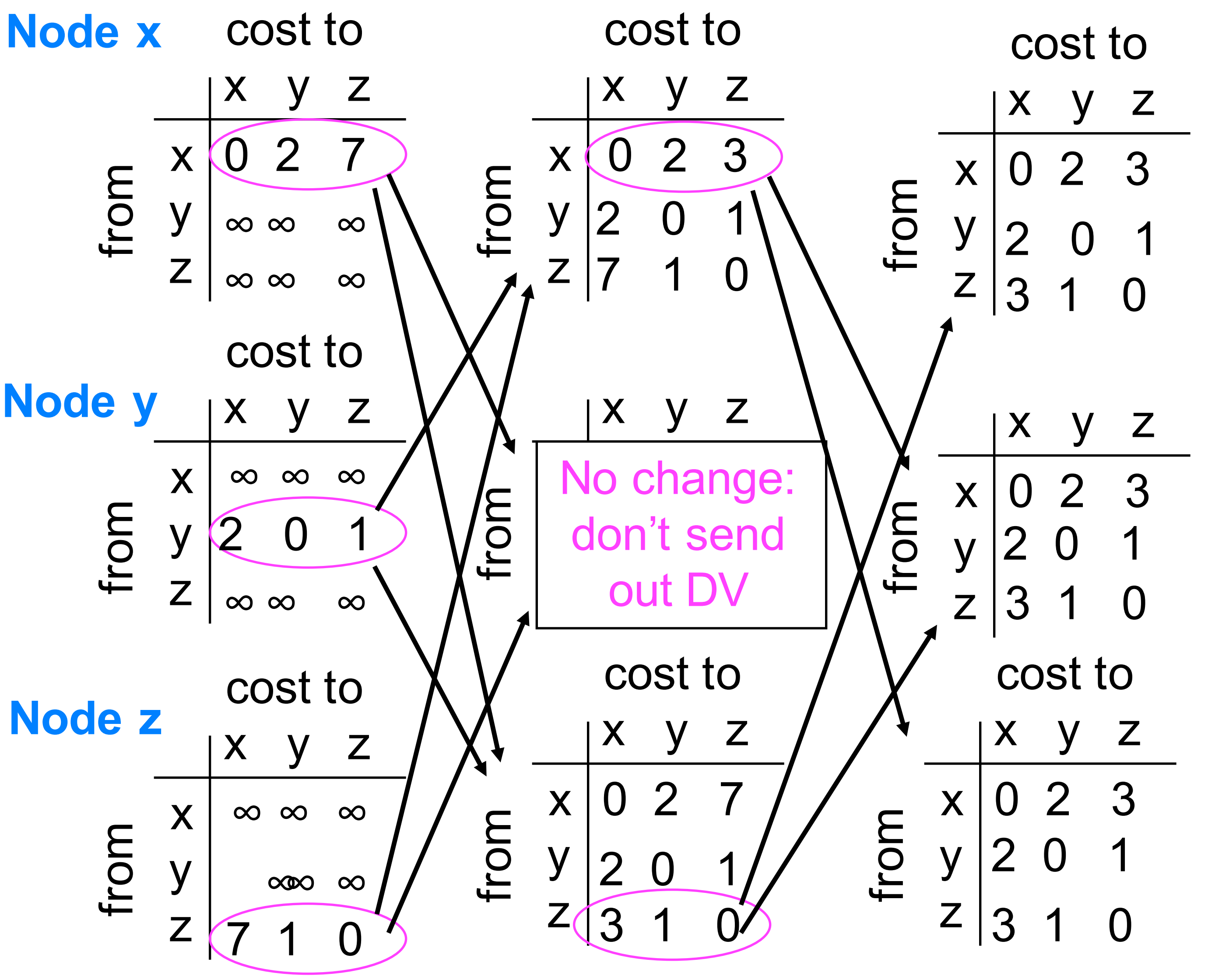
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

$D_z(x) = \min\{c(z,x)+D_x(x), c(z,y)+D_y(x)\}$
$= \min\{7+0 , 1+2\} = 3$



time



Node x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node y

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from	x	y	z

No change: don’t send out DV

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node z

cost to

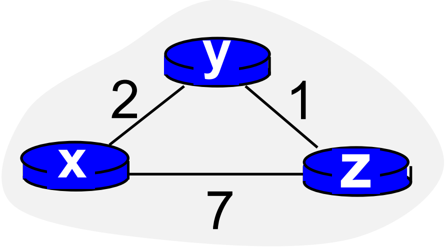
from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

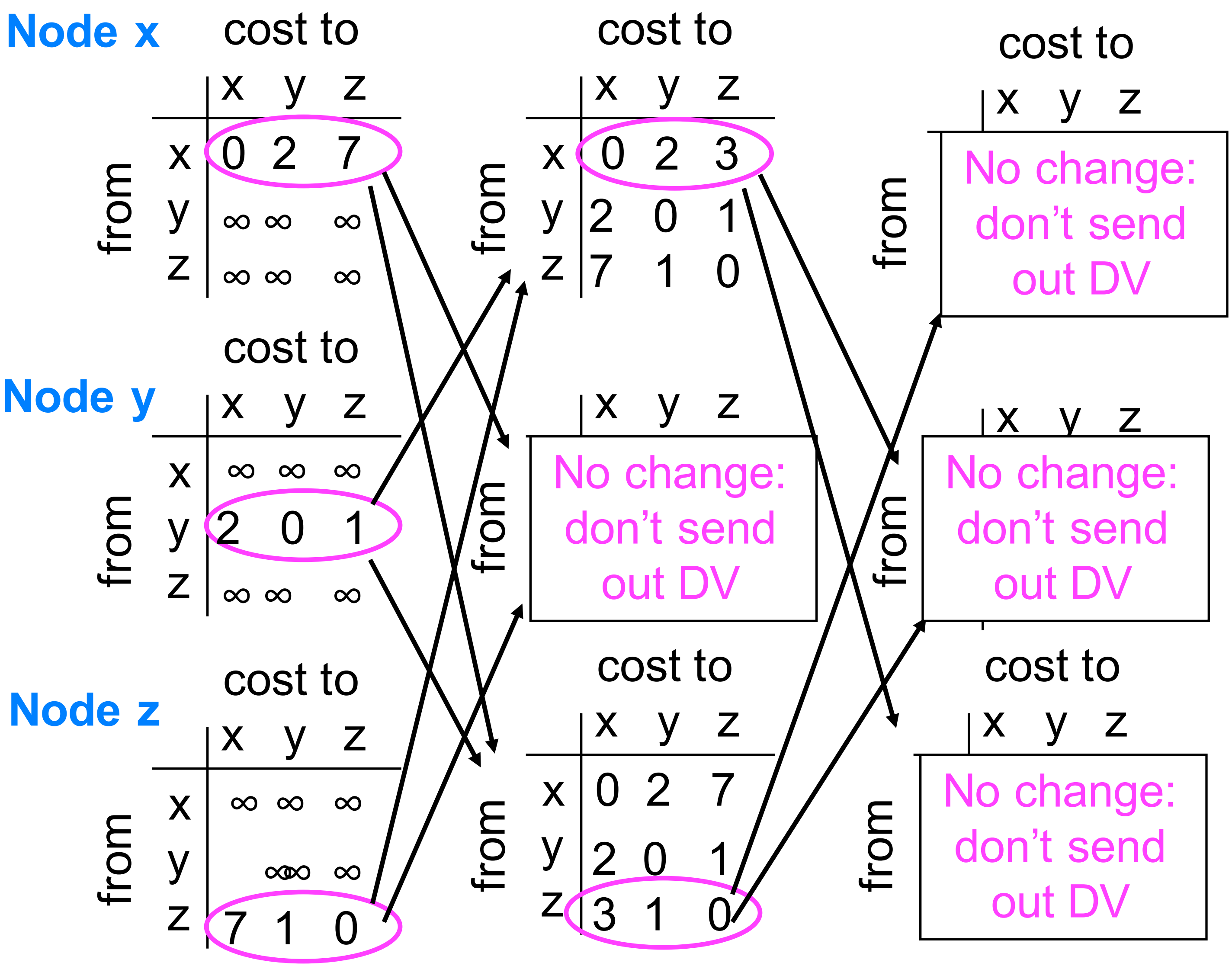
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0





Node x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z

No change: don’t send out DV

Node y

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from	x	y	z

No change: don’t send out DV

from	x	y	z

No change: don’t send out DV

Node z

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

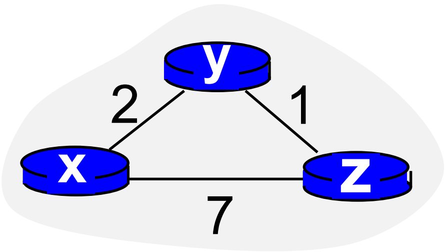
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

from	x	y	z

No change: don’t send out DV

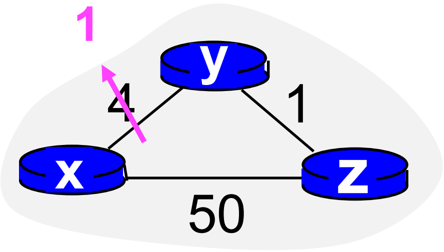


DONE

time

Node detects local link cost change

1. Updates routing info
2. Recalculates DV
3. If DV changes, notify neighbors



Good news travels fast

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x*, sends its neighbors its DV

t_2 : *y* receives *z*'s update, updates its distance table. *Y*'s least costs do *not* change, so *y* does *not* send a message to *z*

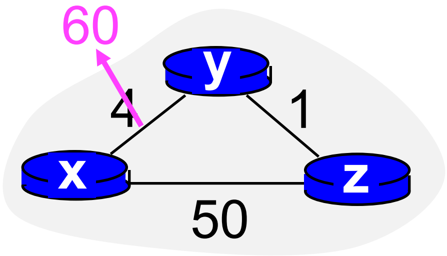
Bad news travels slow

Count to infinity problem

- 44 iterations before algorithm stabilizes

Intuitively

- when z tells y it has a path to x, y has no way of knowing that z is using y on its path



cost to

Y	x	y	z
x	0	4	3
y	4	0	1
z	5	1	0

from

cost to

Y	x	y	z
x	0	4	3
y	6	0	1
z	5	1	0

from

$$D_y(x) = \min\{c(y,x)+D_x(x),\ c(y,z) + D_z(x)\}$$
$$= \min\{60+0,\ 1+5\} = 6$$

→ Routing Loop

$$D_z(x) = \min\{c(z,x) + D_x(x),\ c(z,y) + D_y(x)\}$$
$$= \min\{50+0\ ,\ 1+6\} = 7$$

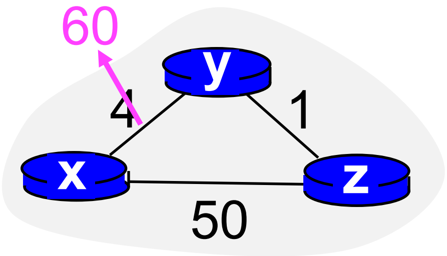
→ Count-to-infinity

Problem arises because y still expects z can get to x with cost of 5

A proposed solution: poisoned reverse

If Z routes through Y to get to X

- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)



cost to

Y	x	y	z
from x	0	4	3
y	4	0	1
z	∞	1	0

$$D_y(x) = \min\{c(y,x)+D_x(x),\ c(y,z)+D_z(x)\}$$
$$= \min\{60+0,\ 1+\infty\} = 60$$

Q: Will this completely solve count to infinity problem?

- no, only for 2 node loops

Another proposed solution: hold time

- don't process route updates for period of time after route retraction
- ameliorates problem but does not solve

Distance vector routing summary

Easy to implement

- likely you will implement for hw8 :-)

Distributed

- x doesn’t compute paths in isolation
- requires route info (path costs) computed by neighbors

Iterative

- x updates its DV whenever
  - local link costs change
  - DV update received from nbr

Asynchronous

- updates, exchanges happen asynchronously

Self-terminating

- x stops updating DV when no more changes received

Control Plane

LINK STATE VS. DISTANCE VECTOR ROUTING

Message complexity

n nodes
E links

Link state

- O(nE) messages sent
  - every node floods its link state message out over every link in network to reach every node
- smaller messages
  - message size depends on the number of neighbors a node has
  - any link change requires a broadcast

Distance vector

- # of messages depends on convergence time which varies
  - nodes only exchange messages between neighbors
- larger routing update messages
  - message size is proportional to the number of nodes in the network
  - if link changes don't affect shortest path, no message exchange

Speed of convergence

n nodes
E links

Link state

- $\Sigma_{i=1}^{n-1} = n(n+1)/2 = O(n^2)$
  - search through n-1 nodes to find min, recompute routes
  - search through n-2 nodes to find min, recompute routes
  - …
- converges quickly but may have oscillations
  - route computation is centralized
  - a node stores a complete view of the network

Distance vector

- slow to converge and convergence time varies
  - route computation is distributed
- may be routing loops, count-to-infinity problem

What happens if router malfunctions?

n nodes
E links

Link state

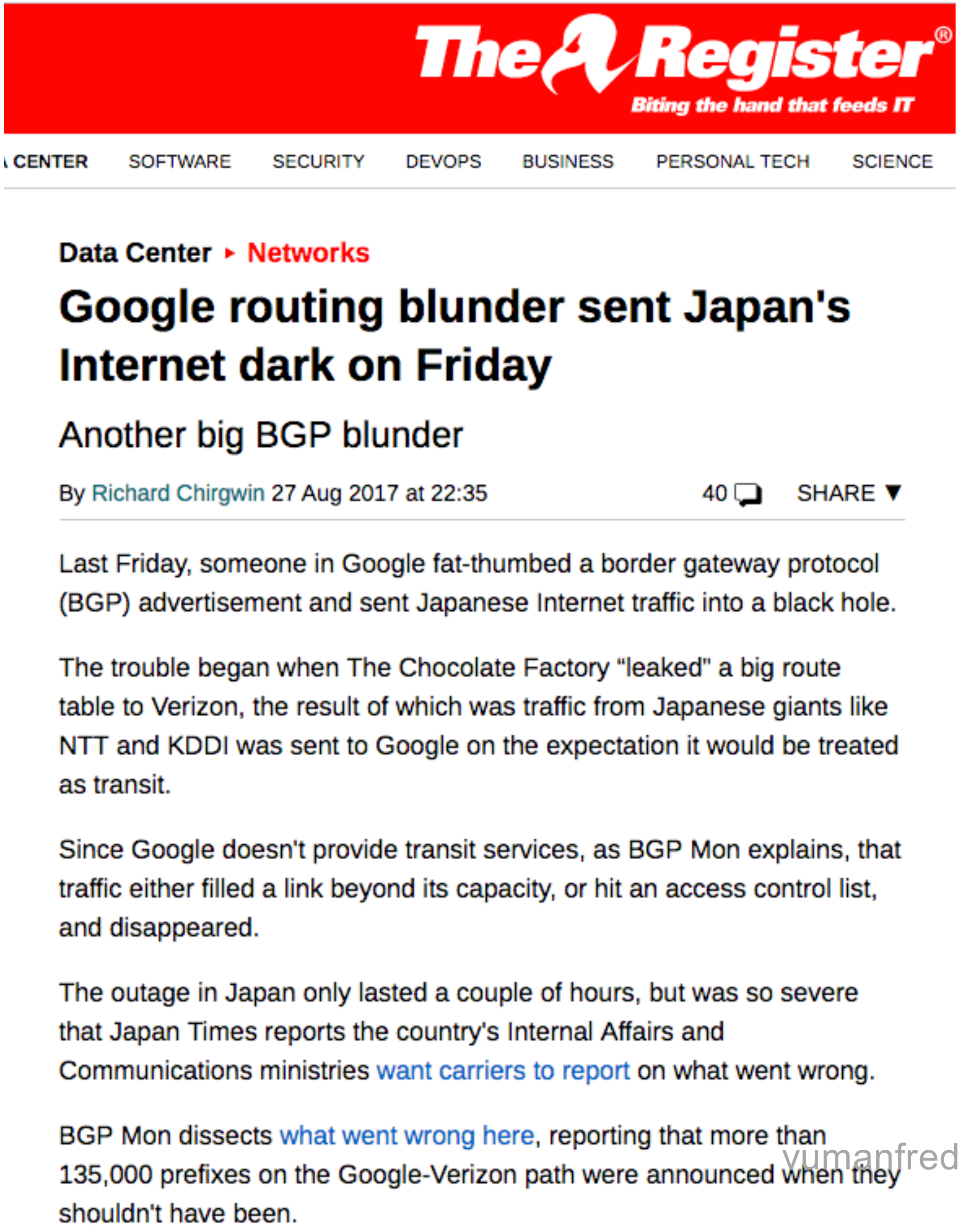
- node can advertise incorrect link cost
- each node computes only its own table

Distance vector

- DV node can advertise incorrect path cost
- each node’s DV used by others: errors propagate through network

Both have strengths and weaknesses.
One or the other is used in almost every network

Routing blackholes



The Register®
Biting the hand that feeds IT

CENTER SOFTWARE SECURITY DEVOPS BUSINESS PERSONAL TECH SCIENCE

Data Center ▸ Networks

Google routing blunder sent Japan's Internet dark on Friday

Another big BGP blunder

By Richard Chirgwin 27 Aug 2017 at 22:35 40 SHARE ▼

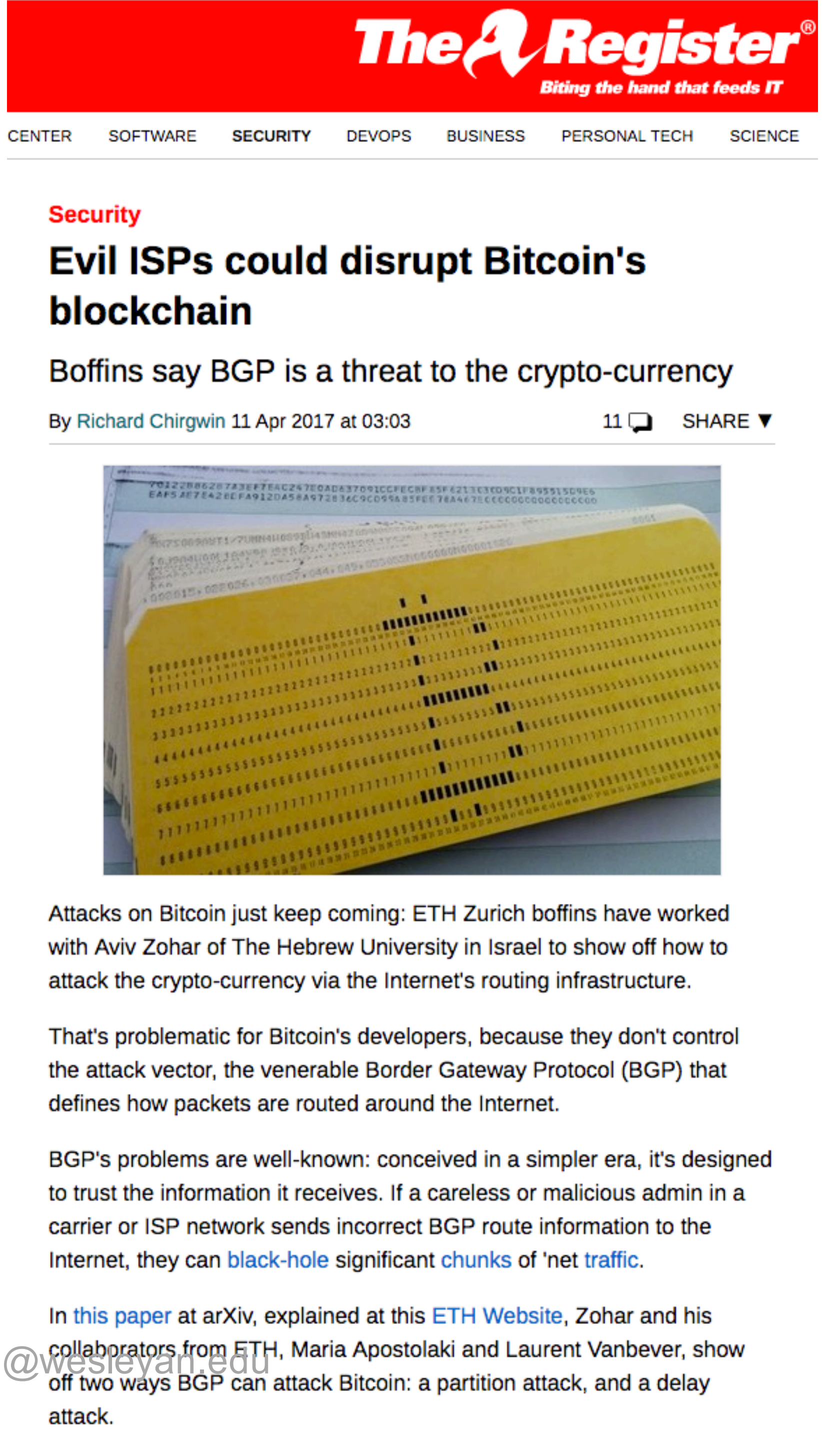
Last Friday, someone in Google fat-thumbed a border gateway protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

The trouble began when The Chocolate Factory "leaked" a big route table to Verizon, the result of which was traffic from Japanese giants like NTT and KDDI was sent to Google on the expectation it would be treated as transit.

Since Google doesn't provide transit services, as BGP Mon explains, that traffic either filled a link beyond its capacity, or hit an access control list, and disappeared.

The outage in Japan only lasted a couple of hours, but was so severe that Japan Times reports the country's Internal Affairs and Communications ministries want carriers to report on what went wrong.

BGP Mon dissects what went wrong here, reporting that more than 135,000 prefixes on the Google-Verizon path were announced when they shouldn't have been.



The Register®
Biting the hand that feeds IT

CENTER SOFTWARE SECURITY DEVOPS BUSINESS PERSONAL TECH SCIENCE

Security

Evil ISPs could disrupt Bitcoin's blockchain

Boffins say BGP is a threat to the crypto-currency

By Richard Chirgwin 11 Apr 2017 at 03:03 11 SHARE ▼

Attacks on Bitcoin just keep coming: ETH Zurich boffins have worked with Aviv Zohar of The Hebrew University in Israel to show off how to attack the crypto-currency via the Internet's routing infrastructure.

That's problematic for Bitcoin's developers, because they don't control the attack vector, the venerable Border Gateway Protocol (BGP) that defines how packets are routed around the Internet.

BGP's problems are well-known: conceived in a simpler era, it's designed to trust the information it receives. If a careless or malicious admin in a carrier or ISP network sends incorrect BGP route information to the Internet, they can black-hole significant chunks of 'net traffic.

In this paper at arXiv, explained at this ETH Website, Zohar and his collaborators from ETH, Maria Apostolaki and Laurent Vanbever, show off two ways BGP can attack Bitcoin: a partition attack, and a delay attack.

Network Programming

RAW SOCKETS

Raw sockets

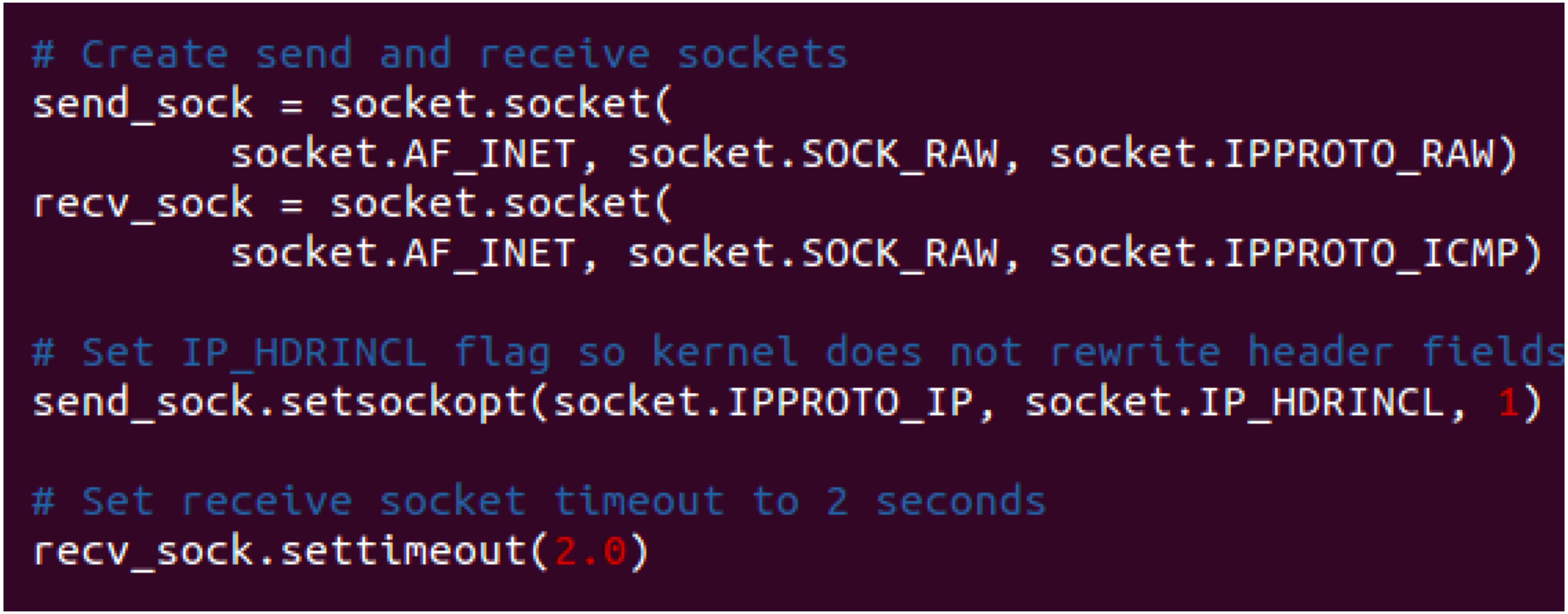
Take bytes put into socket and push out of network interface

- no IP or transport layer headers added by operating system!

Lets you create your own transport and network layer headers

- set field values as you choose
  - e.g., time-to-live fields

Raw sockets



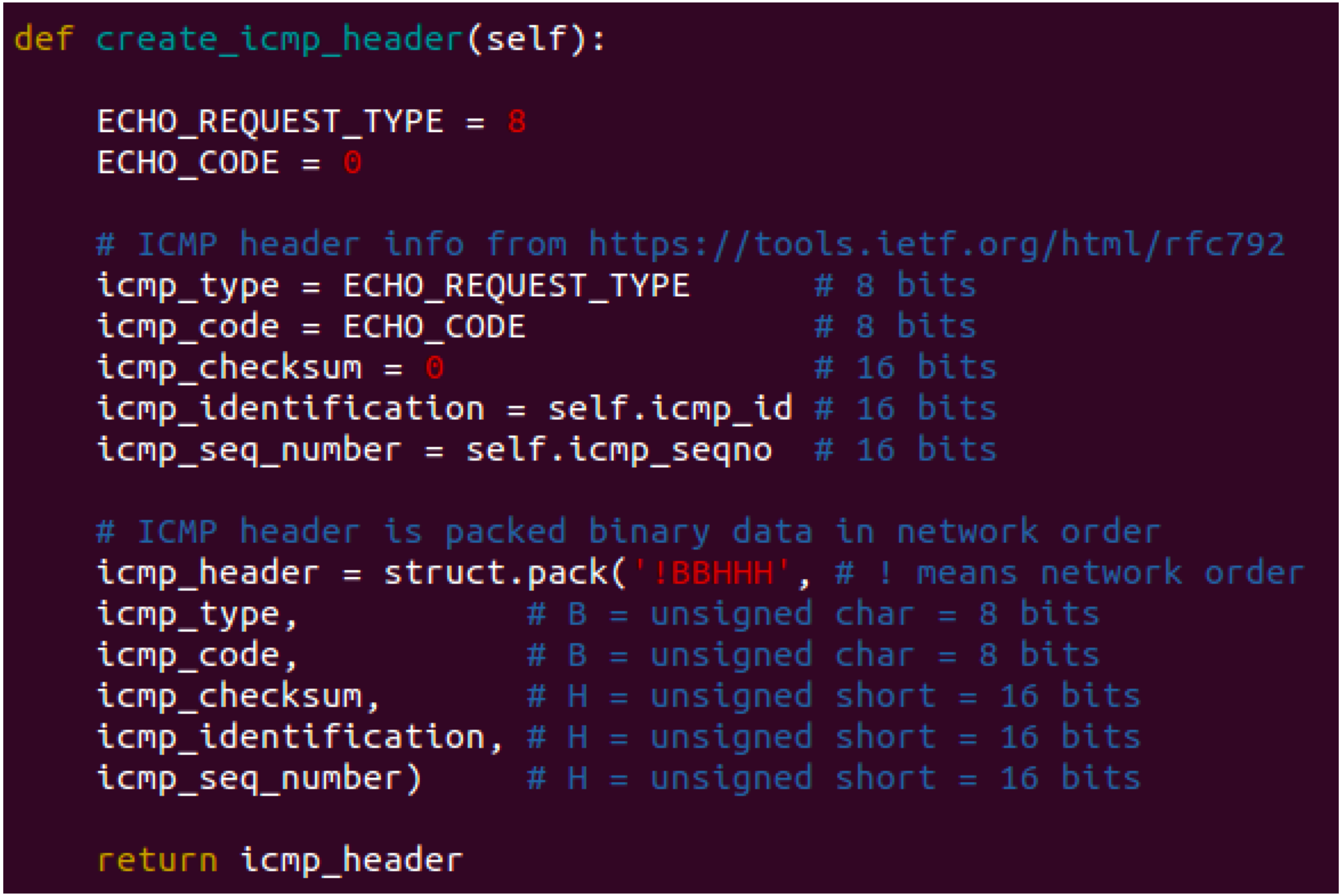
```
# Create send and receive sockets
send_sock = socket.socket(
        socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_RAW)
recv_sock = socket.socket(
        socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)

# Set IP_HDRINCL flag so kernel does not rewrite header fields
send_sock.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# Set receive socket timeout to 2 seconds
recv_sock.settimeout(2.0)
```

https://docs.python.org/3/library/socket.html

Byte packing and structs



```
def create_icmp_header(self):

    ECHO_REQUEST_TYPE = 8
    ECHO_CODE = 0

    # ICMP header info from https://tools.ietf.org/html/rfc792
    icmp_type = ECHO_REQUEST_TYPE      # 8 bits
    icmp_code = ECHO_CODE              # 8 bits
    icmp_checksum = 0                  # 16 bits
    icmp_identification = self.icmp_id # 16 bits
    icmp_seq_number = self.icmp_seqno  # 16 bits

    # ICMP header is packed binary data in network order
    icmp_header = struct.pack('!BBHHH', # ! means network order
    icmp_type,           # B = unsigned char = 8 bits
    icmp_code,           # B = unsigned char = 8 bits
    icmp_checksum,       # H = unsigned short = 16 bits
    icmp_identification, # H = unsigned short = 16 bits
    icmp_seq_number)     # H = unsigned short = 16 bits

    return icmp_header
```

https://docs.python.org/3/library/struct.html