Chapter 4
Network Layer: Data Plane
Part 2

Instructor: Zhuozhao Li

Lab: Qing Wang

Department of Computer Science and Engineering

Miscellaneous

- PA3 release today

- Project specification released
  - https://docs.qq.com/sheet/DSUNqSmJYVnJMb2Nk?groupUin=0laSTOMmleqHpIIKdKvaLw%253D%253D&tab=BB08J2
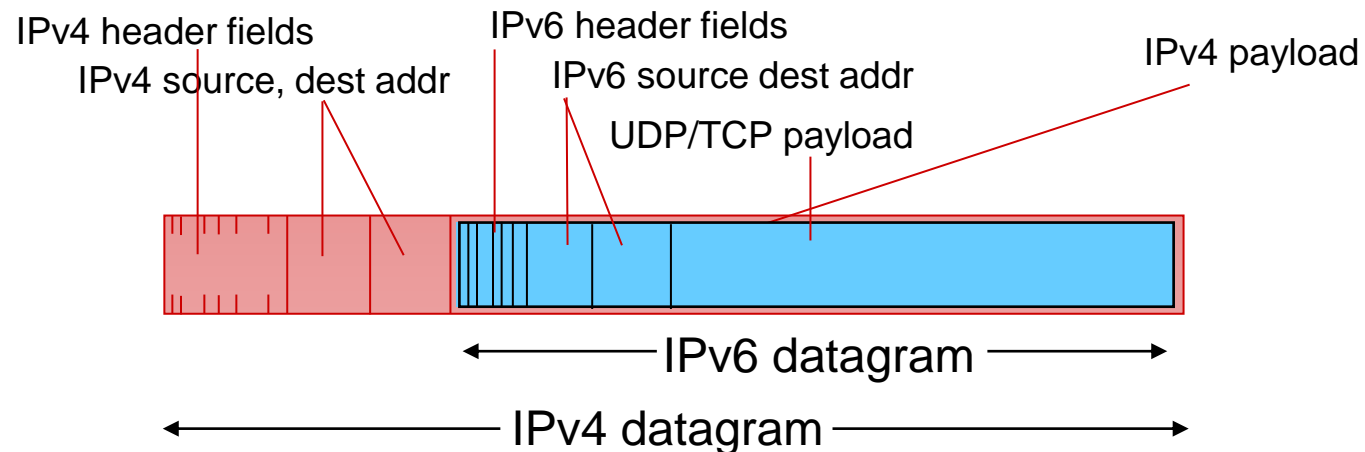 - https://github.com/Nancyzxy/CS305-proj

Quick review

- Network-layer
  - Queue drop policy and scheduling
  - IP datagram structure: 20-60 Bytes
  - IP addressing: CIDR, subnet part, host part
  - DHCP
  - NAT
  - IPv6

```
ipconfig /renew
```

|						
843	13.098103	10.16.95.67	172.18.19.31	DHCP	358 DHCP Request	- Transaction ID 0xa4f8c833
845	13.133405	172.18.19.31	10.16.95.67	DHCP	342 DHCP ACK	- Transaction ID 0xa4f8c833

Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?

- tunneling: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers ("packet within a packet")
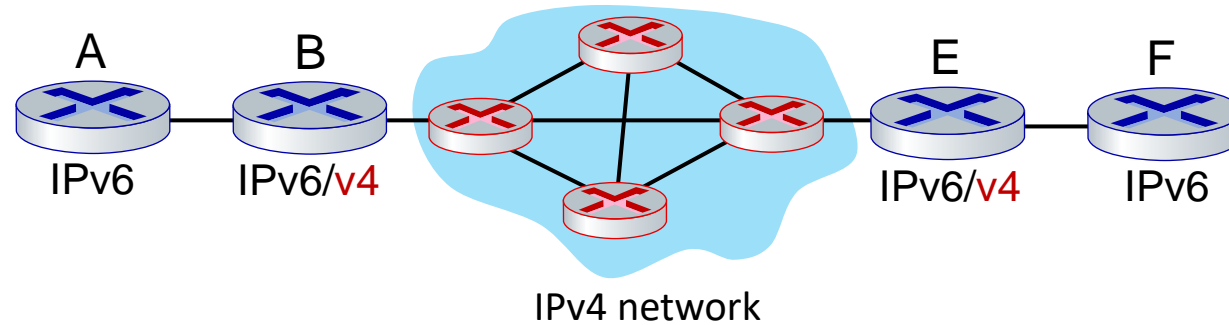 - tunneling used extensively in other contexts (4G/5G)

IPv4 header fields
IPv4 source, dest addr

IPv6 header fields
IPv6 source dest addr
UDP/TCP payload

IPv4 payload

IPv6 datagram

IPv4 datagram

Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

Ethernet connects two IPv6 routers

IPv6 datagram

Link-layer frame
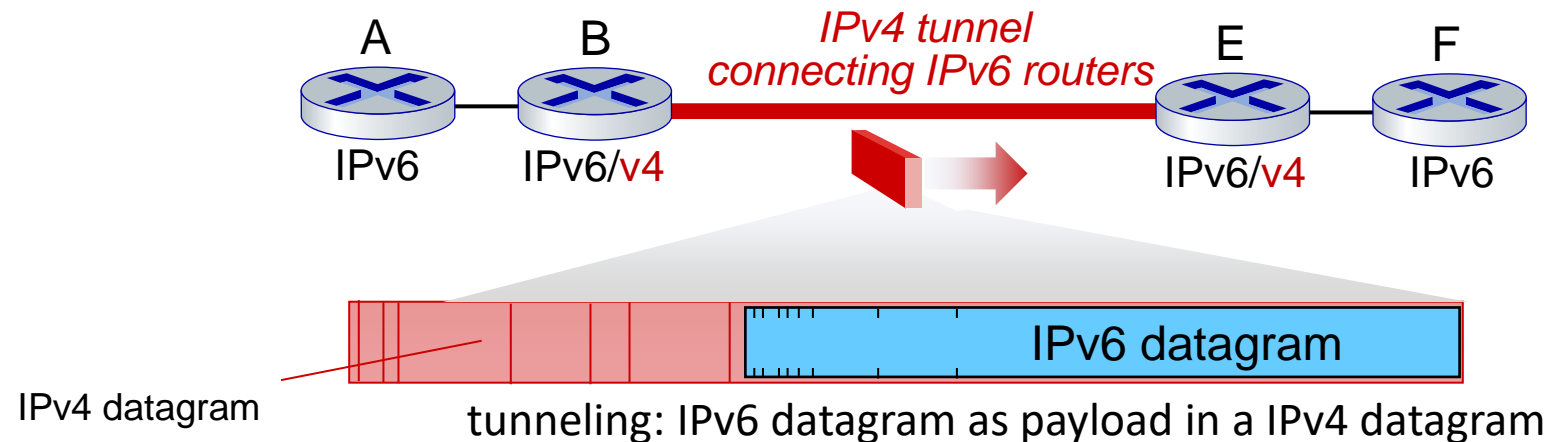
The usual: datagram as payload in link-layer frame
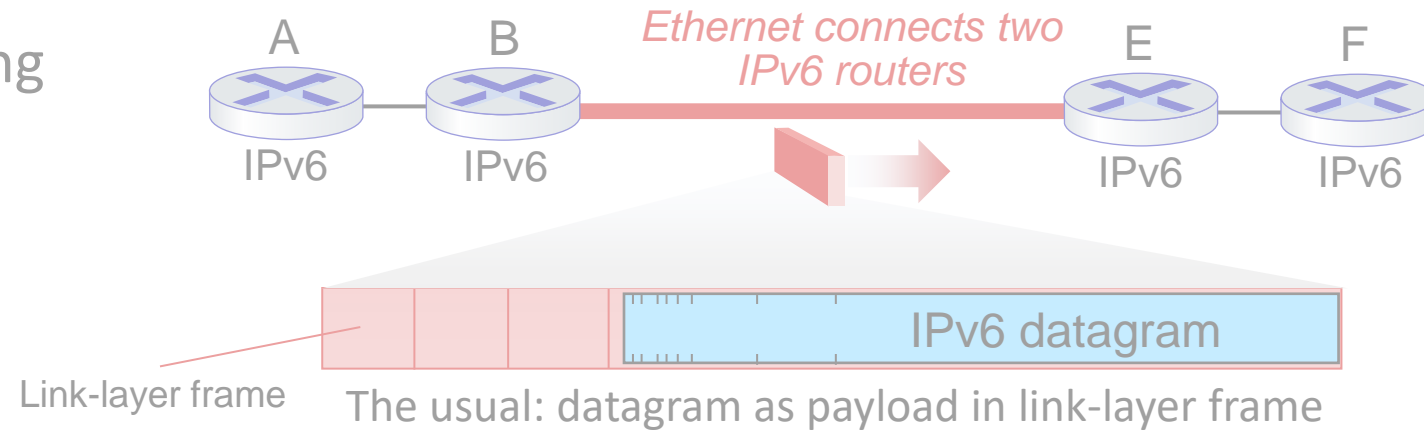
IPv4 network connecting two IPv6 routers

IPv4 network

Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

A — B (IPv6, IPv6)   *Ethernet connects two IPv6 routers*   E — F (IPv6, IPv6)

Link-layer frame | IPv6 datagram

The usual: datagram as payload in link-layer frame

IPv4 tunnel connecting two IPv6 routers

A — B (IPv6, IPv6/v4)   *IPv4 tunnel connecting IPv6 routers*   E — F (IPv6/v4, IPv6)

IPv4 datagram | IPv6 datagram

tunneling: IPv6 datagram as payload in a IPv4 datagram

Tunneling

logical view:

A   B   *IPv4 tunnel connecting IPv6 routers*   E   F

IPv6   IPv6/v4   IPv6/v4   IPv6

physical view:

A   B   C   D   E   F

IPv6   IPv6/v4   IPv4   IPv4   IPv6/v4   IPv6

flow: X
src: A
dest: F

data

src:B
dest: E

Flow: X
Src: A
Dest: F

data

src:B
dest: E

Flow: X
Src: A
Dest: F

data

src:B
dest: E

Flow: X
Src: A
Dest: F

data

flow: X
src: A
dest: F

data

Note source and destination addresses!

A-to-B:
IPv6

B-to-C:
IPv6 inside
IPv4

B-to-C:
IPv6 inside
IPv4

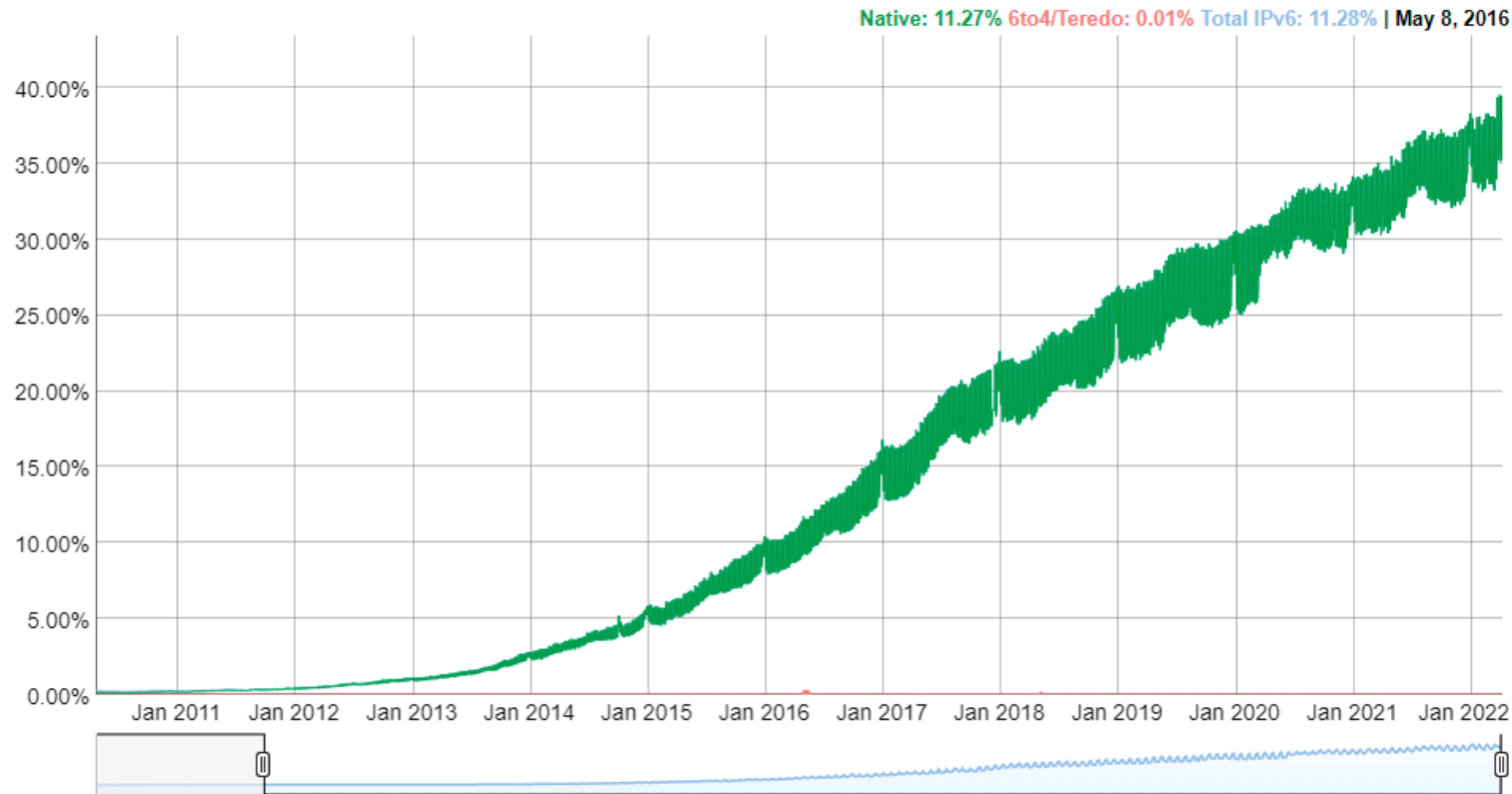B-to-C:
IPv6 inside
IPv4

E-to-F:
IPv6

IPv6: adoption

- Google[1]: ~ 40% of clients access services via IPv6

IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.

Native: 11.27% 6to4/Teredo: 0.01% Total IPv6: 11.28% | May 8, 2016

https://www.google.com/intl/en/ipv6/statistics.html

IPv6: adoption

- 中央部委、省级政府门户网站IPv6支持率达到81.42%
- 互联网网站对IPv6的支持率仅为33.7%

https://network.51cto.com/article/685981.html

IPv6: adoption

- Google[1]: ~ 30% of clients access services via IPv6

- Long (long!) time for deployment, use
  - 25 years and counting!
  - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, …
  - *Why?*

[1] https://www.google.com/intl/en/ipv6/statistics.html

Network layer: "data plane" roadmap

- Network layer: overview
  - data plane
  - control plane

- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling

- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

- **Generalized Forwarding, SDN**
  - Match+action
  - OpenFlow: match+action in action
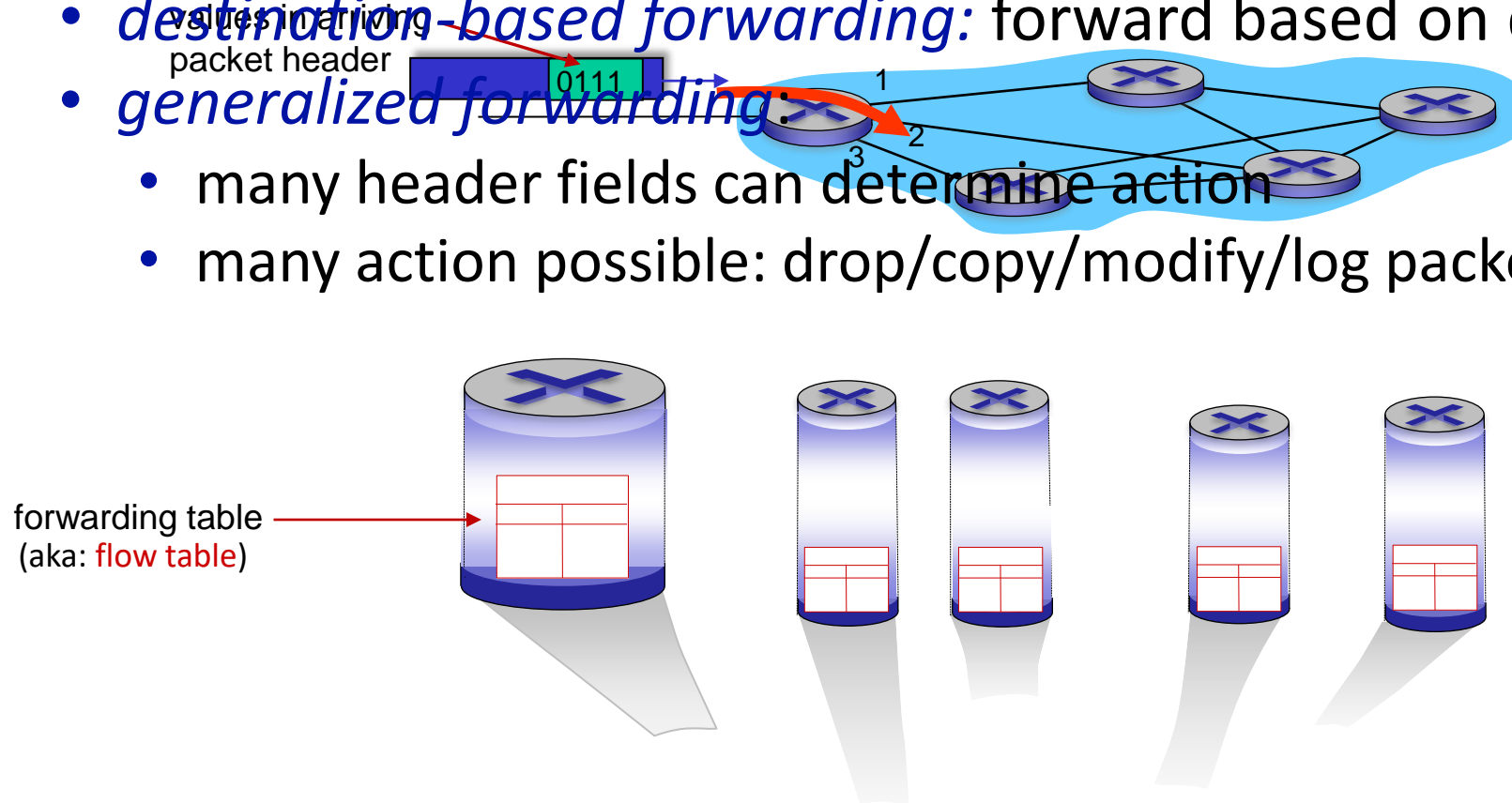
Generalized forwarding: match plus action

Review: each router contains a forwarding table (aka: flow table)

- "match plus action" abstraction: match bits in arriving packet, take action
 - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding:*
    - many header fields can determine action
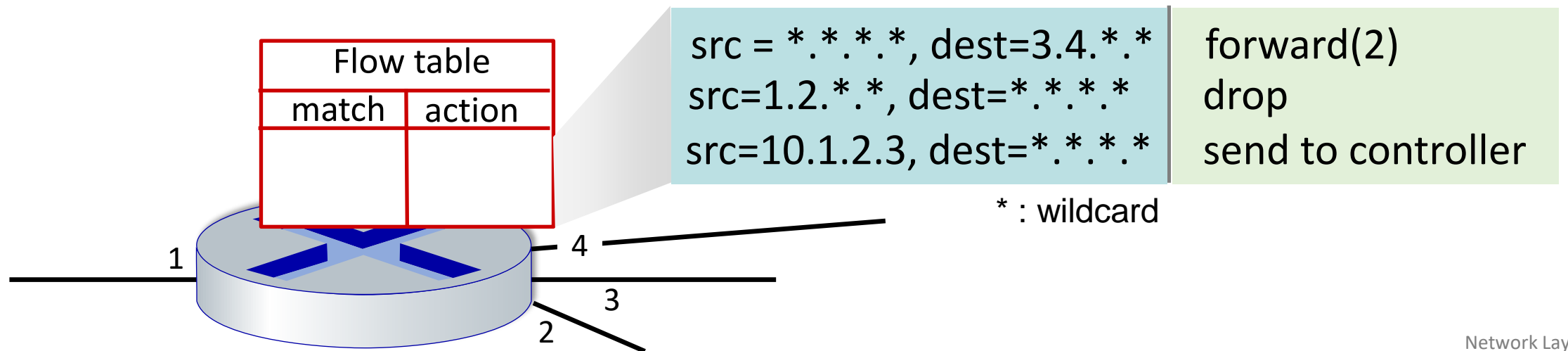   - many action possible: drop/copy/modify/log packet

values in arriving
packet header

`0111`

1
3 2

forwarding table
(aka: flow table)

Flow table abstraction

- **flow:** defined by header field values (in link-, network-, transport-layer fields)

- **generalized forwarding:** simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets

Flow table	
match	action

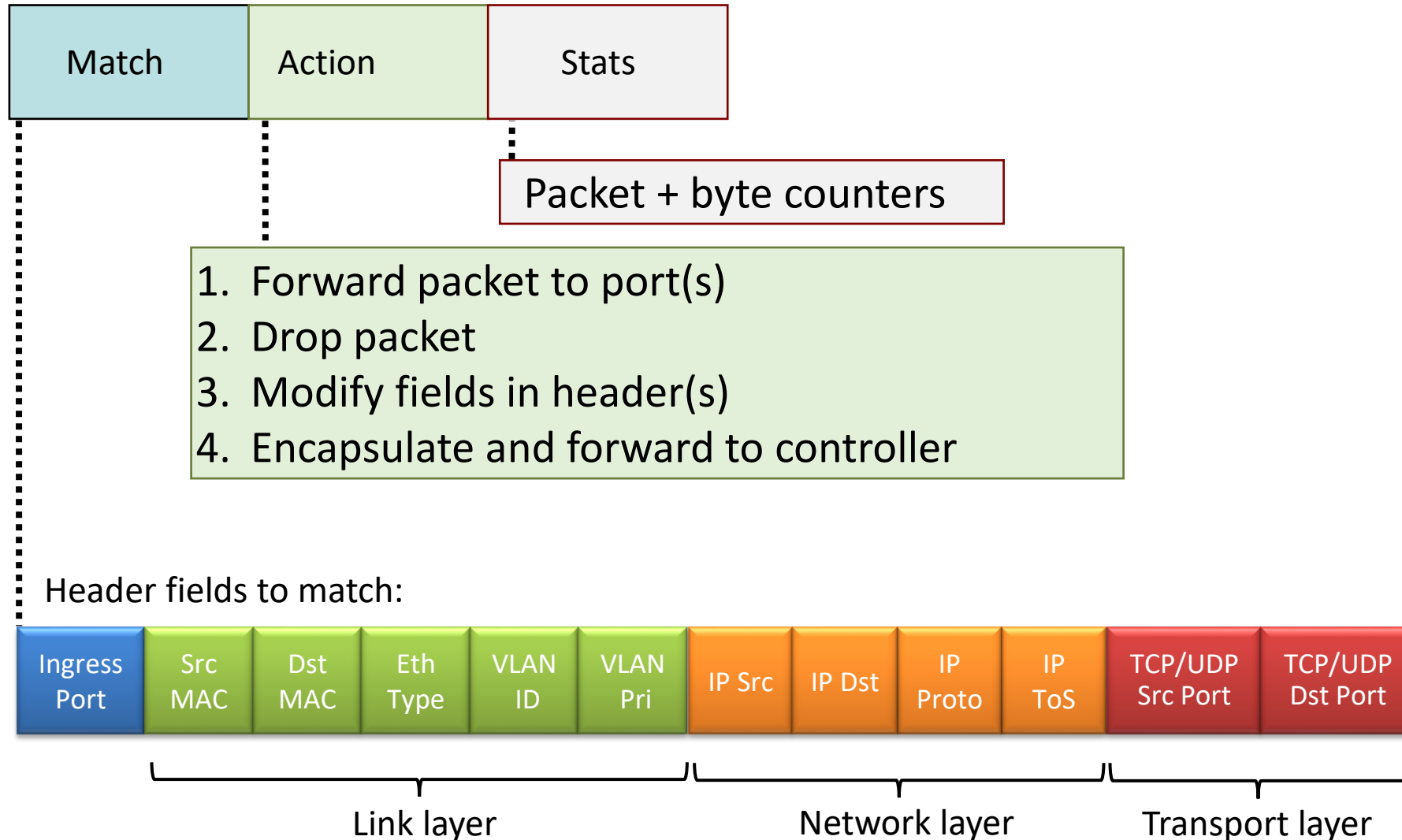Router's flow table define router's match+action rules
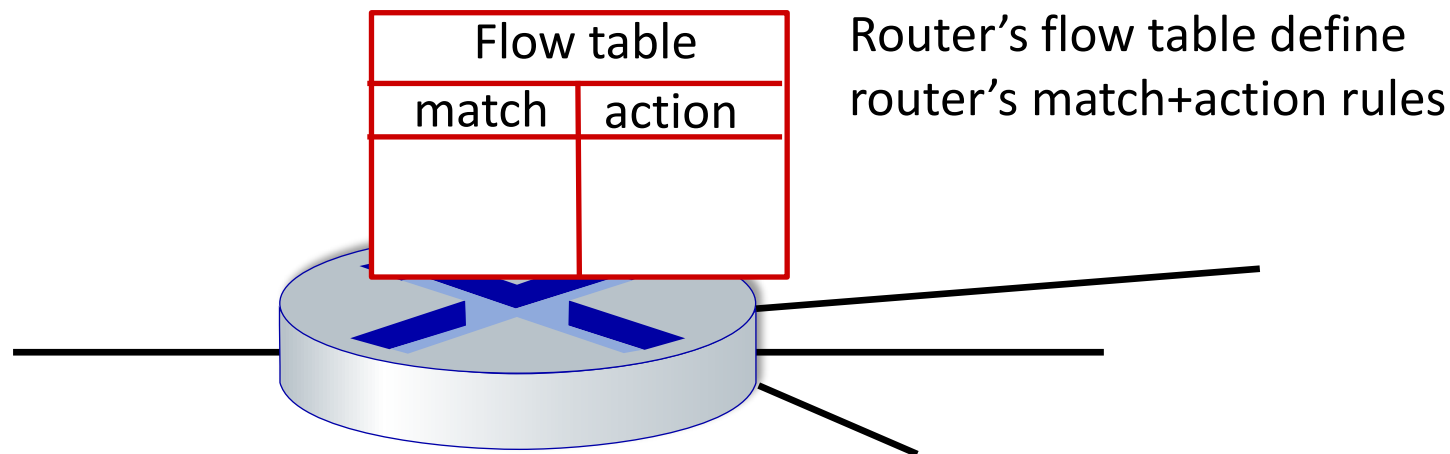
Flow table abstraction

- **flow:** defined by header fields

- **generalized forwarding: simple** packet-handling rules
  - **match:** pattern values in packet header fields
  - **actions:** for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **priority:** disambiguate overlapping patterns
  - **counters:** #bytes and #packets

Flow table	
match	action

src = *.*.*.*, dest=3.4.*.*	forward(2)
src=1.2.*.*, dest=*.*.*.*	drop
src=10.1.2.3, dest=*.*.*.*	send to controller

* : wildcard

1
4
3
2

OpenFlow: flow table entries

Match	Action	Stats

Packet + byte counters

1. Forward packet to port(s)
2. Drop packet
3. Modify fields in header(s)
4. Encapsulate and forward to controller

Header fields to match:

Ingress Port	Src MAC	Dst MAC	Eth Type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Proto	IP ToS	TCP/UDP Src Port	TCP/UDP Dst Port

Link layer · Network layer · Transport layer

OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices

Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

NAT
- *match:* IP address and port
- *action:* rewrite address and port

OpenFlow example
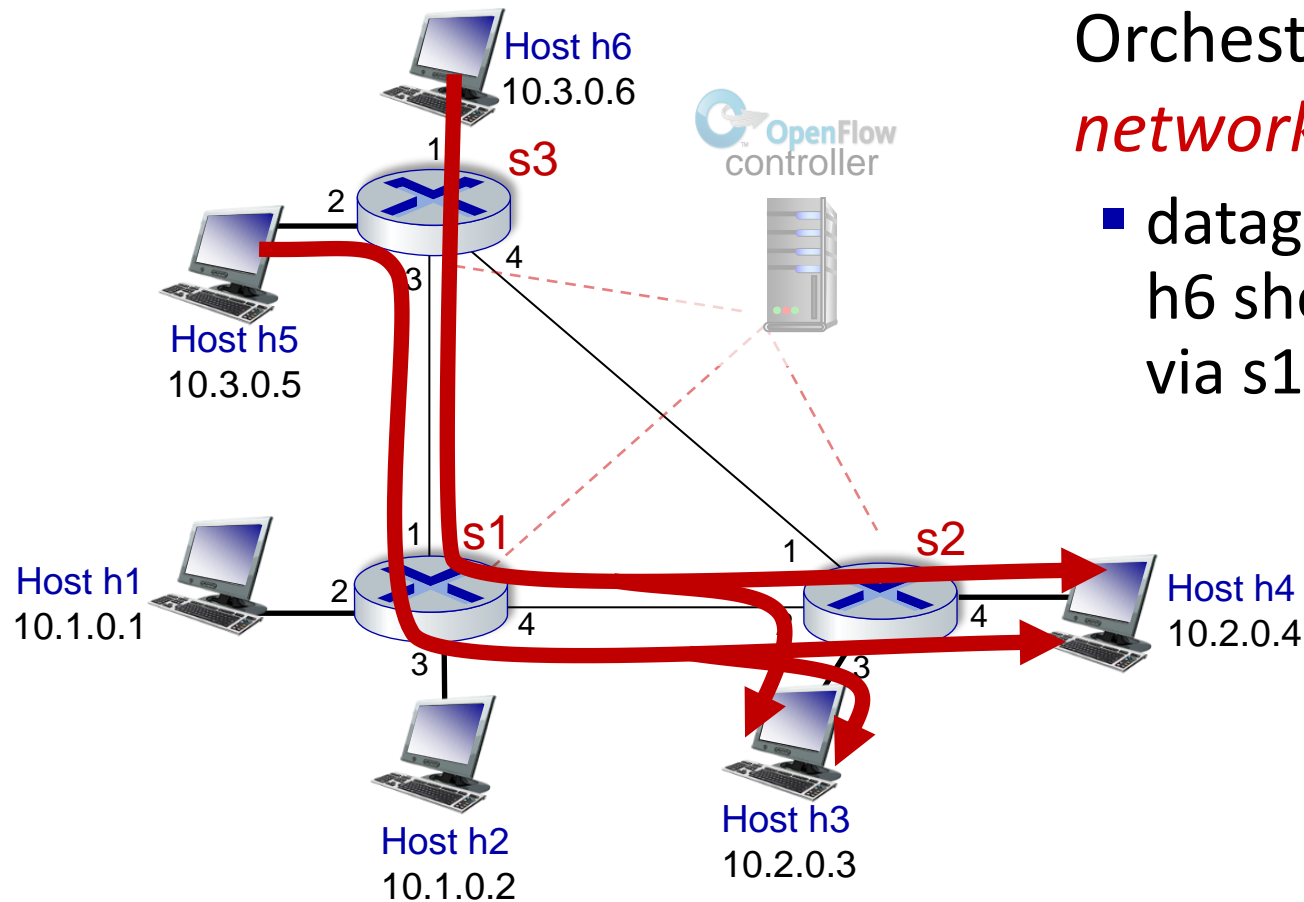
Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

OpenFlow example

match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

Host h6
10.3.0.6

s3

OpenFlow controller

Host h5
10.3.0.5

Host h1
10.1.0.1

s1

Host h2
10.1.0.2

s2

Host h4
10.2.0.4

Host h3
10.2.0.3
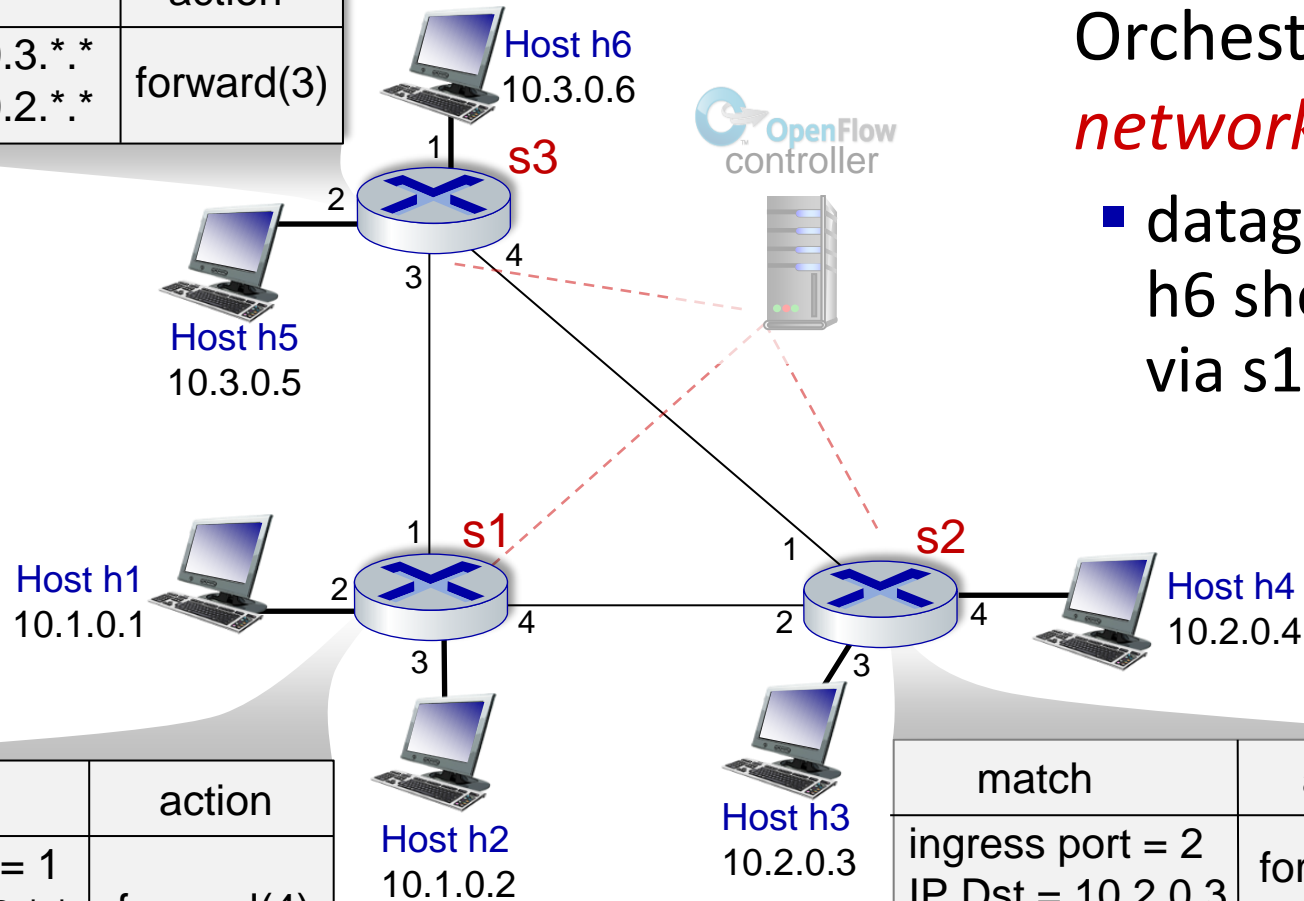
match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)

Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

Generalized forwarding: summary

- "match plus action" abstraction: match bits in arriving packet header(s) in any layers, take action
    - matching over many fields (link-, network-, transport-layer)
    - local actions: drop, forward, modify, or send matched packet to controller
    - "program" *network-wide* behaviors
- simple form of "network programmability"
    - programmable, per-packet "processing"
    - *historical roots:* active networking
    - *today:* more generalized programming: P4 (see p4.org).

Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN

Question: how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

Answer: by the control plane (next chapter)

Additional Chapter 4 slides

IP fragmentation/reassembly

- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs

- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at *destination*
  - IP header bits used to identify, order related fragments

fragmentation:
in: one large datagram
out: 3 smaller datagrams
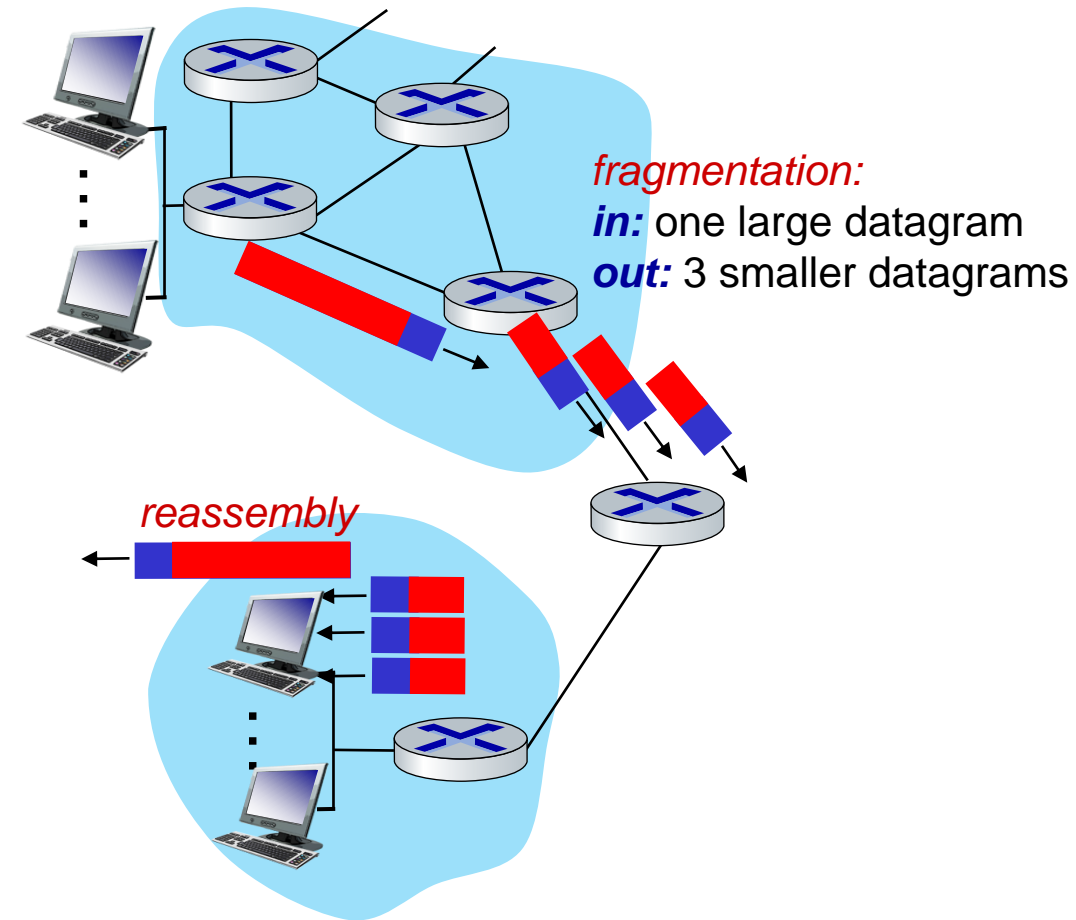
reassembly

IP fragmentation/reassembly

example:
- 4000 byte datagram
- MTU = 1500 bytes

	length =4000	ID =x	fragflag =0	offset =0	

one large datagram becomes several smaller datagrams

1480 bytes in data field

offset = 1480/8

	length =1500	ID =x	fragflag =1	offset =0	

	length =1500	ID =x	fragflag =1	offset =185	

	length =1040	ID =x	fragflag =0	offset =370	

Chapter 5
Network Layer:
Control Plane
Part 1

Instructor: Zhuozhao Li

Lab: Qing Wang

Department of Computer Science and Engineering

Network layer: "control plane" roadmap

- **introduction**
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Network-layer functions

- forwarding: move packets from router's input to appropriate router output

data plane

- routing: determine route taken by packets from source to destination

control plane

Two approaches to structuring network control plane:
- per-router control (traditional)
- logically centralized control (software defined networking)

Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane
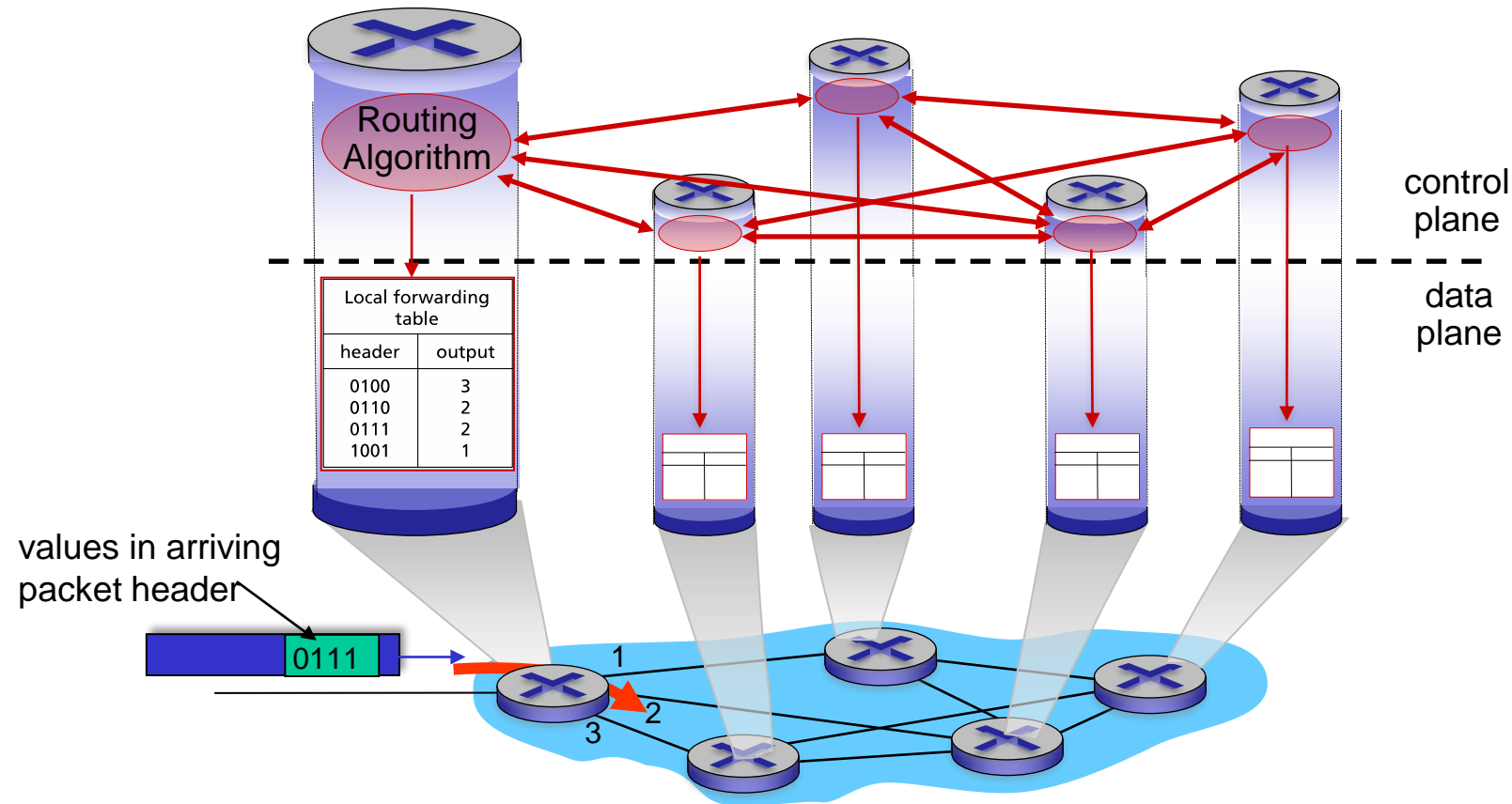
Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers
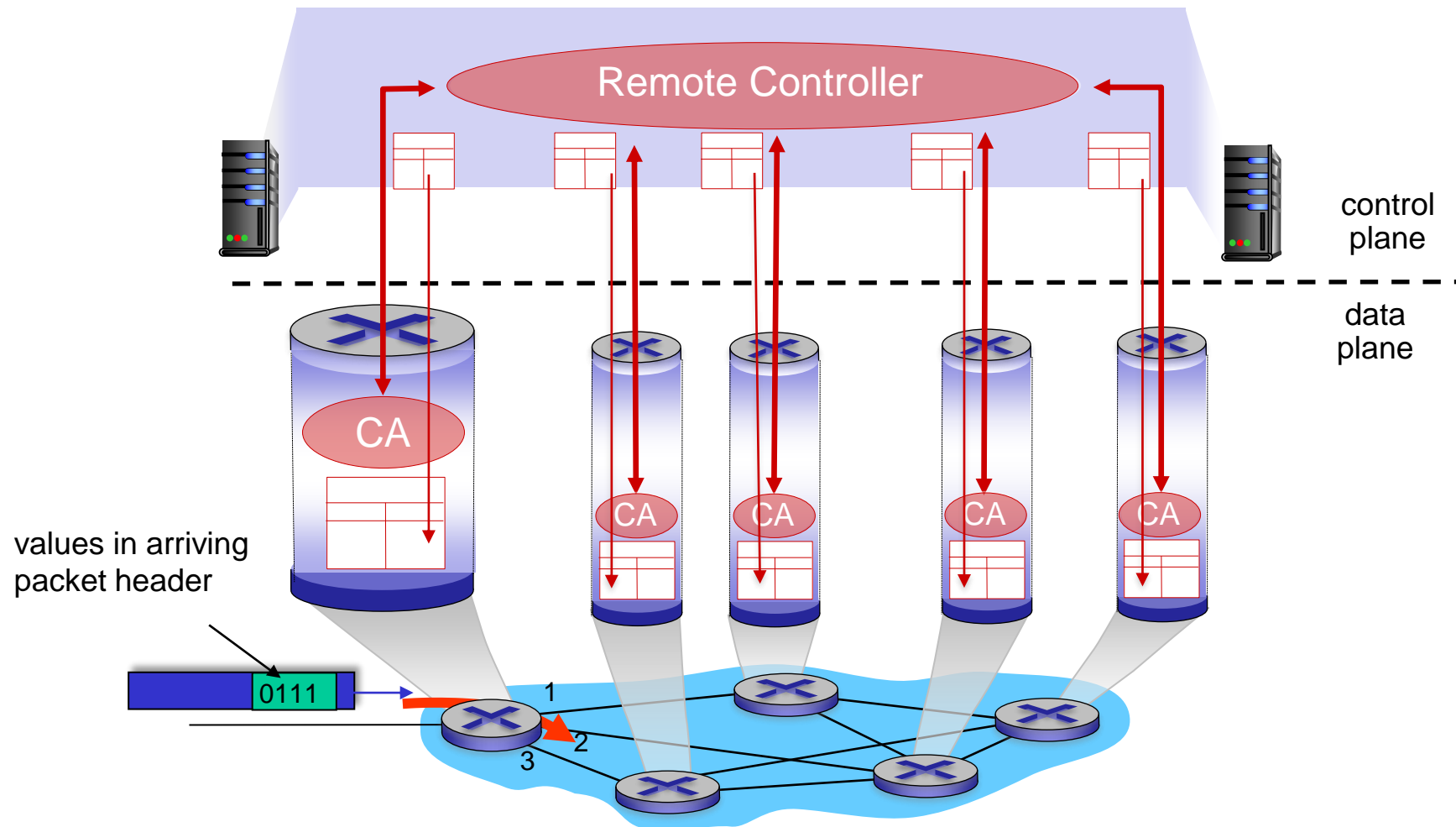
Network layer: "control plane" roadmap

- introduction

- **routing protocols**
  - **link state**
  - **distance vector**

- intra-ISP routing: OSPF

- routing among ISPs: BGP

- SDN control plane

- Internet Control Message Protocol

- network management, configuration
 - SNMP
  - NETCONF/YANG

Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host

- **"good":** least "cost", "fastest", "least congested"

- routing: a "top-10" networking challenge!
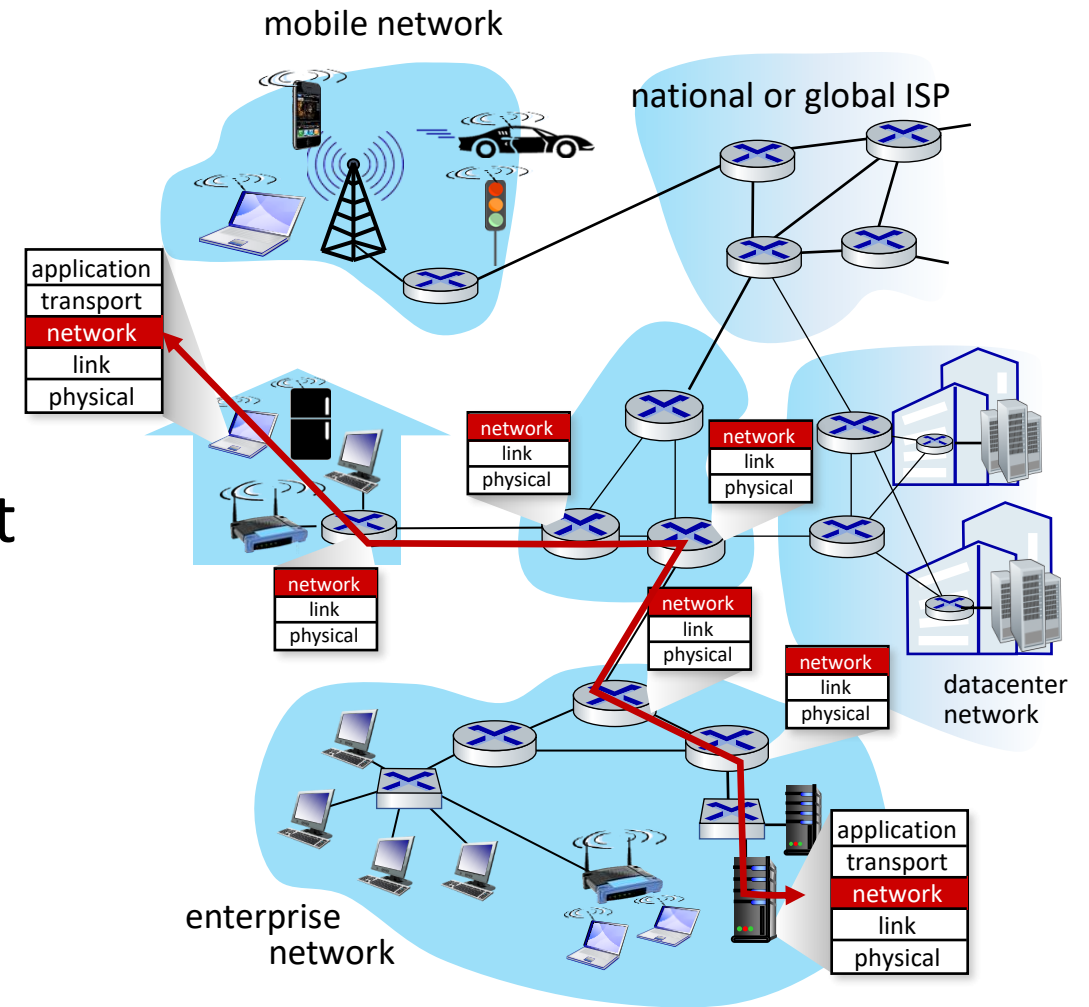
Graph abstraction: link costs
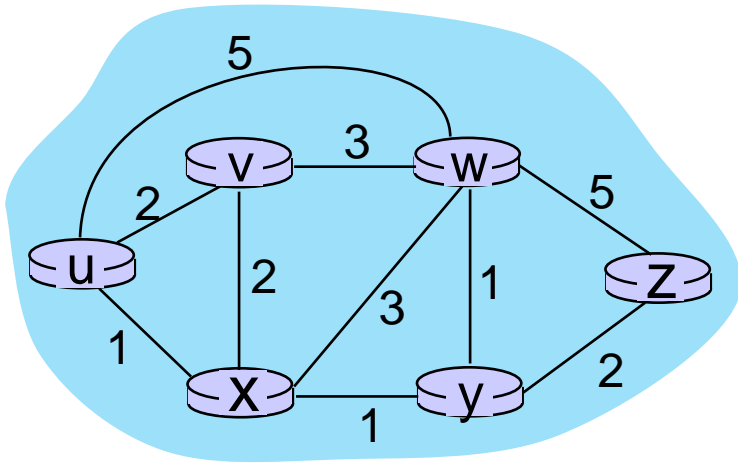
$c_{a,b}$: cost of *direct* link connecting a and b

e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: $G = (N,E)$

N: set of routers = { u, v, w, x, y, z }

E: set of links ={ $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$ }

Routing algorithm classification
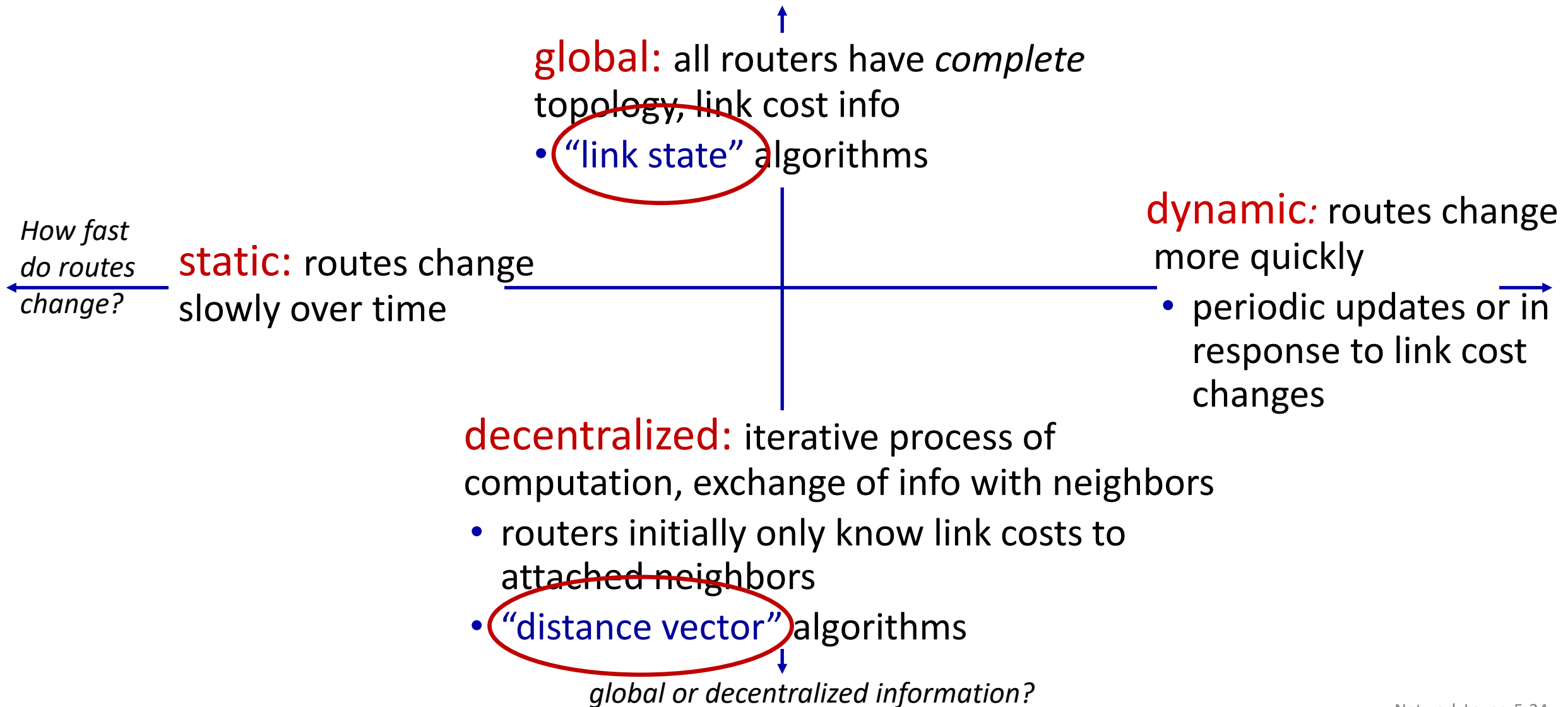
global: all routers have *complete* topology, link cost info
- "link state" algorithms

dynamic: routes change more quickly
- periodic updates or in response to link cost changes

static: routes change slowly over time

How fast do routes change?

decentralized: iterative process of computation, exchange of info with neighbors
- routers initially only know link costs to attached neighbors
- "distance vector" algorithms

global or decentralized information?

Network layer: "control plane" roadmap

- introduction

- **routing protocols**
  - **link state**
  - distance vector

- intra-ISP routing: OSPF

- routing among ISPs: BGP

- SDN control plane

- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Dijkstra's link-state routing algorithm

- **centralized:** network topology, link costs known to *all* nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- **computes least cost paths from one node ("source") to all other nodes**
  - gives *forwarding table* for that node
- **iterative:** after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: <u>direct</u> link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v): current* estimate of cost of least-cost-path from source to destination *v*
- *p(v):* predecessor node along path from source to *v*
- *N':* set of nodes whose least-cost-path *definitively* known

Dijkstra's algorithm: an example

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					
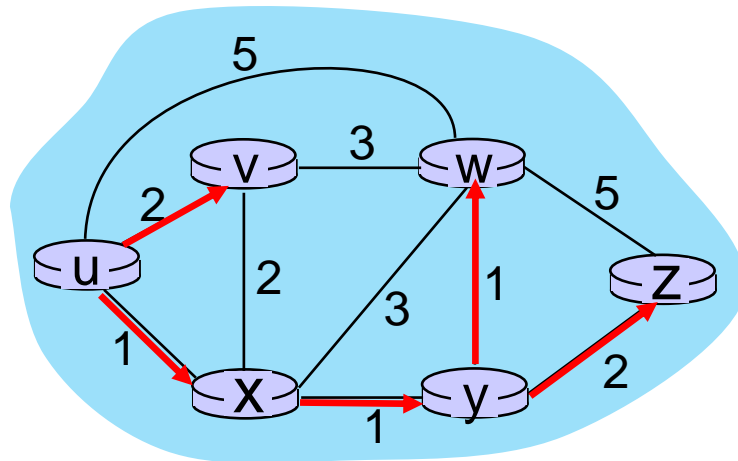
Initialization (step 0): For all a: if a adjacent to then D(a) = c_{u,a}

find a not in N' such that D(a) is a minimum
add a to N'
update D(b) for all b adjacent to a and not in N' :
D(b) = min (D(b), D(a) + c_{a,b})

Dijkstra's link-state routing algorithm

```
1  Initialization:
2     N' = {u}                        /* compute least cost path from u to all other nodes */
3     for all nodes v
4        if v adjacent to u           /* u initially knows direct-path-cost only to  direct neighbors   */
5           then D(v) = c_u,v          /* but may not be minimum cost!                          */
6        else D(v) = ∞
7

8  Loop
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12        D(v) = min ( D(v),  D(w) + c_w,v )
13    /* new least-path-cost to v is either old least-cost-path to v or known
14    least-cost-path to w plus direct-cost from w to v */
15 until all nodes in N'
```

Dijkstra's algorithm: an example
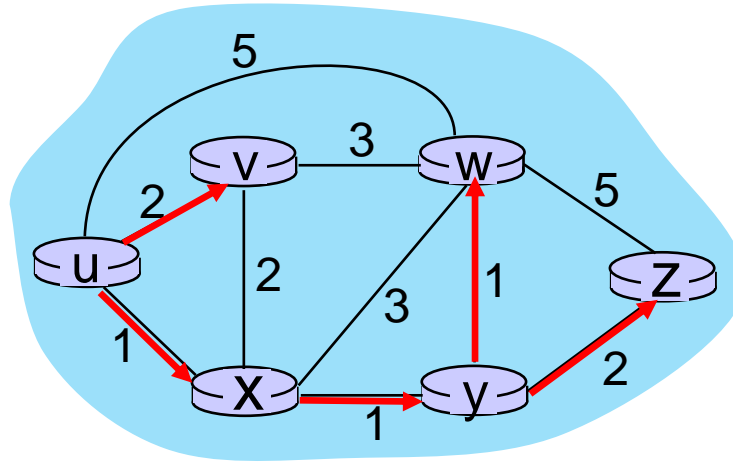
resulting least-cost-path tree from u:
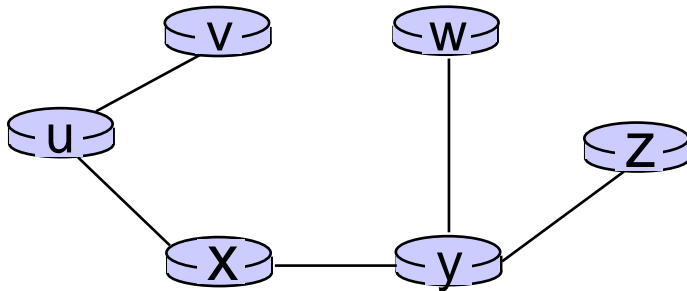
resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra's algorithm: discussion

algorithm complexity: *n* nodes

- each of *n* iteration: need to check all nodes, *w*, not in *N*

- *n*(*n*+1)/2 comparisons: O(*n*²) complexity

- more efficient implementations possible: O(*n*log*n*)

message complexity:

- each router must *broadcast* its link state information to other *n* routers

- efficient (and interesting!) broadcast algorithms: O(*n*) link crossings to disseminate a broadcast message from one source

- each router's message crosses O(*n*) links: overall message complexity: O(*n*²)

Dijkstra's algorithm: oscillations possible

- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, b with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent
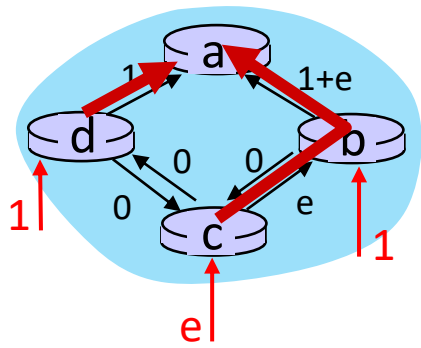
initially
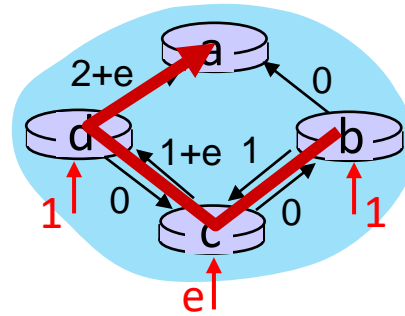
given these costs,
find new routing….
resulting in new costs
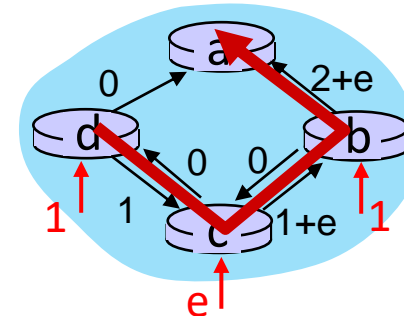
given these costs,
find new routing….
resulting in new costs
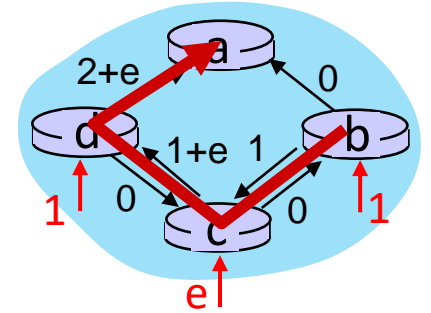
given these costs,
find new routing….
resulting in new costs

Network layer: "control plane" roadmap

- introduction
- **routing protocols**
  - link state
  - **distance vector**
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

v's estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

Suppose that *u*'s neighboring node*s, x,v,w,* know that for destination *z*:

$D_v(z) = 5$

$D_w(z) = 3$

$D_x(z) = 3$
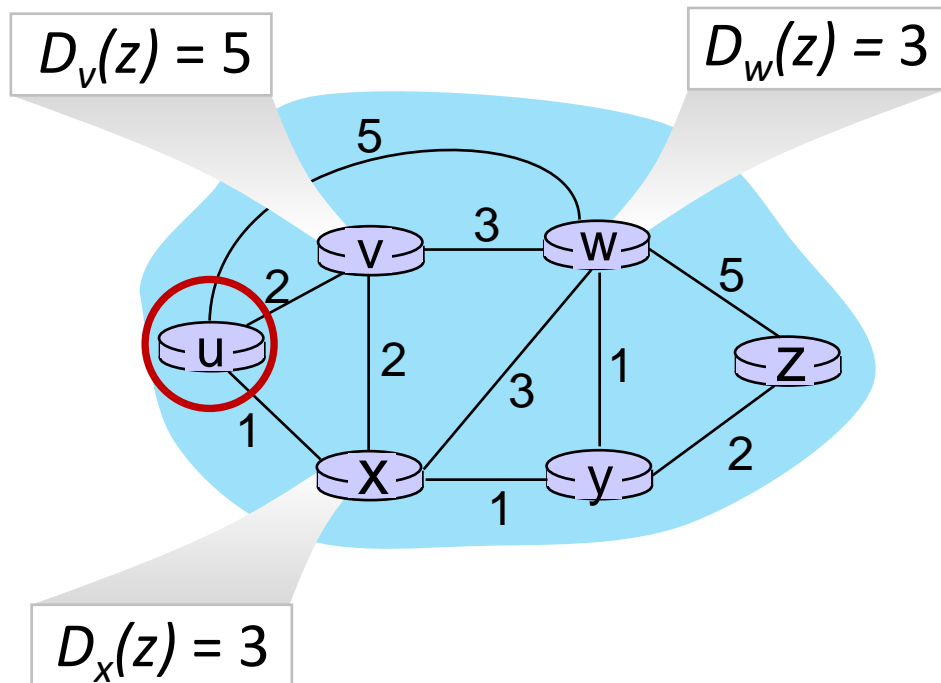
Bellman-Ford equation says:

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

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors

- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Distance vector algorithm:

each node:

wait for (change in local link cost or msg from neighbor)

↓

recompute DV estimates using DV received from neighbor

↓

if DV to any destination has changed, *notify* neighbors
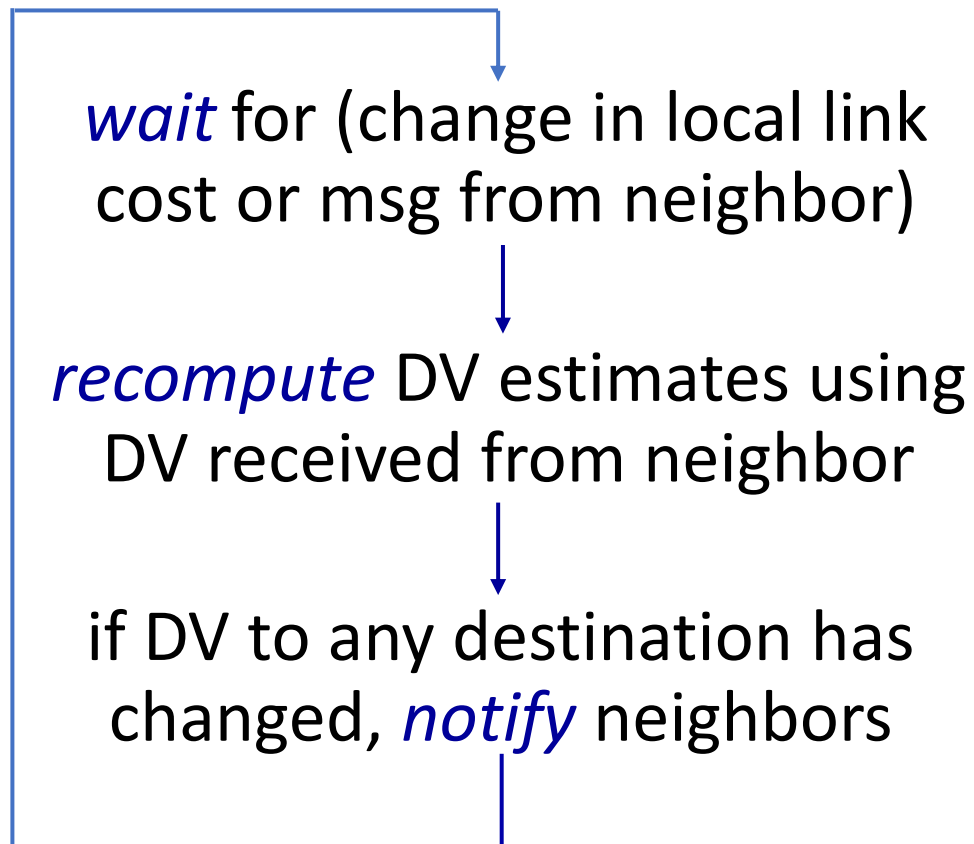
iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

Distance vector: example

t=0

DV in a:
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

a — 8 — b — 1 — c

1 (a–d), 1 (b–e)

d — 1 — e — 1 — f

1 (d–g), 1 (e–h), 1 (f–i)

g — 1 — h — 1 — i
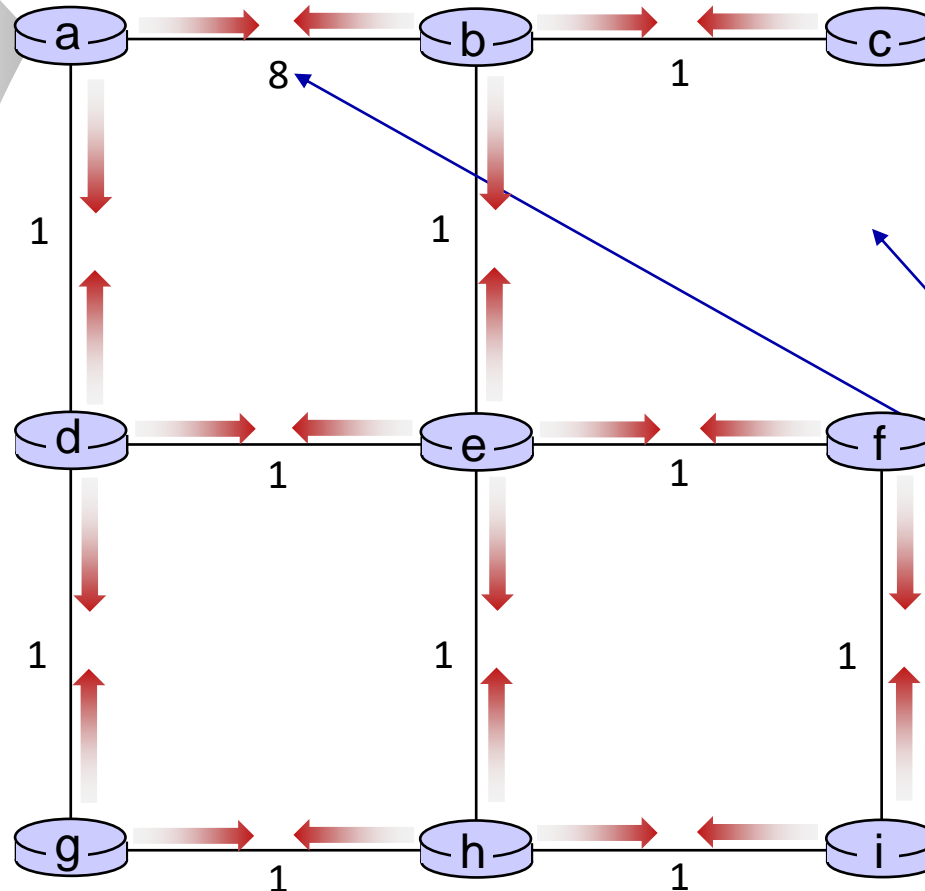
A few asymmetries:
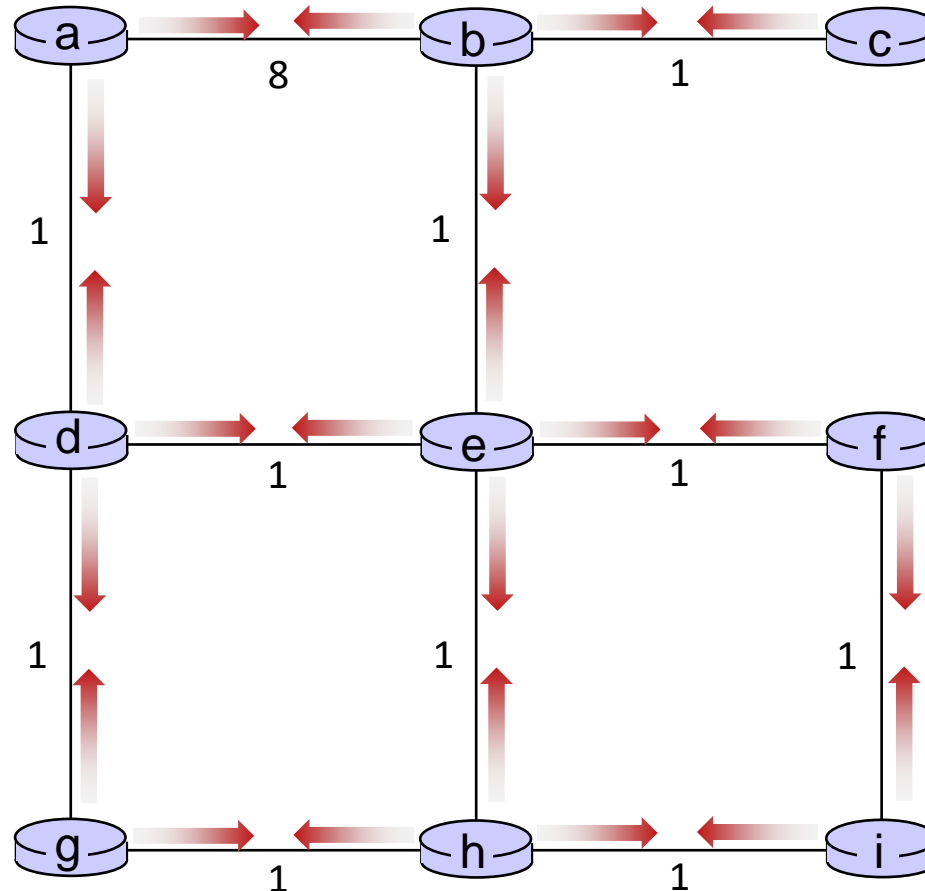- missing link
- larger cost

Distance vector example: iteration

t=1

All nodes:
- **receive distance vectors from neighbors**
- compute their new local distance vector
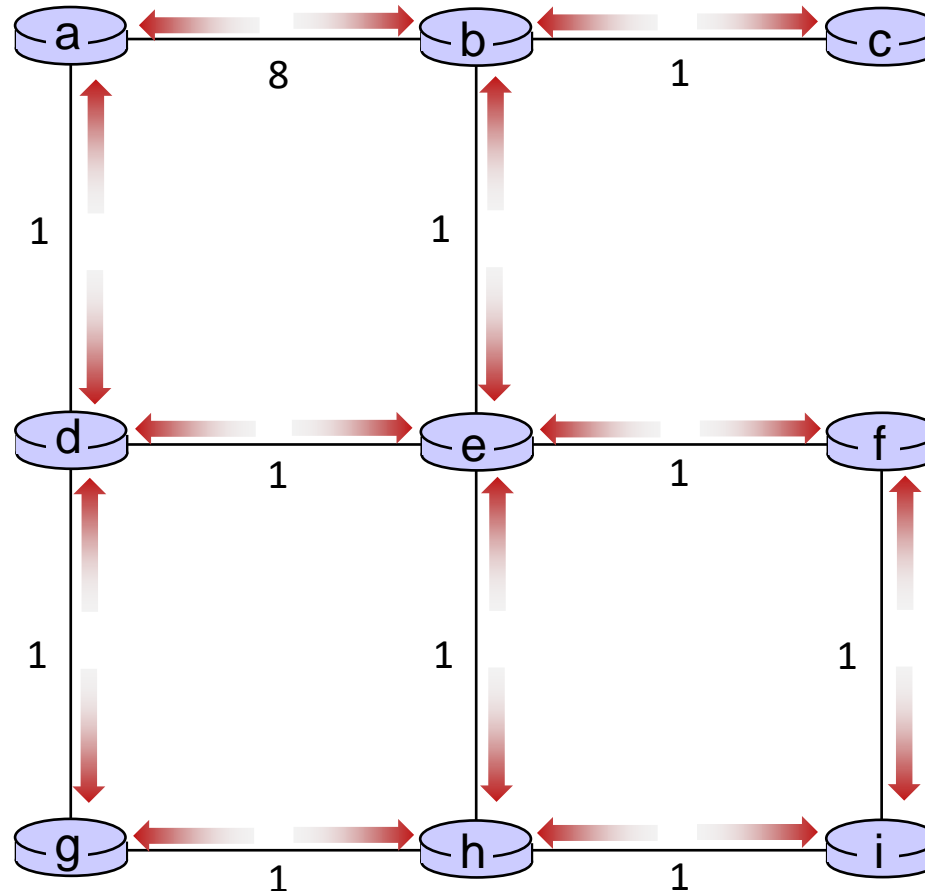- send their new local distance vector to neighbors

Distance vector example: iteration

t=1

All nodes:
- receive distance vectors from neighbors
- **compute their new local distance vector**
- send their new local distance vector to neighbors

Distance vector example: iteration

t=1

All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
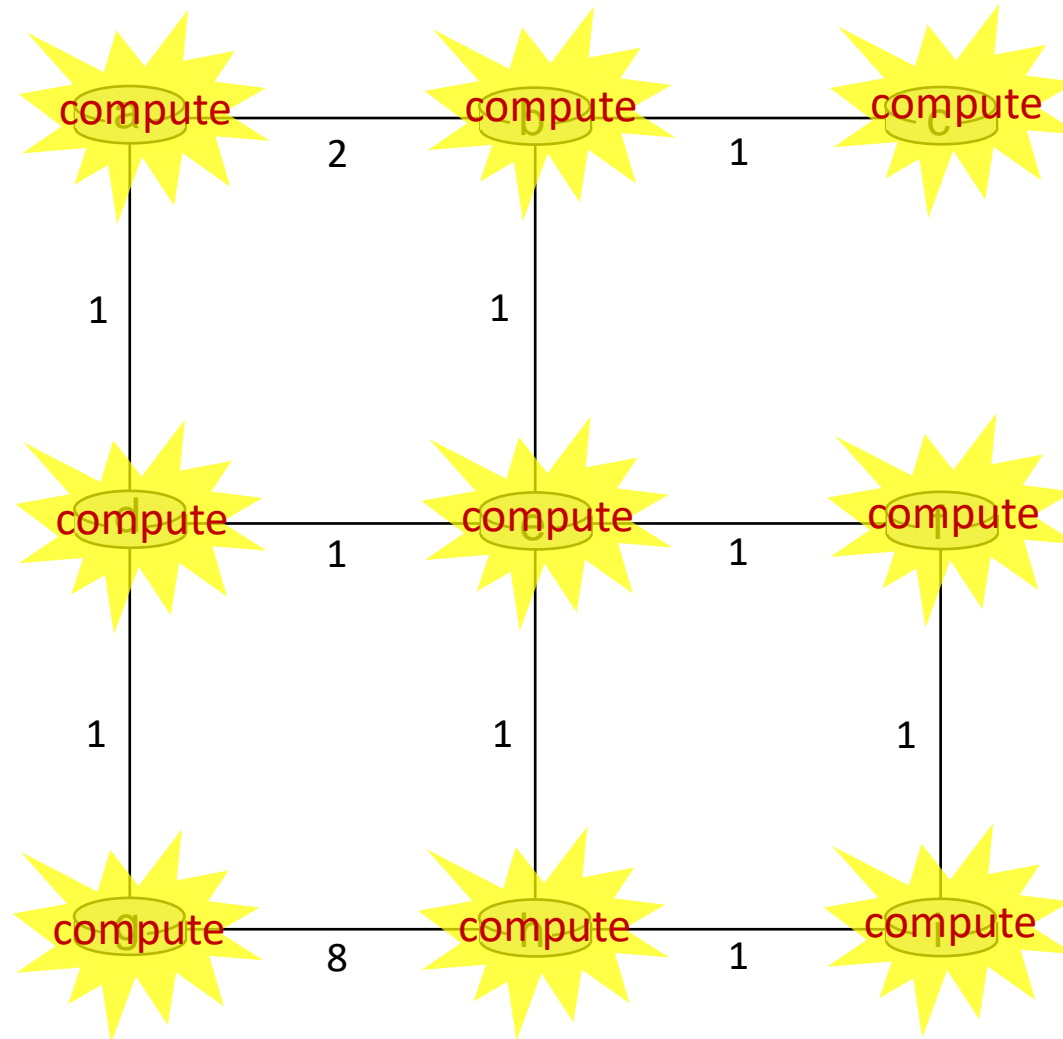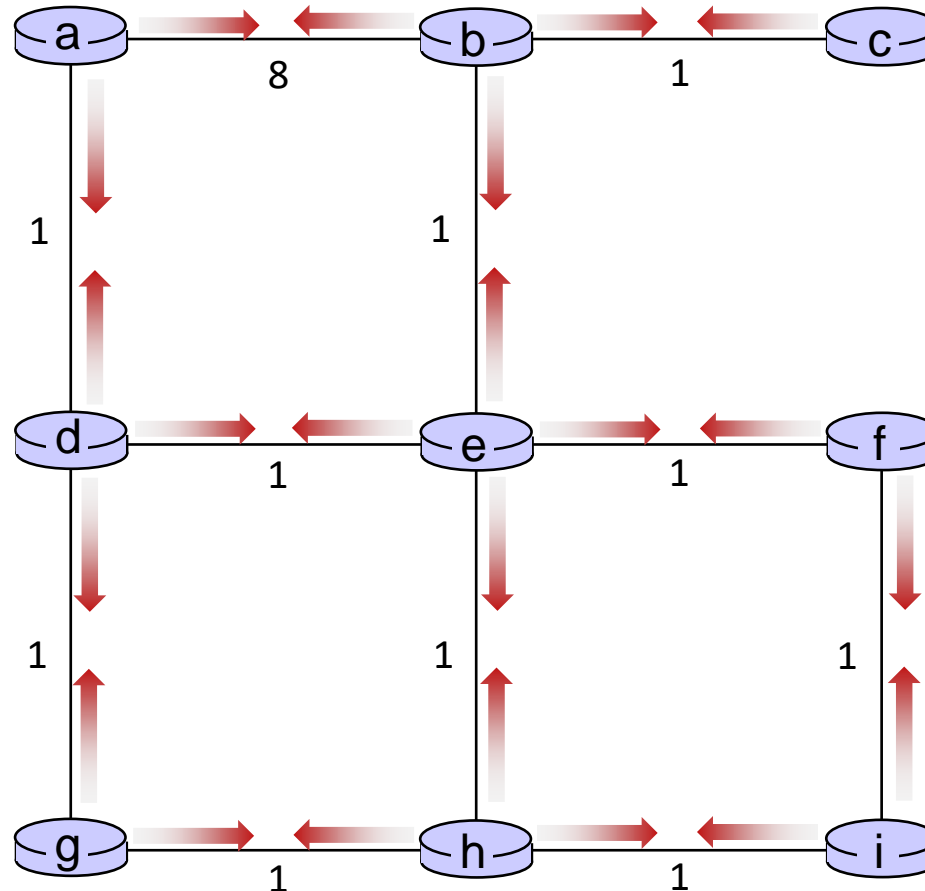- **send their new local distance vector to neighbors**

Distance vector example: iteration

t=2

All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

Distance vector example: iteration

t=2

All nodes:
- receive distance vectors from neighbors
- **compute their new local distance vector**
- send their new local distance vector to neighbors

Distance vector example: iteration

t=2

All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- **send their new local distance vector to neighbors**

Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

Distance vector example:

t=1

- b receives DVs from a, c, e

DV in a:

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$D_b(a) = 8$ $D_b(f) = \infty$
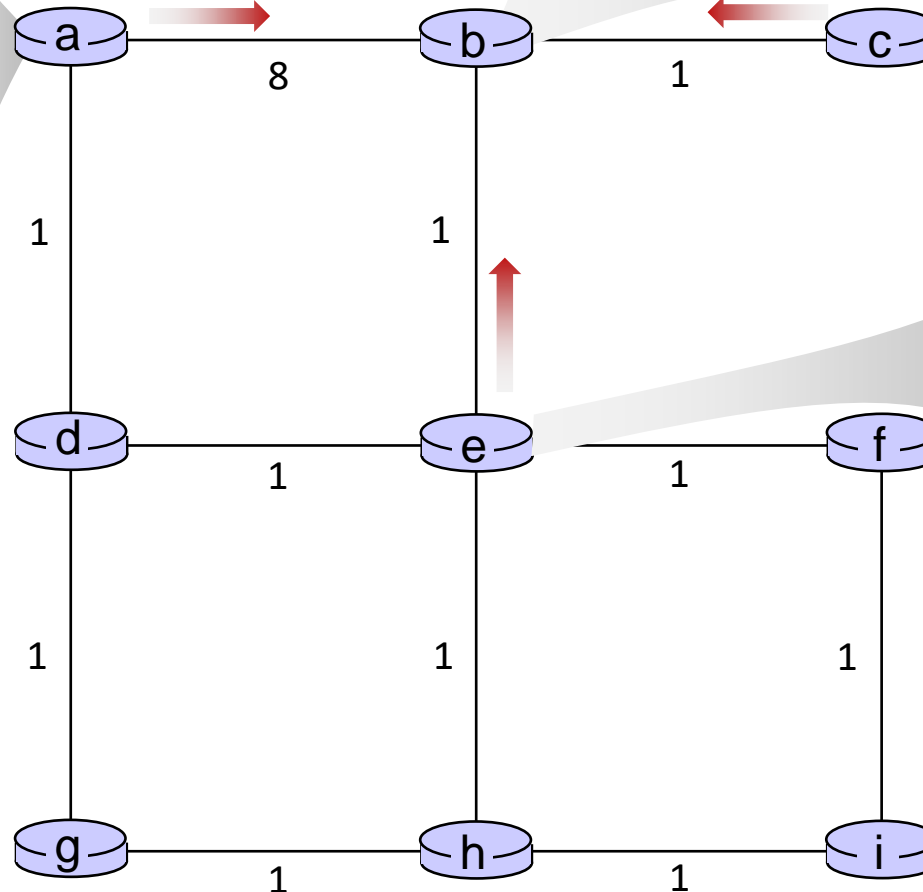$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
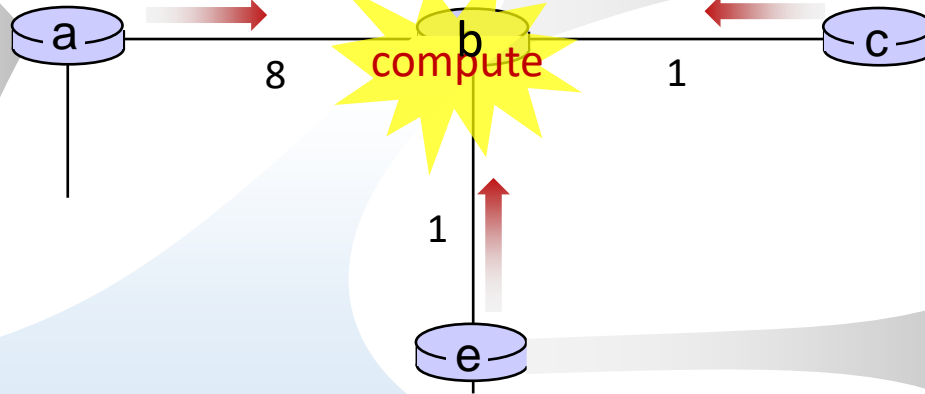$D_e(h) = 1$
$D_e(i) = \infty$

Distance vector example:

DV in b:

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in a:

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

t=1

- b receives DVs from a, c, e, computes:

$D_b(a) = \min\{c_{b,a}+D_a(a),\ c_{b,c}+D_c(a),\ c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c),\ c_{b,c}+D_c(c),\ c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d),\ c_{b,c}+D_c(d),\ c_{b,e}+D_e(d)\} = \min\{9,2,\infty\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e),\ c_{b,c}+D_c(e),\ c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f),\ c_{b,c}+D_c(f),\ c_{b,e}+D_e(f)\} = \min\{\infty,\infty,2\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g),\ c_{b,c}+D_c(g),\ c_{b,e}+D_e(g)\} = \min\{\infty,\infty,\infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h),\ c_{b,c}+D_c(h),\ c_{b,e}+D_e(h)\} = \min\{\infty,\infty,2\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i),\ c_{b,c}+D_c(i),\ c_{b,e}+D_e(i)\} = \min\{\infty,\infty,\infty\} = \infty$

a —8— b compute —1— c

1

e

DV in b:

$D_b(a) = 8$ $D_b(f) = 2$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = 2$ $D_b(h) = 2$
$D_b(e) = 1$ $D_b(i) = \infty$

Distance vector example: c

t=1

- c receives DVs from b

DV in a:

$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
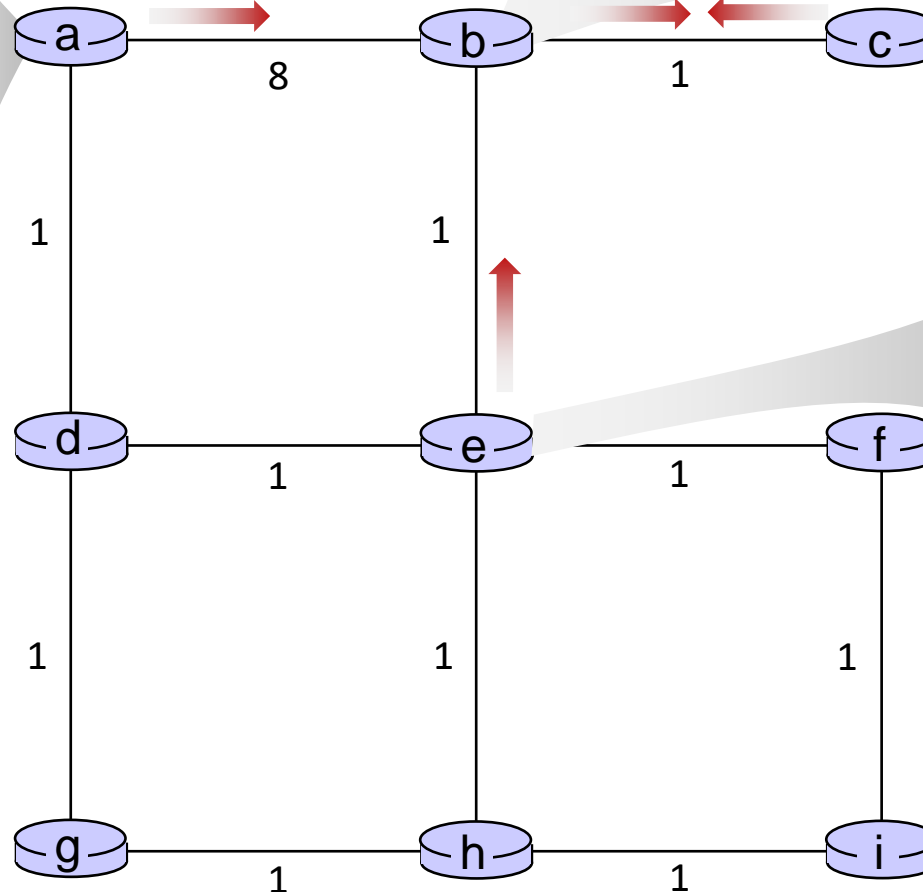$D_b(e) = 1$ $D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
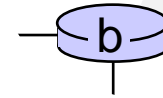$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

Distance vector example:

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
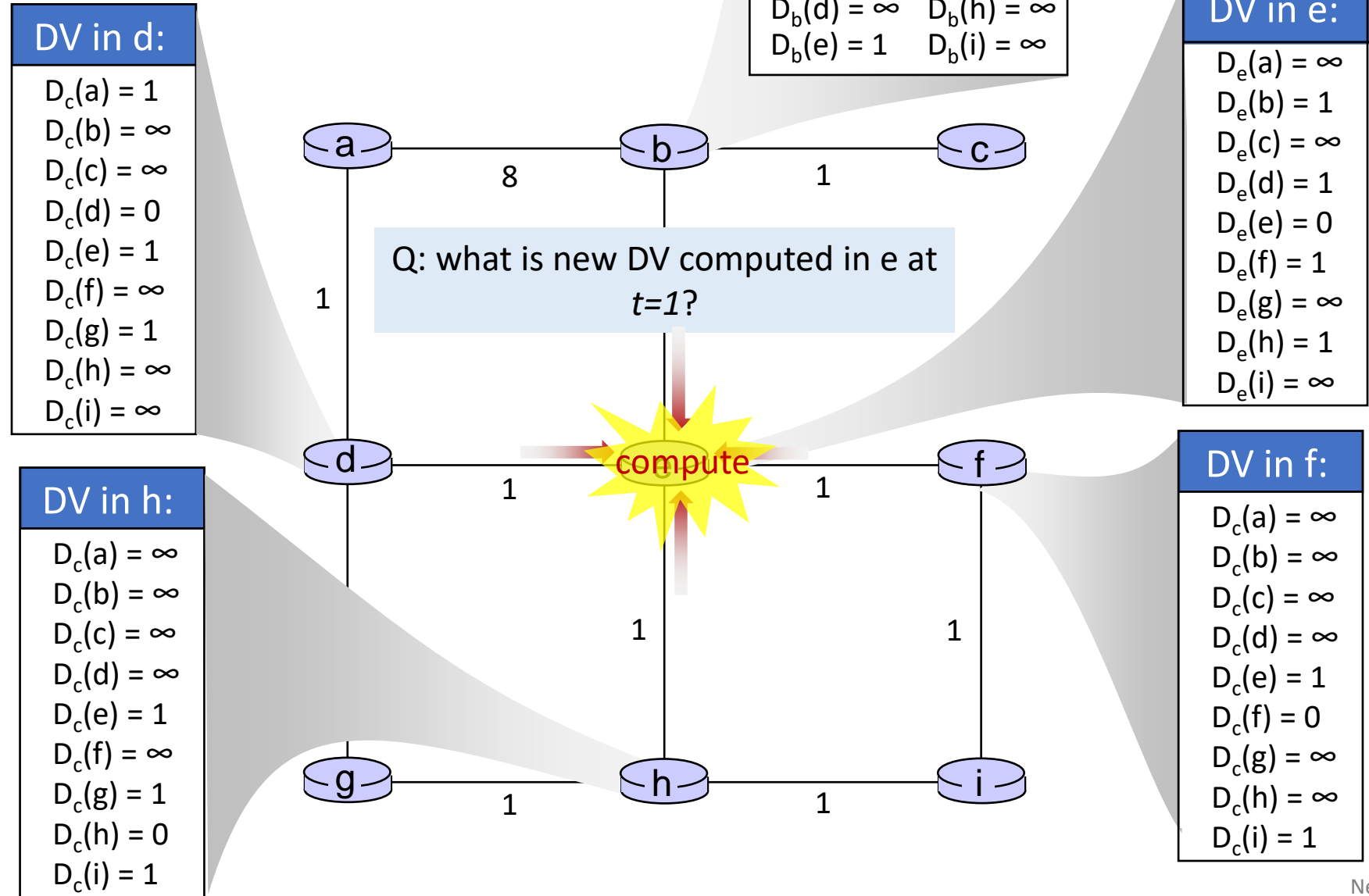$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

b $\longrightarrow$ 1 compute

t=1

- c receives DVs from b computes:

$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$

$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$

$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$

$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$

$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$

$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty = \infty$

$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

DV in c:

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

Distance vector example: c

DV in b:

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

DV in d:

$D_c(a) = 1$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = 0$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in h:

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = 0$
$D_c(i) = 1$

DV in f:

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = 0$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = 1$

t=1

- e receives DVs from b, d, f, h

Q: what is new DV computed in e at *t=1*?

compute

Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

t=0   c's state at t=0 is at c only

t=1   c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b

t=2   c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well

t=3   c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well

t=4   c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well
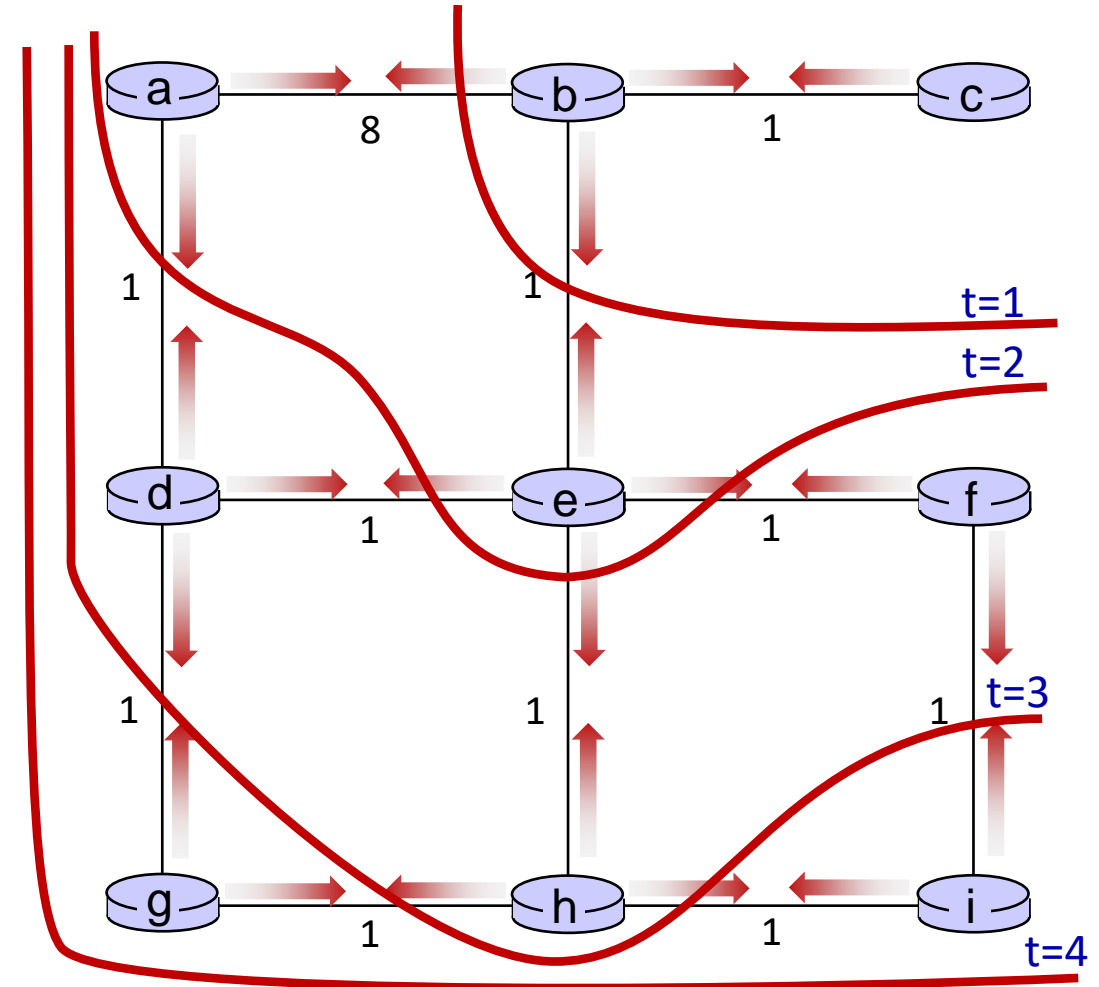
Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates local DV
- if DV changes, notify neighbors
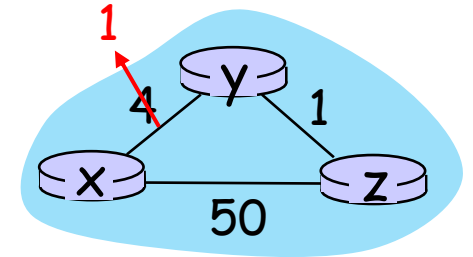
t_0 : y detects link-cost change, updates its DV, informs its neighbors.

"good news travels fast"

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.

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- "bad news travels slow" – count-to-infinity problem:

  - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x*.

  - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x*.

  - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x*.

  - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x*.

    ...

- see text for solutions. *Distributed algorithms are tricky!*
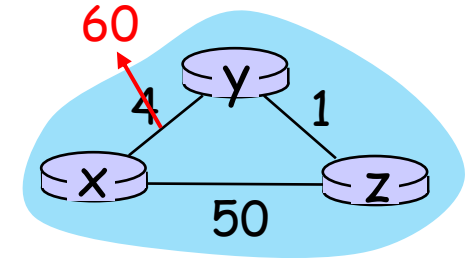
Comparison of LS and DV algorithms

message complexity

   LS: n routers, O(n^2) messages sent

   DV: exchange between neighbors; convergence time varies

speed of convergence

   LS: O(n^2) algorithm, O(n^2) messages
- may have oscillations

   DV: convergence time varies
- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:
- router can advertise incorrect *link* cost
- each router computes only its *own* table

DV:
- DV router can advertise incorrect *path* cost ("I have a *really* low cost path to everywhere"): black-holing
- each router's table used by others: error propagate thru network

Network layer: "control plane" roadmap

- introduction
- routing protocols
- **intra-ISP routing: OSPF**
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

- network management, configuration
  - SNMP
  - NETCONF/YANG

Making routing scalable

our routing study thus far - idealized
- all routers identical
- network "flat"

… not true in practice

scale: billions of destinations:
- can't store all destinations in routing tables!
- routing table exchange would swamp links!

administrative autonomy:
- Internet: a network of networks
- each network admin may want to control routing in its own network

Internet approach to scalable routing

aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

intra-AS (aka "intra-domain"):
routing among *within same AS ("network")*

- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS (aka "inter-domain"):
routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

Interconnected ASes
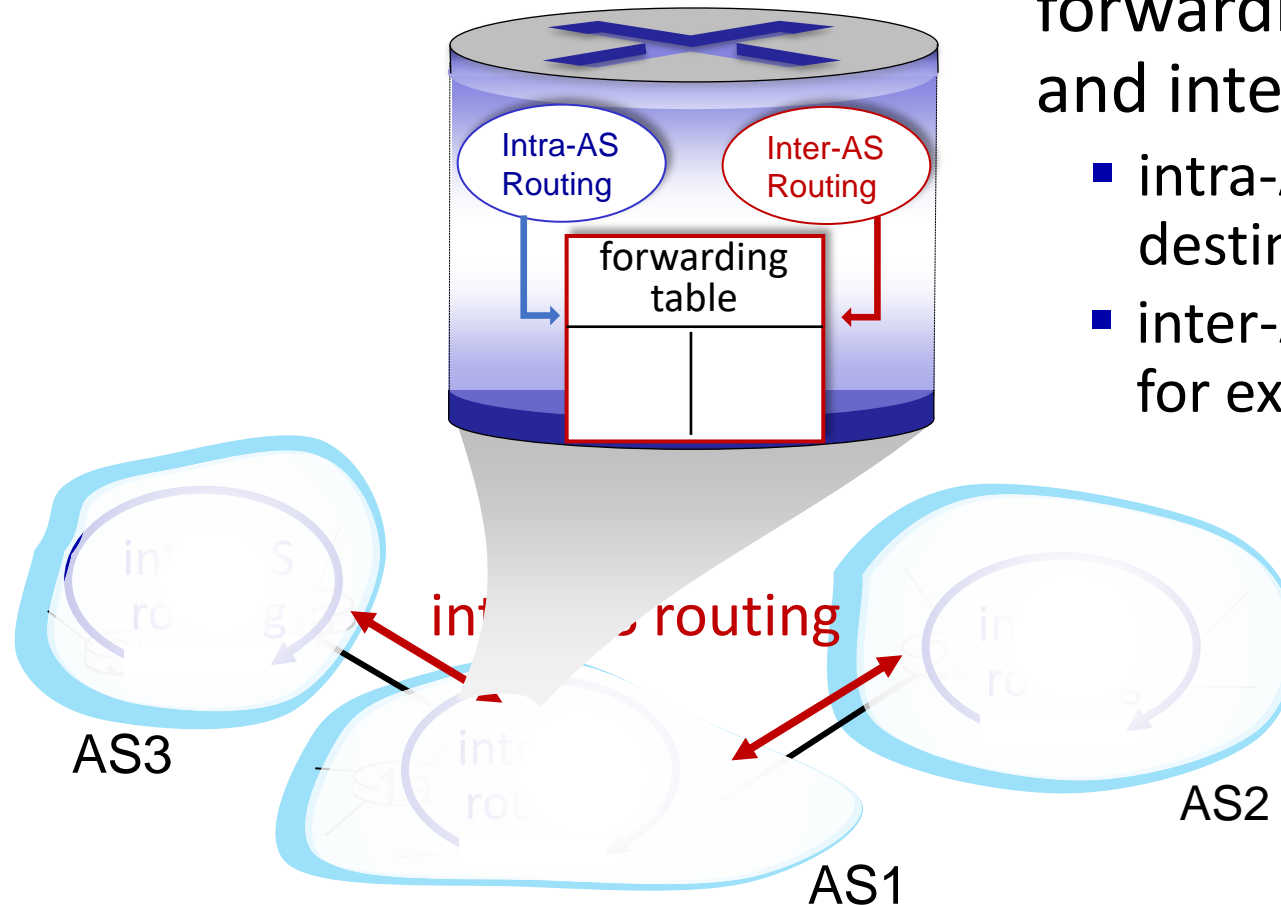
forwarding table configured by intra- and inter-AS routing algorithms

- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

Inter-AS routing: a role in intradomain forwarding

- suppose router in AS1 receives datagram destined outside of AS1:

?
- router should forward packet to gateway router in AS1, but which one?

AS1 inter-domain routing must:
1. learn which destinations reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1
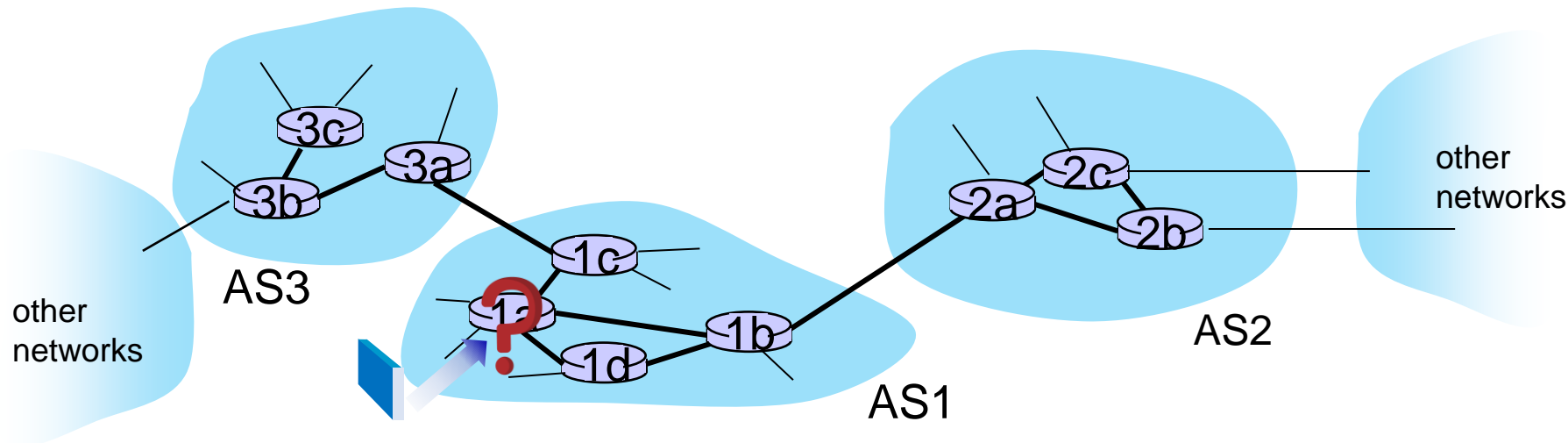
Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

- **RIP: Routing Information Protocol** [RFC 1723]
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used

- **EIGRP: Enhanced Interior Gateway Routing Protocol**
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])

- **OSPF: Open Shortest Path First** [RFC 2328]
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

OSPF (Open Shortest Path First) routing

- "open": publicly available

- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra's algorithm to compute forwarding table

- *security:* all OSPF messages authenticated (to prevent malicious intrusion)

Hierarchical OSPF

- **two-level hierarchy:** local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations

area border routers:
"summarize" distances to destinations in own area, advertise in backbone

local routers:
- flood LS in area only
- compute routing within area
- forward packets to outside via area border router

boundary router:
connects to other ASes

backbone router:
runs OSPF limited to backbone

backbone

internal routers

area 1

area 2

area 3