Computer Networks

Chapter 5 – The Network Layer: Control Plane

Edition8-1

Topics

- Understand principles behind network control plane:
  - Traditional routing algorithms
  - SDN controllers
  - Network management, configuration
- In Internet:
  - Traditional routing algorithms: OSPF, BGP, …
  - SDN controllers: OpenFlow, ODL and ONOS controllers
  - Network management, configuration
    - Internet Control Message Protocol: ICMP
    - SNMP, YANG/NETCONF

Contents

5.1 Introduction

- How forwarding and flow tables are computed, maintained and installed

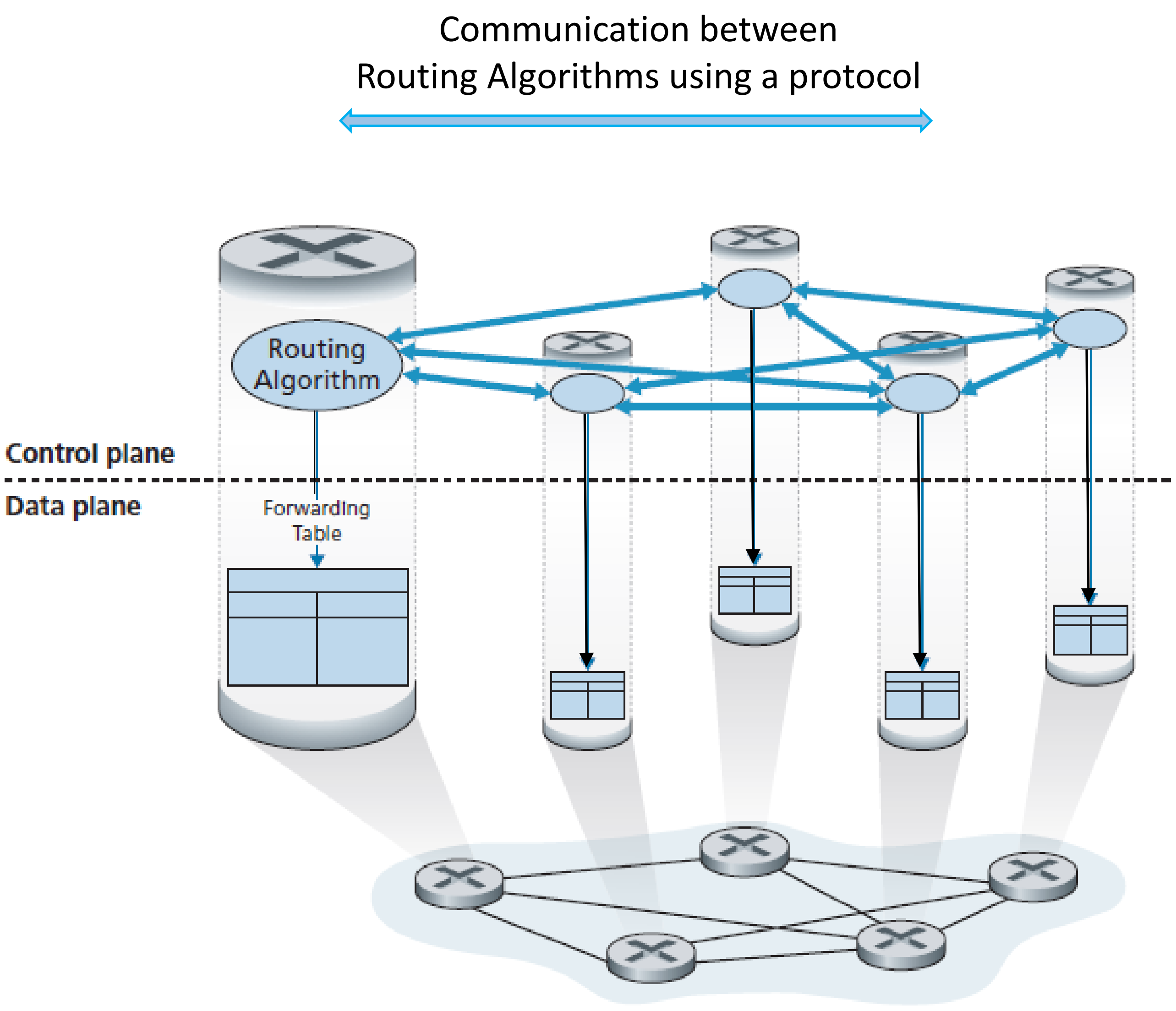
Two approaches:

1- Per-router control: A routing algorithm runs in each and every router (Figure 5.1)

  - Each router has **a routing algorithm** that **uses a protocol to communicates** with routing algorithms in other routers to compute values for its forwarding table
  - OSPF and BGP protocols (Sections 5.3 and 5.4) are per-router control

Figure 5.1

- Per-router control: Individual routing algorithm components interact in control plane
- Routing protocol: A **routing algorithm** together with a **protocol** for communication between routing algorithms



Computing forwarding/flow tables

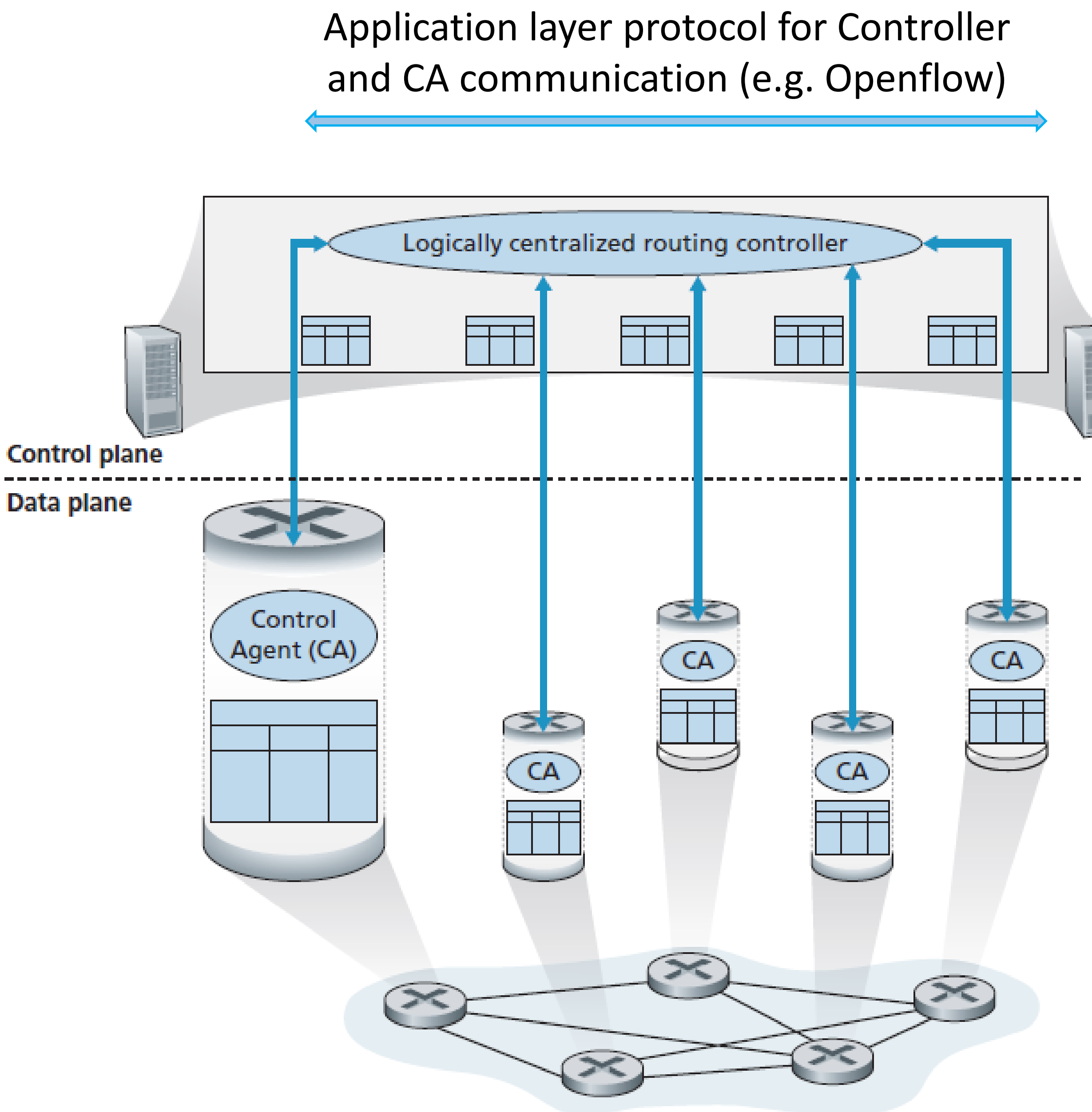
Two approaches:

2- Logically centralized control: A logically centralized controller computes and distributes forwarding tables to be used by each and every router (Figure 5.2)

- Generalized match-plus-action abstraction allows router to perform traditional IP forwarding as well as a rich set of other functions that had been previously implemented in separate middleboxes
- Controller interacts with a control agent (CA) in each of routers via a well-defined protocol to configure and manage that router's flow table
  - CA communicates with controller
  - **CAs do not directly interact with each other** nor do they actively take part in computing forwarding table

Figure 5.2

- Logically centralized control: A distinct, typically remote, controller interacts with local control agents (CAs)



Logically centralized control

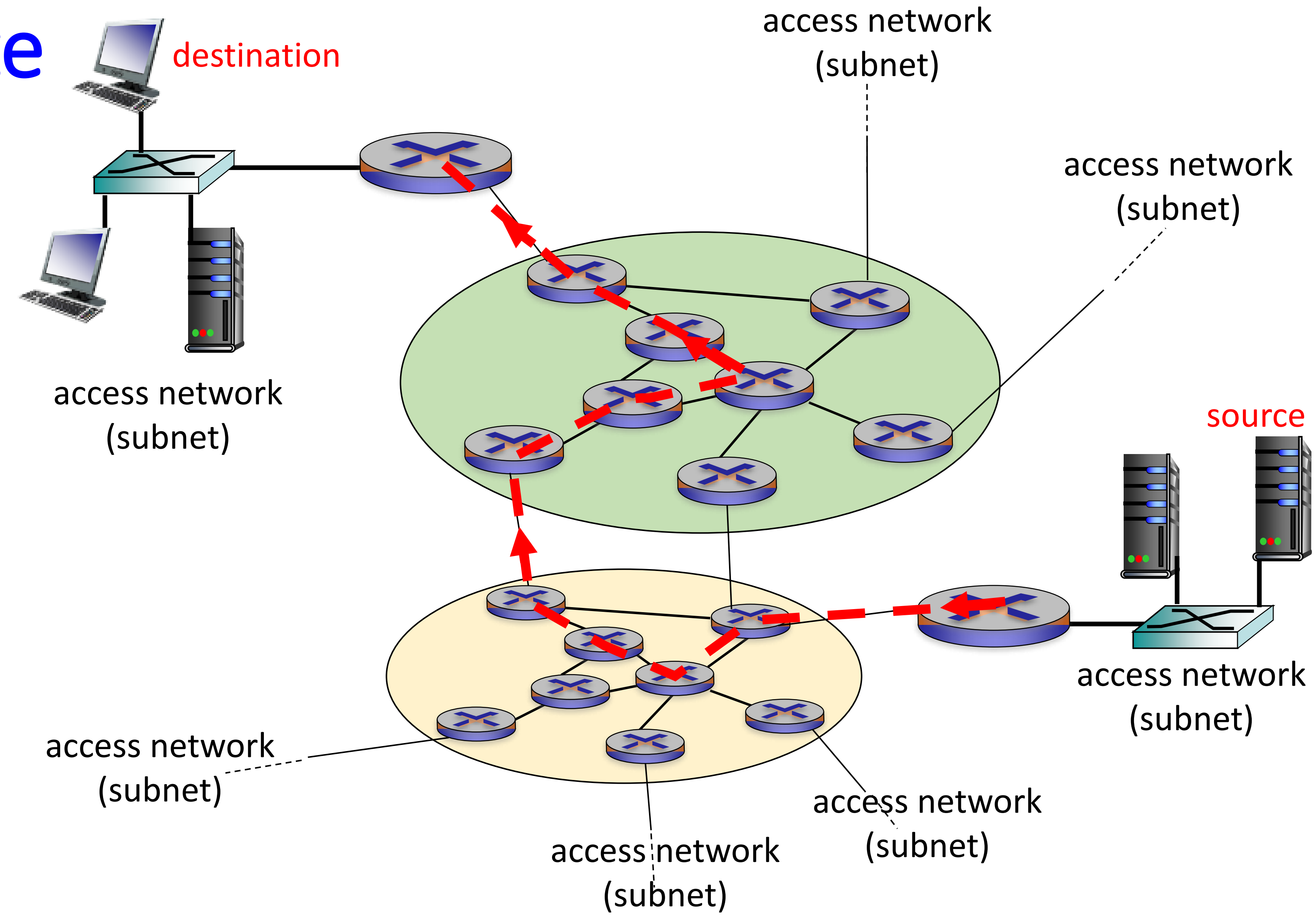
- Service is implemented via **multiple servers** for fault-tolerance, and performance scalability reasons
  - SDN adopts notion of a logically centralized controller, approach that is finding increased use in production deployments
- Google uses SDN to control routers in its internal **B4 global wide-area network** that interconnects its data centers
- **SWAN**, from Microsoft Research, uses a logically centralized controller to manage routing and forwarding between a wide area network and a data center network
- Major ISP deployments, including COMCAST's ActiveCore and Deutsche Telecom's Access 4.0 are actively integrating SDN into their networks
- SDN control is central to 4G/5G cellular networking
- China Telecom and China Unicom are using SDN both within data centers and between data centers

Contents

5.2 Routing Algorithms

- Routing algorithms: determine good paths (routes), from senders to receivers, through network of routers
- “good” path is least cost path or fastest path or least congested path
- However, policy issues (for example, a rule such as “router **x**, belonging to organization **y**, should not forward any packets originating from network owned by organization **z** ”) also come into play
- Routing algorithms are fundamentally important in networking

Path=Route


destination
access network
(subnet)
access network
(subnet)
access network
(subnet)
source
access network
(subnet)
access network
(subnet)
access network
(subnet)
access network
(subnet)

Routing algorithm classification

centralized: all routers have complete topology, link cost info

- "**link state**" algorithms

static: routes change slowly over time

dynamic*:* routes change more quickly

- periodic updates or in response to link cost changes
- load sensitive/insensitive

decentralized: iterative process of computation, exchange of info with neighbors

- routers initially only know link costs to attached neighbors
- "**distance vector**" algorithms

Centralized routing algorithm

- Centralized routing algorithm computes least-cost path between a source and destination using complete, global knowledge about network, (algorithm takes connectivity between all nodes and all link costs as inputs)
- Calculation can be run at one site (e.g., a logically centralized controller) or could be replicated in routing component of each and every router (Figure 5.1)
- Algorithms with global state information are often referred to as link-state (LS), (algorithm must be aware of cost of each link in network)
- SDN uses centralized routing algorithm

Decentralized routing algorithm

- Decentralized routing algorithm, calculation of least-cost path is carried out in an iterative, distributed manner by routers
- Each node begins with knowledge of costs of its own directly attached links
- Then, through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates least-cost path to a destination or set of destinations
- Decentralized routing algorithm is called a distance-vector (DV), (each node maintains a vector of estimates of costs (distances) to all other nodes)
- Decentralized algorithms, with interactive message exchange between neighboring routers is perhaps more naturally suited to control planes where routers interact directly with each other, as in Figure 5.1

Static or Dynamic routing algorithms

- Static routing algorithms: routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a link costs)
- Dynamic routing algorithms change routing paths as network traffic loads or topology change
- A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes
- While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and route oscillation

Load sensitive/insensitive algorithms

- Load-sensitive algorithm, link costs vary dynamically to reflect current level of congestion in underlying link
- If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link
- Internet routing algorithms (such as RIP, OSPF, and BGP) are load-insensitive, as a link's cost does not explicitly reflect its current (or recent past) level of congestion

A graph is used to formulate routing problems

- Graph G = (N, E) is a set N of nodes and a collection E of edges
- Nodes represent routers, edges represent physical links between routers
- In BGP (inter-domain routing protocol) , nodes represent networks, and edge represents direction connectivity (know as peering) between two networks
- To view some graphs representing real network maps, see caida.org
  - Visualizing IPv4 and IPv6 Internet Topology at a Macroscopic Scale in 2020

Figure 5.3

- 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**)}
- c(a,b): cost of *direct* link connecting a and b
- *e.g.*, c(**w**,**z**)=5, c(**u**,**z**)=∞
- Node **y** is said to be a **neighbor** of node **x** if (**x**, **y**) belongs to E
- Cost of path ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_p$) = c($\mathbf{x}_1$, $\mathbf{x}_2$) + c($\mathbf{x}_2$, $\mathbf{x}_3$) + … + ($\mathbf{x}_{p-1}$, $\mathbf{x}_p$)
- 17 possible paths between **u** and **z**

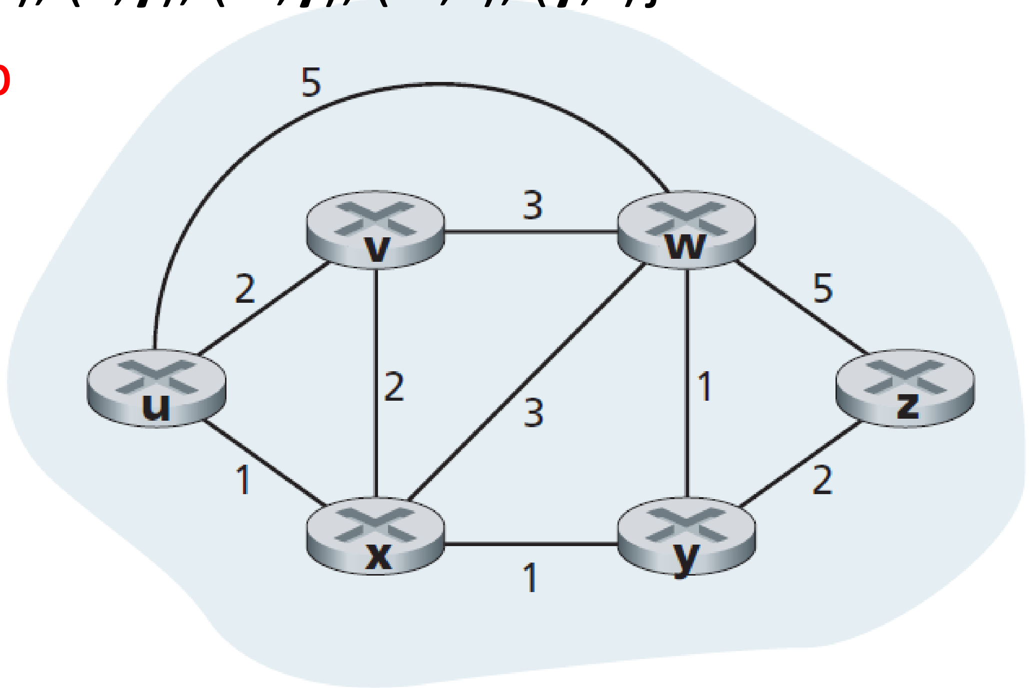


Figure 5.3 Abstract graph model of a computer network

5.2.1 The Link-State (LS) Routing Algorithm

- Network topology and all link costs are available as input to LS algorithm
  - Each node broadcast link-state packets to all other nodes
  - Each link-state packet containing identities and costs of node's attached links
    - e.g. IS-IS uses a variable 1-8 byte system ID (normally 6 bytes) to represent a node in network
    - A router may have a domain name which is maintained by an ADNS (e.g. border1-rt-et-5-0-0.gw.umass.edu)
  - In practice (for example in OSPF routing protocol), this is often accomplished by a link-state broadcast algorithm
- All nodes have an identical and complete view of network
- Each node can then run LS algorithm and compute same set of least-cost paths as every other node
- LS is known as Dijkstra's algorithm, named after its inventor

Link-State (LS) Algorithm for Source Node u

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

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

Complexities

Given N nodes (not counting source)

Algorithm complexity:

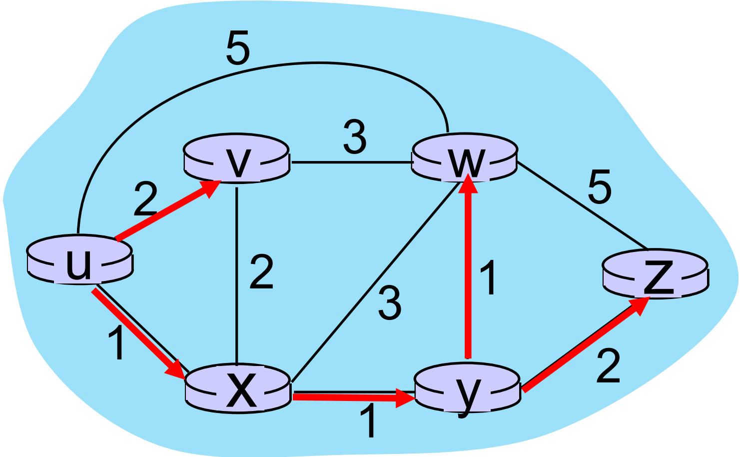
- For: (first iteration) search through all N nodes to determine node w, not in N' that has minimum cost
- In second iteration check N - 1 nodes to determine minimum cost; in third iteration N - 2 nodes, and so on. Overall, total number of nodes we need to search through over all iterations is $N(N + 1)/2$, thus, LS has worst-case complexity of order N squared: $O(N^2)$
- Using a data structure known as HEAP to find minimum in line 9 of algorithm, and reducing complexity

Message complexity:

- Each router must broadcast its link state information to other N routers
- Efficient broadcast algorithms: $O(N)$ link crossings to disseminate a broadcast message from one source
- Each router's message crosses $O(N)$ links and for N nodes overall message complexity is $O(N^2)$

Link-State (LS) Algorithm for Source Node u

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 $D(a) = c_{u,a}$

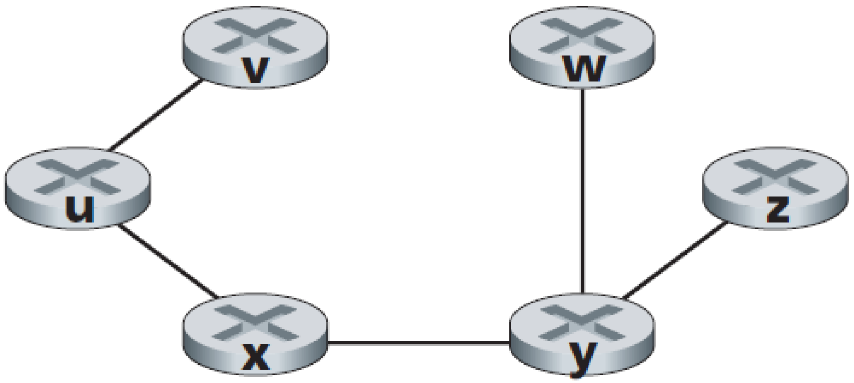
find *a* not in *N′* such that *D(a)* is a minimum

add *a* to *N′*

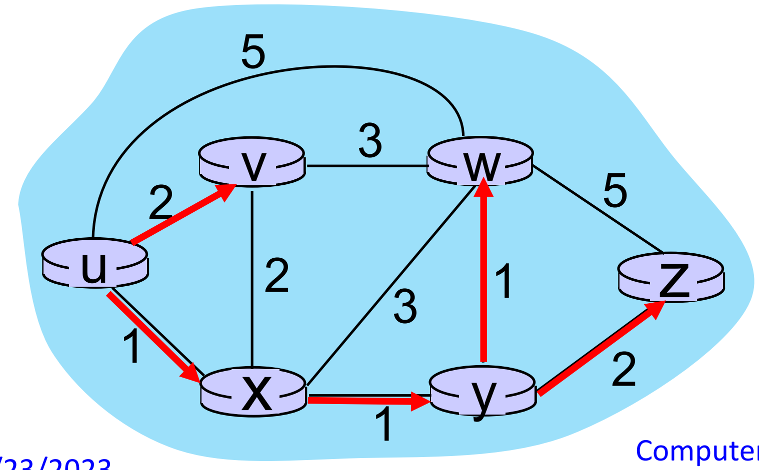
update *D(b)* for all *b* adjacent to *a* and not in *N′* :

$D(b) = \min (D(b), D(a) + c_{a,b})$

Figure 5.4 Least cost path and forwarding table for node u



least-cost-path tree from u



forwarding table in u

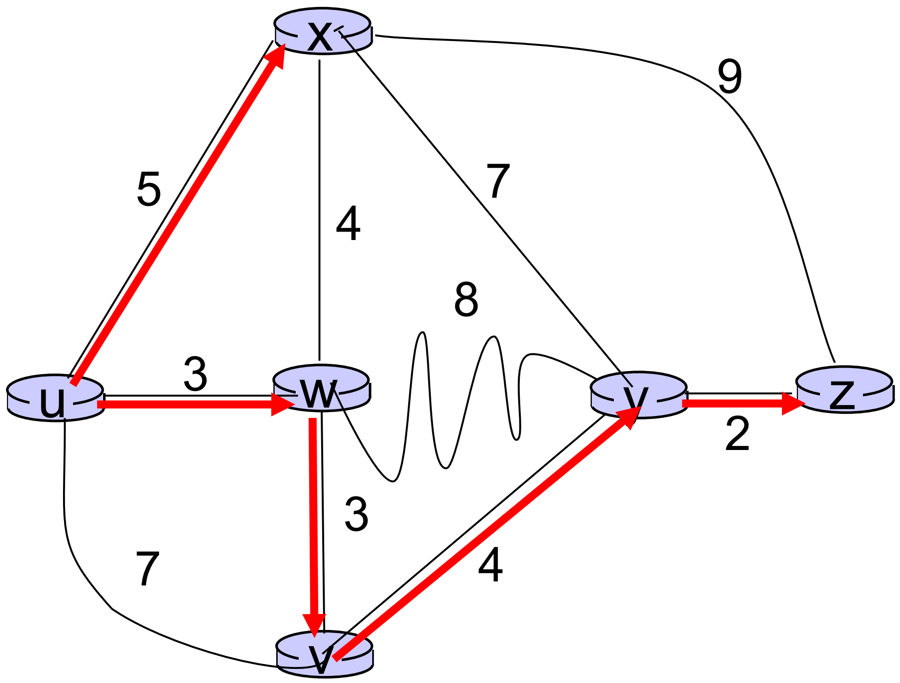
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra’s algorithm: another example for router u

Step	N′	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

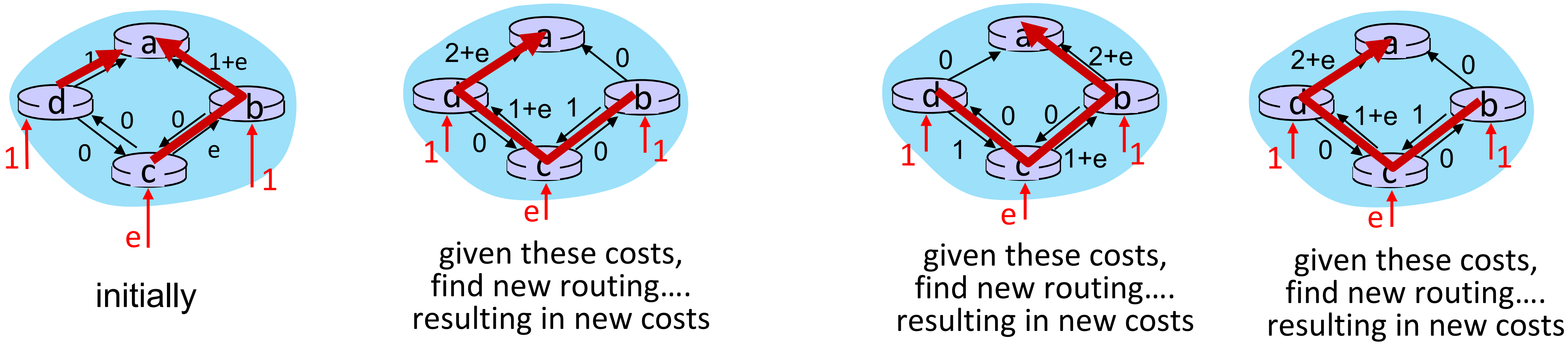


note: ties can exist (can be broken arbitrarily)

Figure 5.5 Oscillations with congestion-sensitive routing

When link costs depend on traffic volume (Load-sensitive algorithm), route oscillations possible

- Sample scenario:
  - Routing to destination a, traffic entering at b and d with rates 1, at c with rate e (<1). Link costs are directional, and load volume-sensitive



5.2.2 Distance-Vector (DV) Routing Algorithm

- A distributed and iterative and asynchronous algorithm

Each node:

- If:
  - Receives **some information (distance vector)** from one or more of its directly attached neighbors
  - There is change in its local link cost
- Then
  - Updates its forwarding table
  - Sends its forwarding table back to its directly attached neighbors

  - This process continues on until no more information is exchanged between neighbors (self-terminating)
  - It does not require all nodes to operate in lockstep with each other

DV algorithm at 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

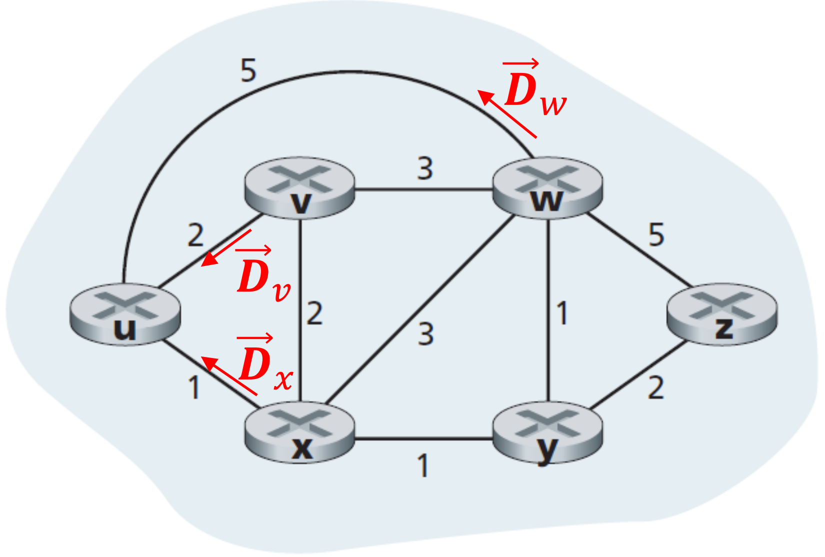
distributed, self-stopping: each node notifies neighbors only when its DV changes

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

Distance vector example

- Distance vector in u: It is the **path cost** from u to all destination nodes $j = v, w, x, y, z$

$$\vec{\boldsymbol{D}}_u = \begin{bmatrix} D_u(v) \\ D_u(w) \\ D_u(x) \\ D_u(y) \\ D_u(z) \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 2 \\ 4 \end{bmatrix}$$



- Node u knows distance vectors of each of its neighbors $i = v, x, w$
  - That is $\vec{\boldsymbol{D}}_v, \vec{\boldsymbol{D}}_w, \vec{\boldsymbol{D}}_x$

Bellman-Ford equation

- Bellman-Ford equation (dynamic programming):

direct cost of link from x to i

$$D_u(j) = min_i\{C_{u,i} + D_i(j)\} \tag{5.1}$$

cost of least-cost path from u to j

cost of least-cost path from i to j

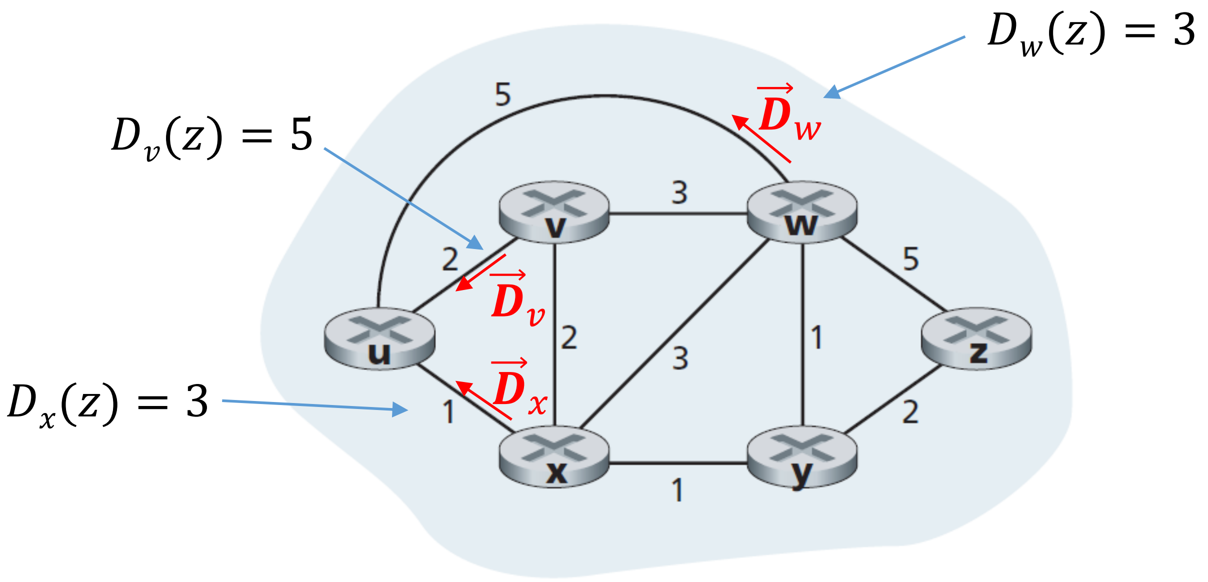
min taken over all neighbors i of u

Bellman-Ford Example for router u

- Suppose that u's neighboring nodes are x, v, w and destination is z

$$D_u(z) = min\{C_{u,v} + D_v(z), C_{u,x} + D_x(z), C_{u,w} + D_w(z)\}$$

$$D_u(z) = min\{2 + 5\ , 1 + 3\ , 5 + 3\} = 4$$



Information in each node with DV algorithm

- Each node u maintains following routing information:

1. For **each neighbor** i, cost $c(u, i)$ from u to directly attached neighbor i

2. Node u's **distance vector**, that is, $\overrightarrow{\boldsymbol{D}}_u = [D_u(j): j \ in \ N]$, containing u's estimate of its cost to all destinations, j, in N (N is set of all nodes in network core except u)

3. Distance vectors of **each of its neighbors**, that is, $\overrightarrow{\boldsymbol{D}}_i = [D_i(j): j \ in \ N]$ for **each neighbor** i of u

Distance-Vector (DV) Algorithm

- From time to time, each node sends a copy of its distance vector to each of its neighbors
- When u receives a new distance vector from any of its neighbors i, it saves w's distance vector, and then uses Bellman-Ford equation to update its own distance vector as follows:

$$D_u(j) = min_i\{C_{u,i} + D_i(j)\} \quad \text{for each node } j \text{ in } N$$

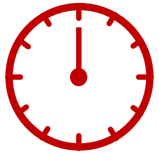
- If $D_u(j)$ has changed, u sends $D_u(j)$ to each of its neighbors, which can in turn update their own distance vectors
- As long as all nodes continue to exchange their distance vectors, each cost estimate $D_u(j)$ converges to actual cost of least-cost path from node u to node j

Distance-Vector (DV) Algorithm

At each node, *x*:

```
Initialization:
  for all destinations y in N:
     D_x(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
  for each neighbor w
     D_w(y) = ? for all destinations y in N
  for each neighbor w
     send distance vector  D_x = [D_x(y): y in N] to w

loop
   wait  (until I see a link cost change to some neighbor w or
              until I receive a distance vector from some neighbor w)

   for each y in N:
       D_x(y) = min_v{c(x,v) + D_v(y)}

if D_x(y) changed for any destination y
      send distance vector D_x  = [D_x(y): y in N] to all neighbors

forever
```

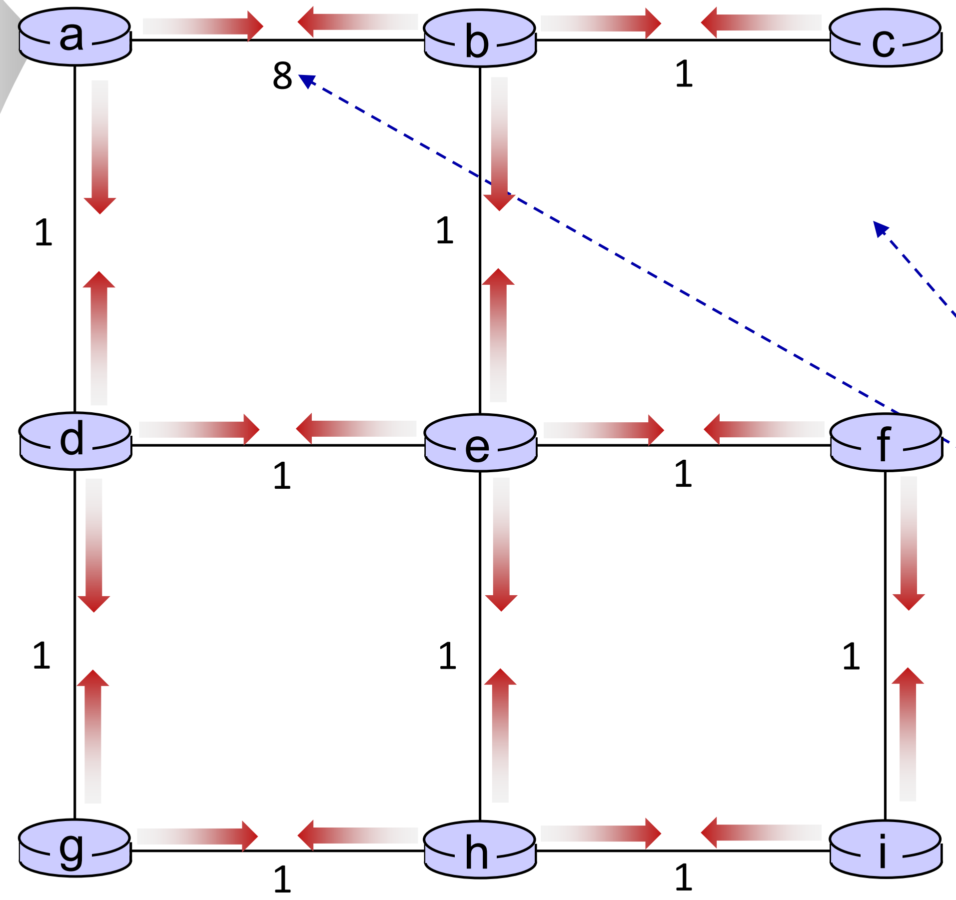
DV example



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

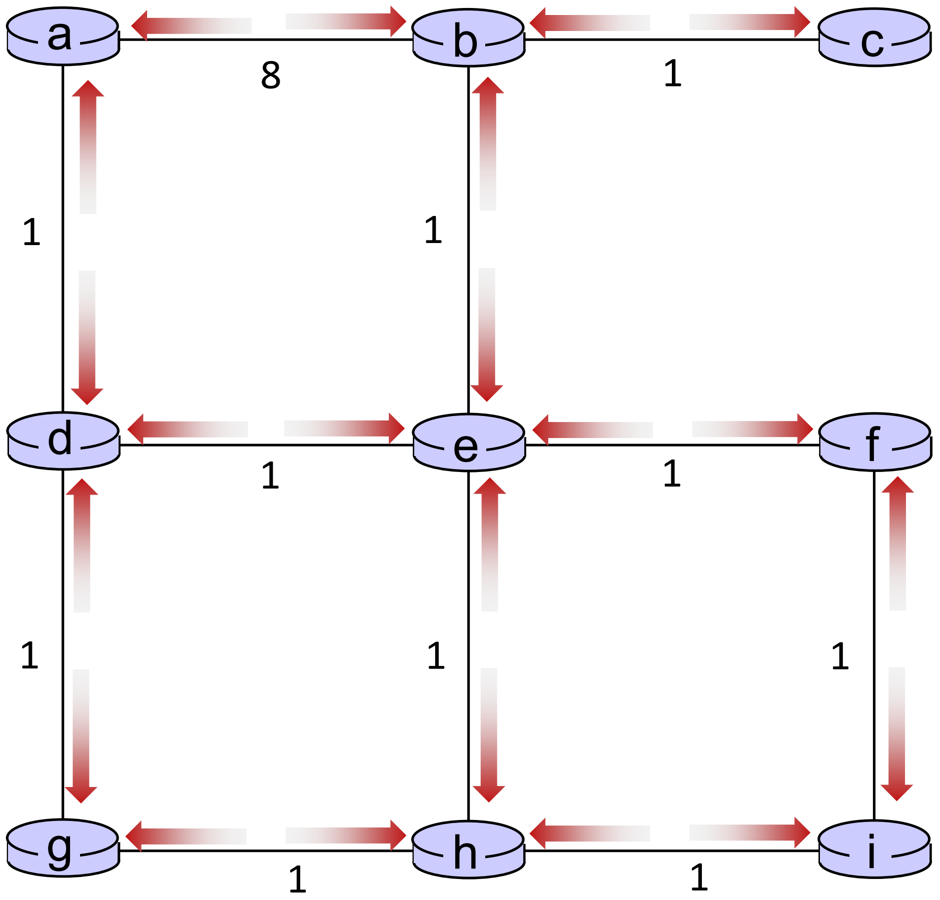


A few asymmetries:
- missing link
- larger cost

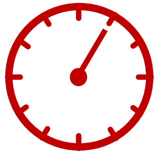
DV example – iteration 1

t=1

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



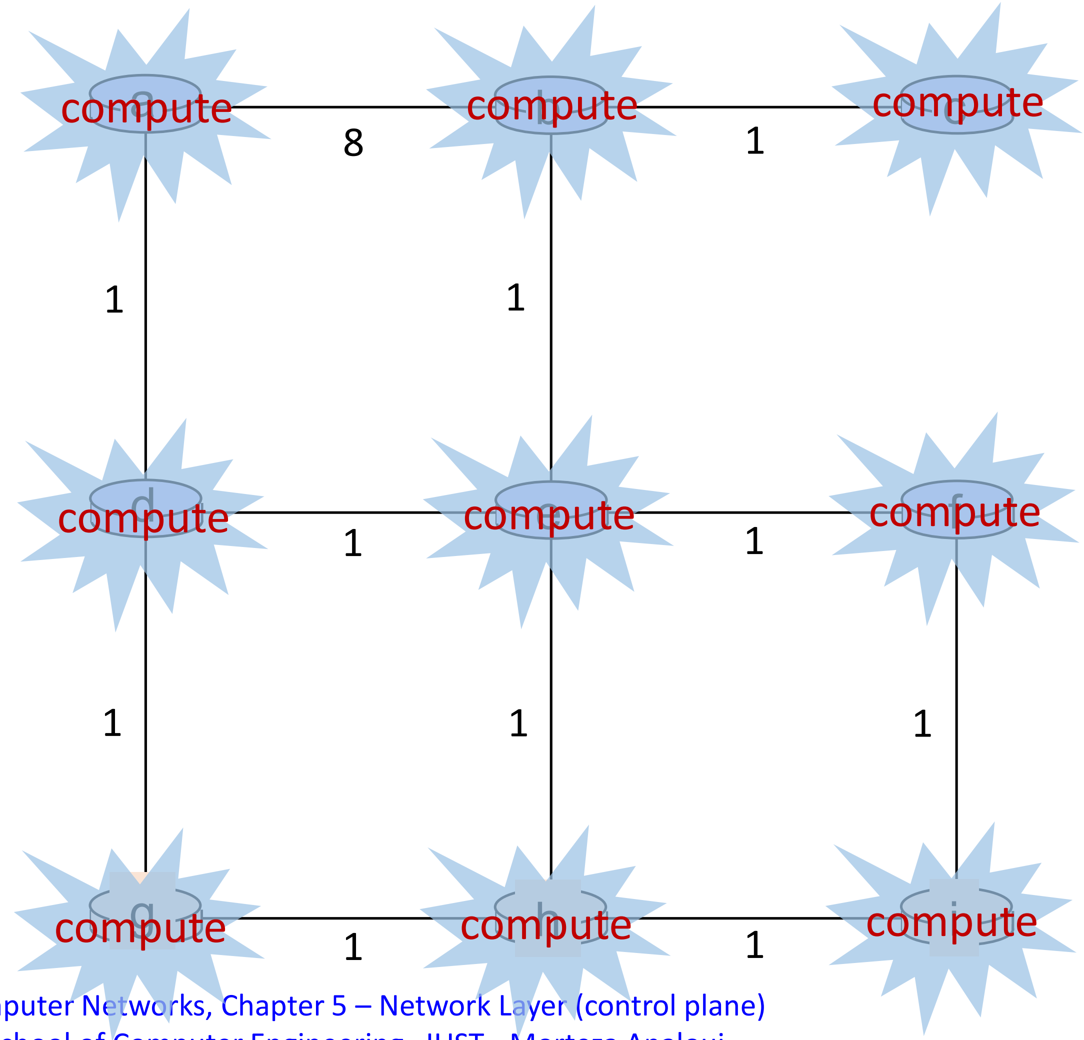
DV example – iteration 1



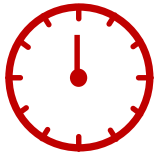
t=1

All nodes:

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



b receives DVs then compute



t=1

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:

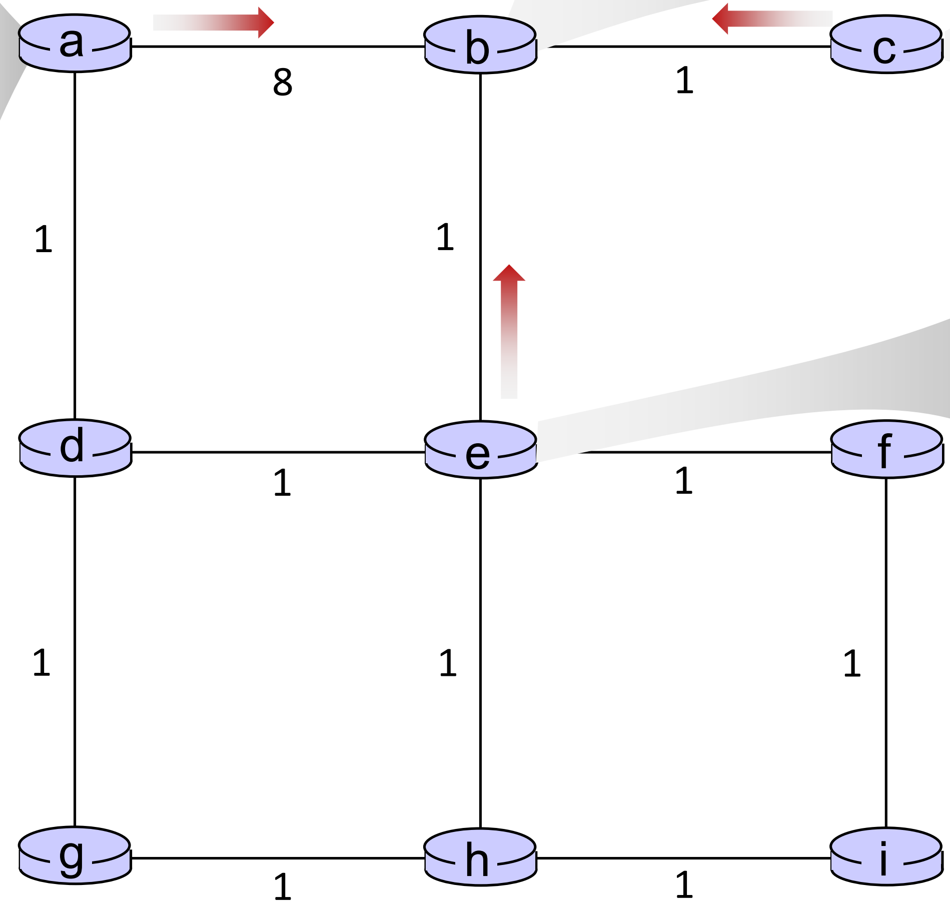
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

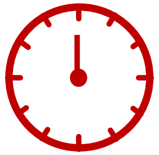
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

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



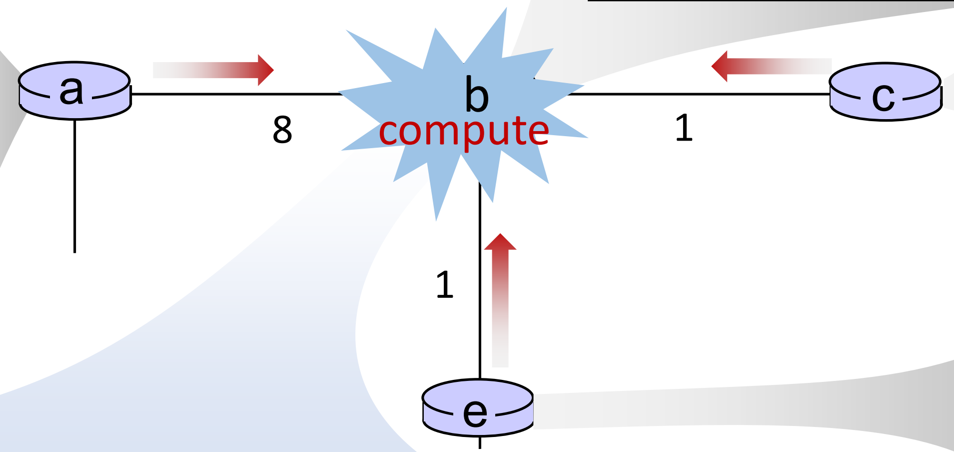
- b receives DVs from a, c, e
- b computes its new local distance vector



t=1

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:

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

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

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

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

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

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

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

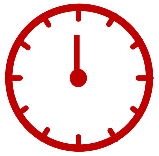
$$D_b(h) = \min\{c_{b,a}+D_a(h),\ c_{b,c}+D_c(h),\ c_{b,e}+D_e(h)\} = \min\{\infty, \infty, 2\} = 2$$

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

DV in b:

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

c receives DV then compute

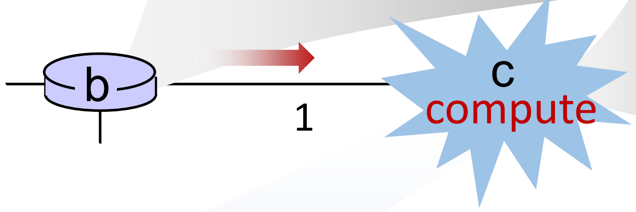


t=1

- c receives DVs from b
- c computes its new local distance vector

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:

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

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

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

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

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

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

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

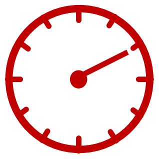
$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

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

DV in c:

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

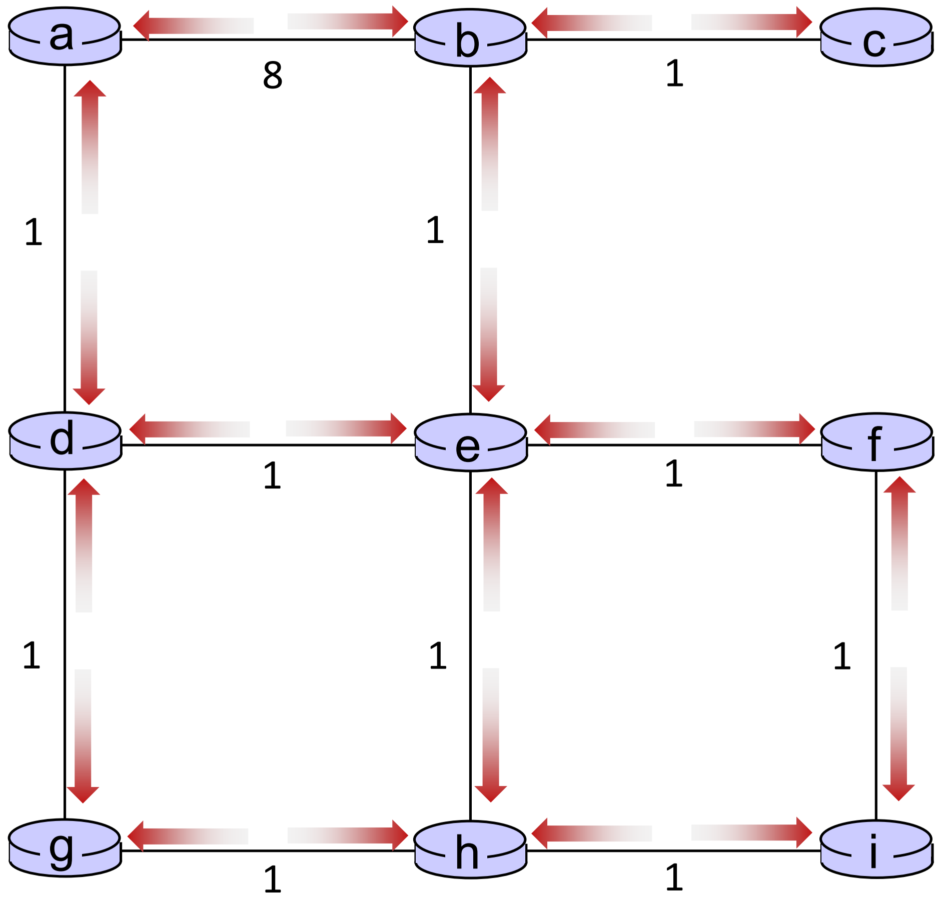
DV example – iteration 2



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: state information diffusion

Iterative communication, computation steps diffuses information through network:

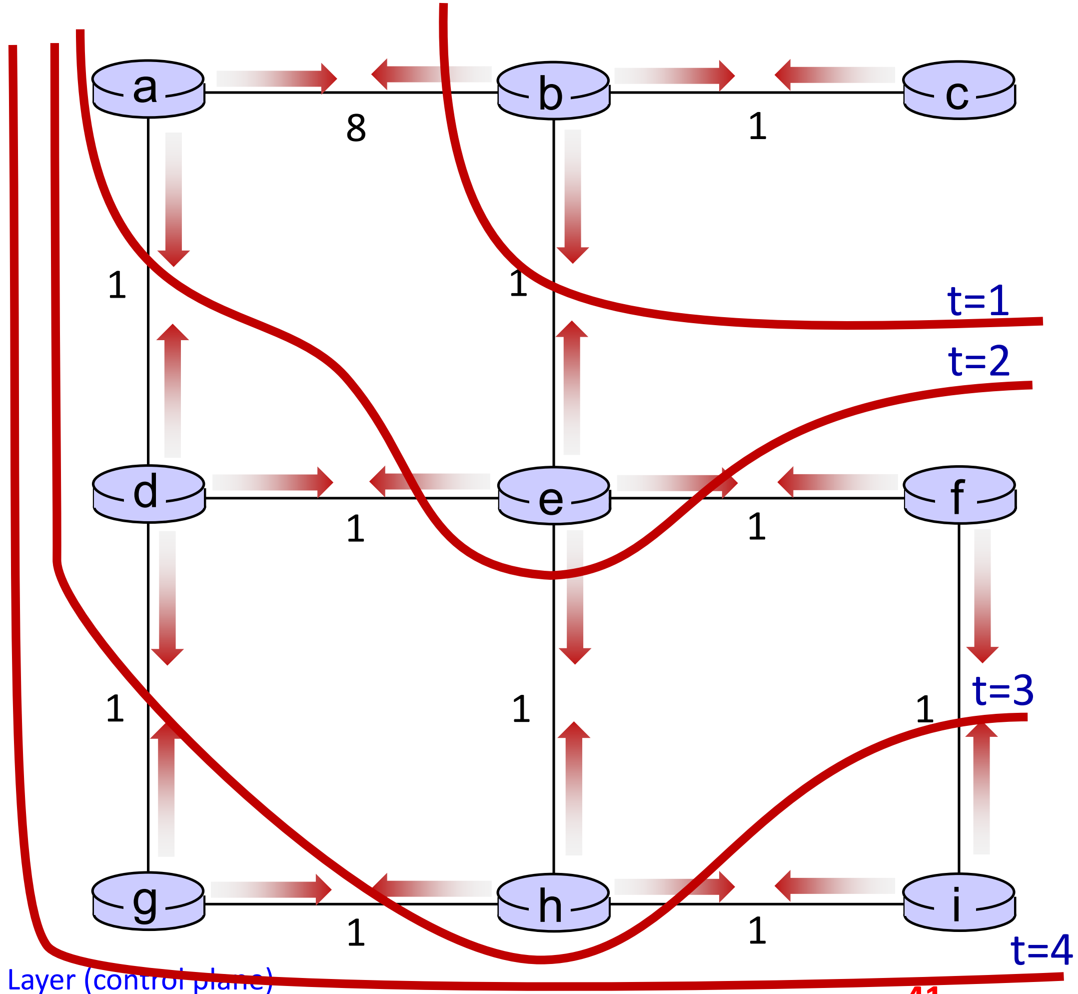
t=0 c's state at t=0 is at c only

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 d,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, d, f, h and now at g,i as well



Distance-Vector Algorithm: Link-Cost Changes

- Figure 5.7(a) illustrates a scenario where link cost from y to x changes from 4 to 1
- We focus here only on y and z distance table entries to destination x ($D_y(x), D_z(x)$)
- DV algorithm causes following sequence of events to occur:

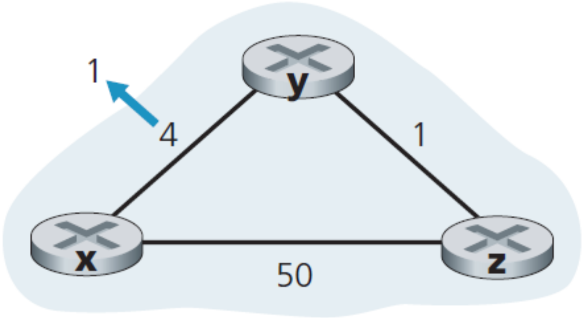


Figure 5.7a Changes in link cost

1. t_0 : y detects link-cost change, updates its DV, informs its neighbors ($D_y(x) = 4 \rightarrow 1$)

2. t_1 : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV ($D_z(x) = 5 \rightarrow 2$)

3. t_2 : y receives z's update, updates its distance table. y's least costs do *not* change $D_y(x) = 1 \rightarrow 1$), so y does *not* send a message to z (only two iterations are required)

Distance-Vector Algorithm: Link-Cost Changes

message exchanges between y and z:

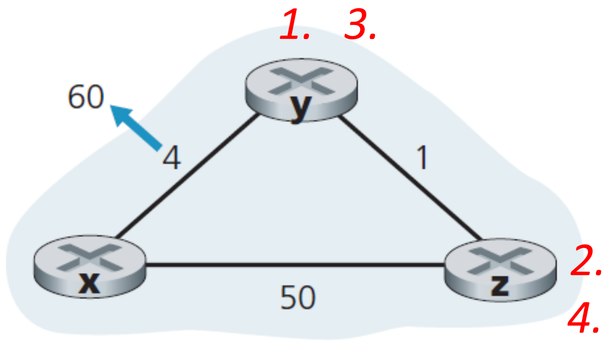


Figure 5.7b Changes in link cost

44 iterations are required

1. y sees direct link to x has new cost 60, but z has said it has a path to x at cost of 5. So y computes "my new cost to x will be **6**, via z; notifies z
2. 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
3. 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
4. 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

…

44. y learns that path to x via z has new cost 49, so y computes "my new cost to x will be 8 via y), notifies z of new cost of **50** to x. Now, z computes cost of its path via y to be greater than 50

- When cost change is very high, it takes a long time to converge; count-to-infinity problem

A Comparison of LS and DV Routing Algorithms

Routers Communication

- LS: each node would need to communicate with all other nodes (via broadcast), it tells them only costs of its directly connected links

- DV: each node talks to only its directly connected neighbors, it provides its neighbors with least-cost estimates from itself to all nodes (that it knows about) in network

A Comparison of LS and DV Routing Algorithms

Message complexity

Recall that N is set of nodes (routers) and E is the set of edges (links)

- LS: each node need to know cost of each link in network. This requires O($|N|\ |E|$) messages to be sent. Whenever a link cost changes, new link cost must be sent to all nodes
- DV: message exchanges between directly connected neighbors at each iteration. Iterations to converge depends on many factors. When link costs change, DV algorithm will propagate results of changed link cost only if new link cost results in a change

A Comparison of LS and DV Routing Algorithms

Speed of convergence

- LS: we have seen that our implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N|\ |E|)$ messages
- DV: it can converge slowly and can have routing loops while algorithm is converging. DV also suffers from count-to-infinity problem

A Comparison of LS and DV Routing Algorithms

Robustness. What can happen if a router fails, misbehaves, or is sabotaged?

- LS: a router could broadcast an incorrect cost for one of its attached links. A node could also corrupt or drop any packets it received as part of an LS broadcast
- Route calculations are separated under LS, providing a degree of robustness
- DV: a node can advertise incorrect least-cost paths to any or all destinations. An incorrect node calculation can be diffused through entire network under DV

Contents

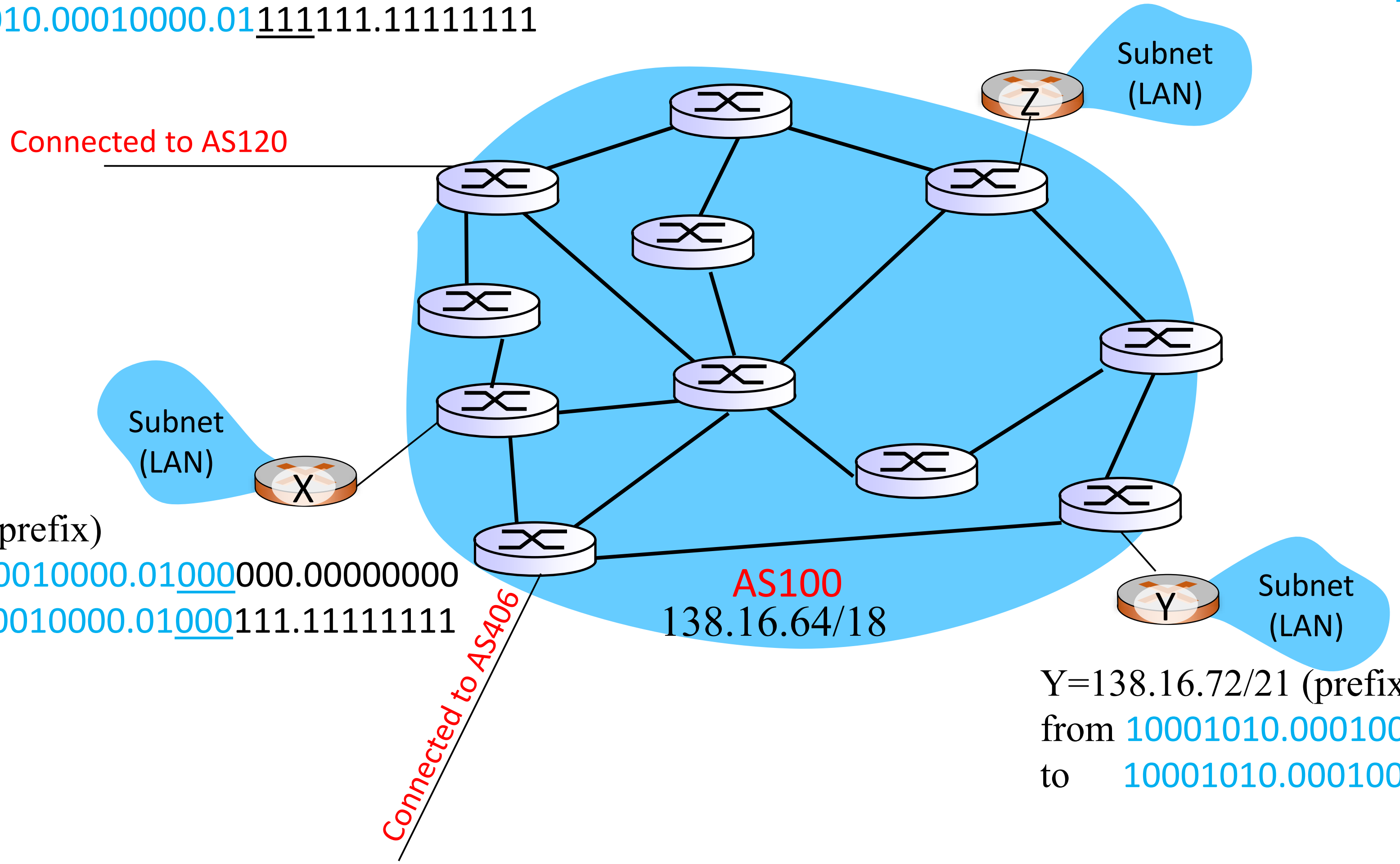
Autonomous systems: AS

- An AS consisting of a group of routers that are under same administrative control
- Some ISPs partition their network core into multiple ASs
- Some tier-1 ISPs use one gigantic AS for their entire network, whereas others break up their ISP into tens of interconnected ASs
- An AS is identified by its globally unique autonomous system number (ASN) [RFC 1930]. AS numbers are assigned by ICANN regional registries
- Routers within same AS all run same routing algorithm (called an intra-autonomous system routing protocol) and have information about each other

Autonomous systems: AS

AS100=138.16.64/18 (prefix)
from 10001010.00010000.01000000.00000000
to 10001010.00010000.01111111.11111111

Z=138.16.160/21 (prefix)
from 10001010.00010000.01010000.00000000
to 10001010.00010000.01010111.11111111



X=138.16.64/21 (prefix)
from 10001010.00010000.01000000.00000000
to 10001010.00010000.01000111.11111111

Y=138.16.72/21 (prefix)
from 10001010.00010000.01001000.00000000
to 10001010.00010000.01001111.11111111

5.3 Intra-AS Routing in the Internet: OSPF

- Open Shortest Path First (OSPF) routing: its specification is publicly available
- Most recent version of OSPF, version 2, [RFC 2328]
- OSPF contains a link-state routing algorithm and a protocol to communicate with OSPF in other routers
- OSPF algorithm:
  - Broadcasts link-state information (link cost and up/down status of link) to all other routers in AS
  - Also broadcasts a link-state periodically (at least once every 30 minutes), even if link's state has not changed
  - OSPF messages are carried directly by IP, with an **upper-layer protocol of 89**, so, implements functionality such as reliable message transfer and link-state broadcast

Open Shortest Path First (OSPF)

- OSPF algorithm:
  - Individual link costs are configured by network administrator
  - Each router constructs a complete topological map of entire AS
  - Each router runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as root node
  - When multiple paths to a destination have same cost, OSPF allows multiple paths to be used (a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist)

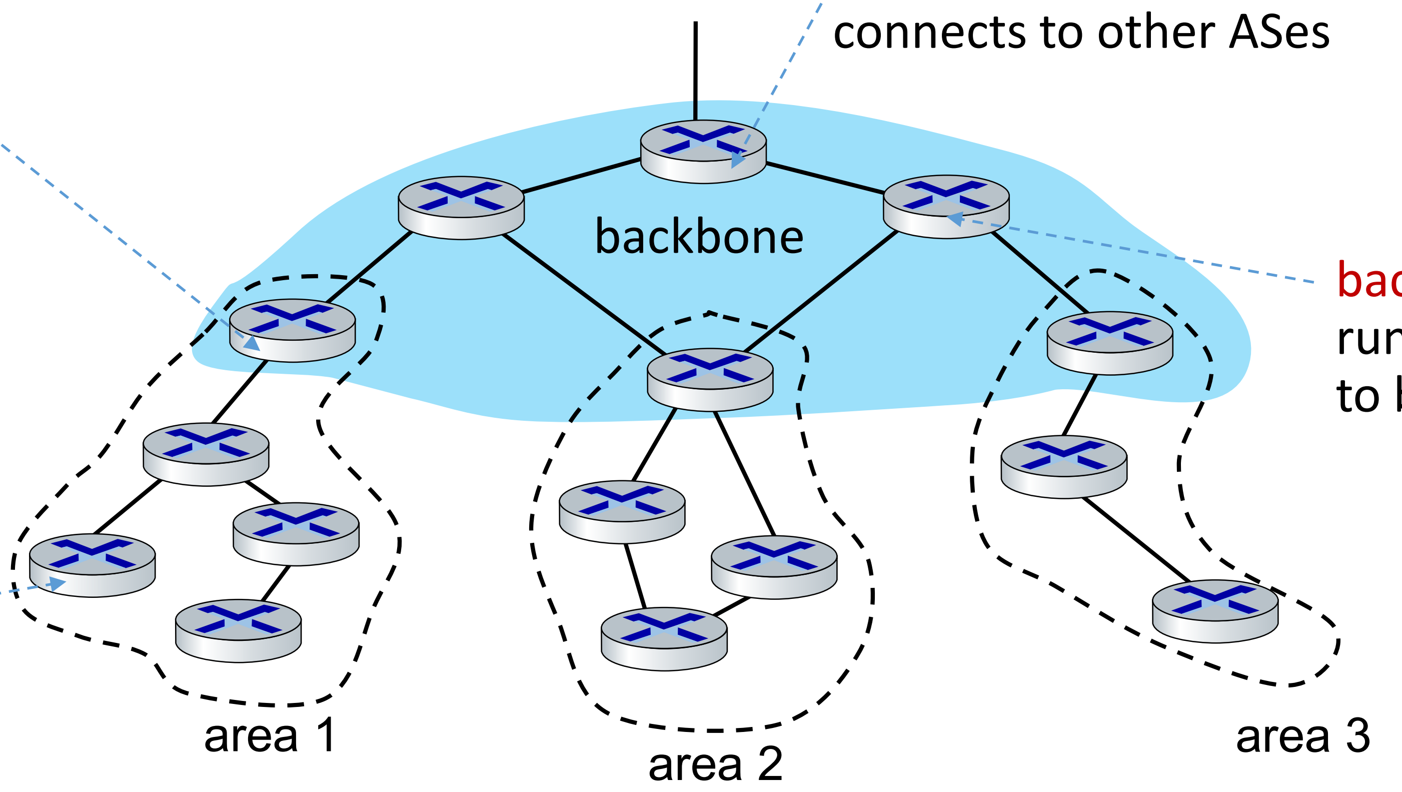
Open Shortest Path First (OSPF)

- OSPF protocol:
  - Checks that links are operational (via a HELLO message that is sent to an attached neighbor)
  - Obtains a neighboring router's database of network-wide link state
  - Security- Exchanges between OSPF routers can be authenticated (only trusted routers can participate in OSPF protocol within an AS)

Support for hierarchy within a single AS

- An OSPF autonomous system can be configured hierarchically into areas
- Each area runs its own OSPF, router in an area broadcasting its link state to all other routers in that area
- One or more area border routers are responsible for routing packets outside area
- One OSPF area in AS is configured to be backbone area
- Backbone routes traffic between areas
- Backbone contains all area border routers
- **Inter-area routing**: packets first routed to an area border router (intra-area routing), then routed through backbone to area border router that is in destination area, and then routed to final destination

Support for hierarchy within a single AS



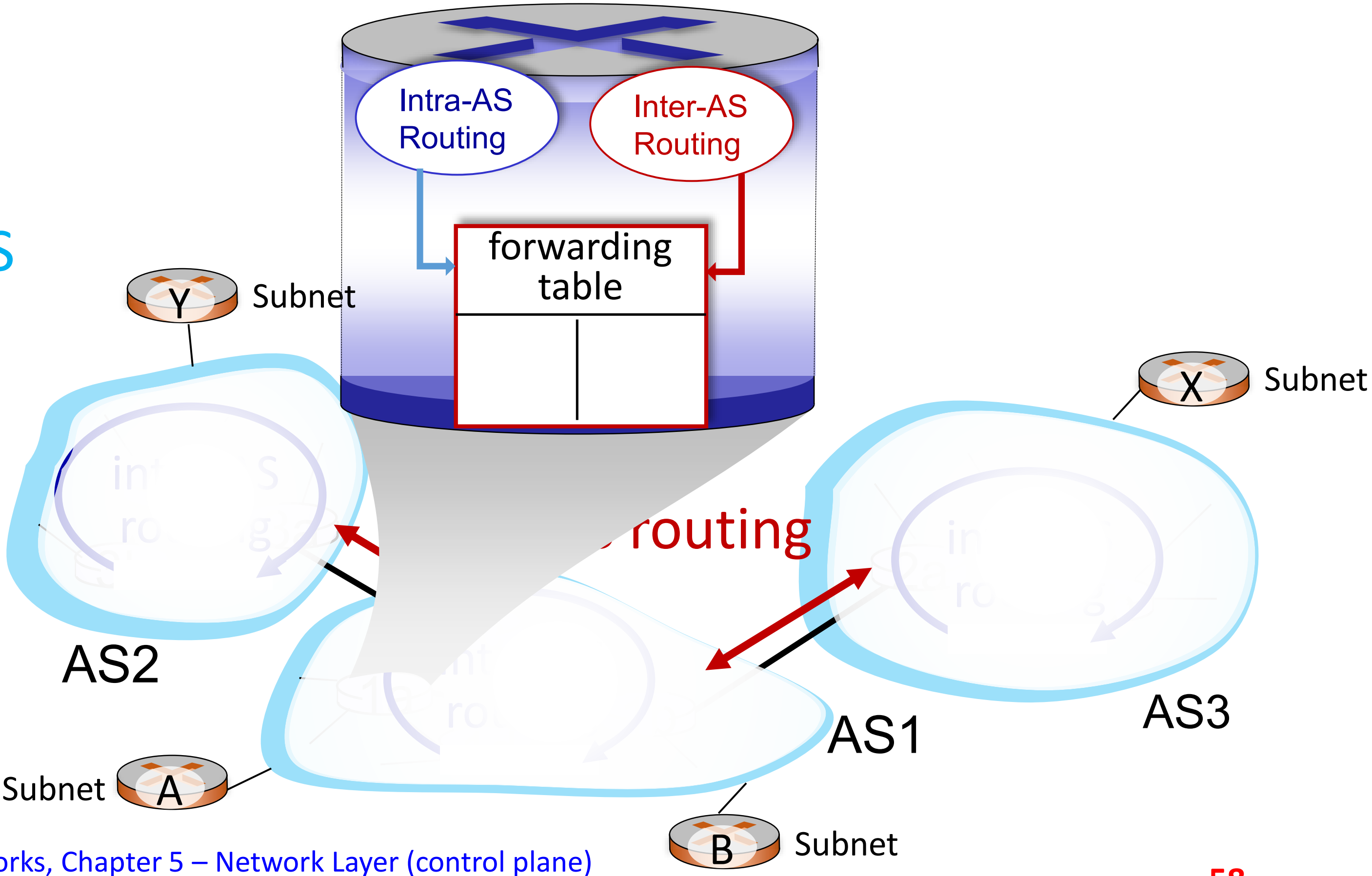
local routers:
- flood LS in area only
- compute routing within area
- forward packets to outside via area border router

Some intra AS routing protocols

- OSPF: Open Shortest Path First [RFC 2328]
  - link-state routing
- 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])

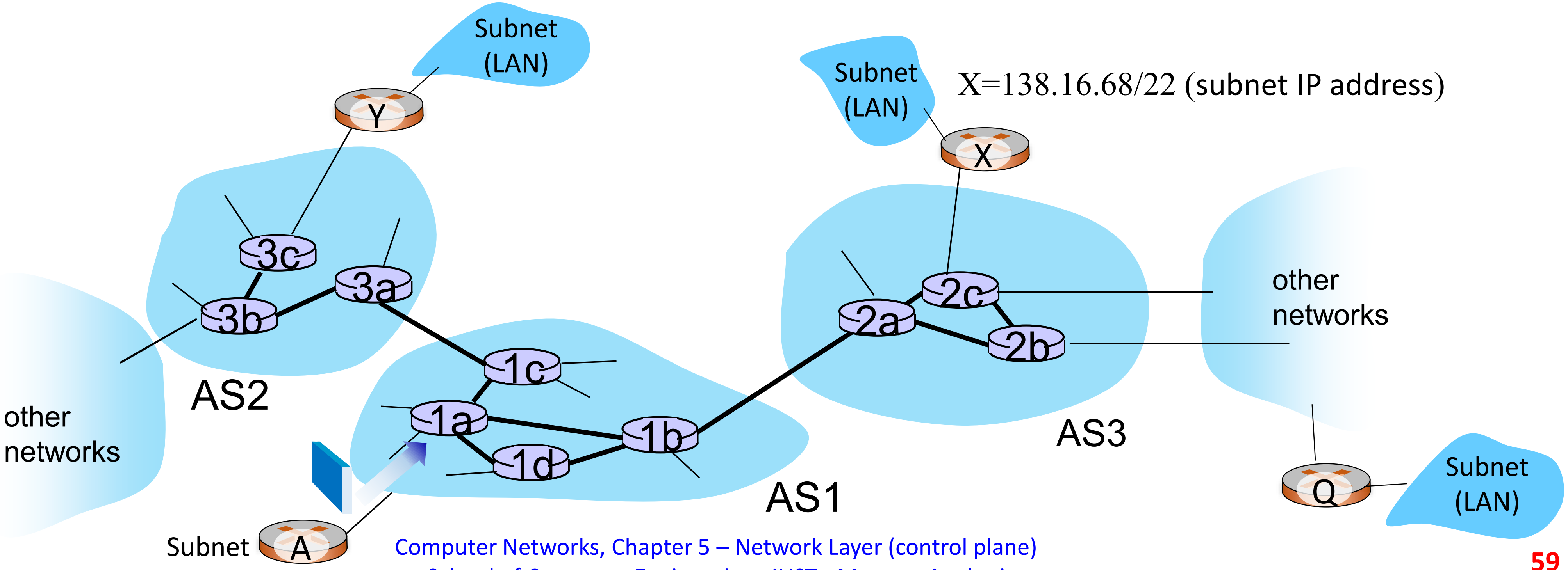
Internet approach to scalable routing

- Forwarding table configured by intra-AS routing algorithm and inter-AS routing algorithm
- Intra-AS routing determine entries for destinations within AS
- Intra-AS routing algorithms within an AS communicate, but no communication with other ASs routing algorithms
- Inter-AS & Intra-AS determine entries for external destinations



Inter-AS routing

- Suppose 1a router receives datagram destined outside of AS1
- 1a router should forward packet to gateway router (boundary router) in AS1, but which one? 1c or 1b ?



Contents

5.4 Routing Among the ISPs: BGP

- To route a packet across multiple ASs, we need an inter-AS routing protocol
- Since an inter-AS routing protocol involves coordination among multiple ASs, communicating ASs must run **same** inter-AS routing protocol
- In Internet, all ASs run **same inter-AS routing protocol**, called Border Gateway Protocol, BGP [RFC 4271]
- BGP is protocol that **glues thousands of ISPs** in Internet together
- BGP is a **decentralized** and **asynchronous** protocol in vein of Distance-Vector routing

5.4.1 The Role of BGP

BGP provides each router within an AS:

1. A **protocol**: to obtain prefix reachability of all subnets in Internet
   - BGP protocol is used to advertise prefix IP address of a subnet. IP prefix will reach to **all routers of all ASs in Internet**
2. **A route selection algorithm:** to determine “best” routes to all prefixes in Internet
   - A router may learn about two or more different routes to a specific prefix
   - To determine best route, each router **will locally run a BGP route-selection algorithm**
   - Best route will be determined based on policy and reachability information

Gateway routers, Internal routers

- Routers: gateway router and internal router
- In AS1, router 1c is a gateway router; routers 1a, 1b, and 1d are internal routers

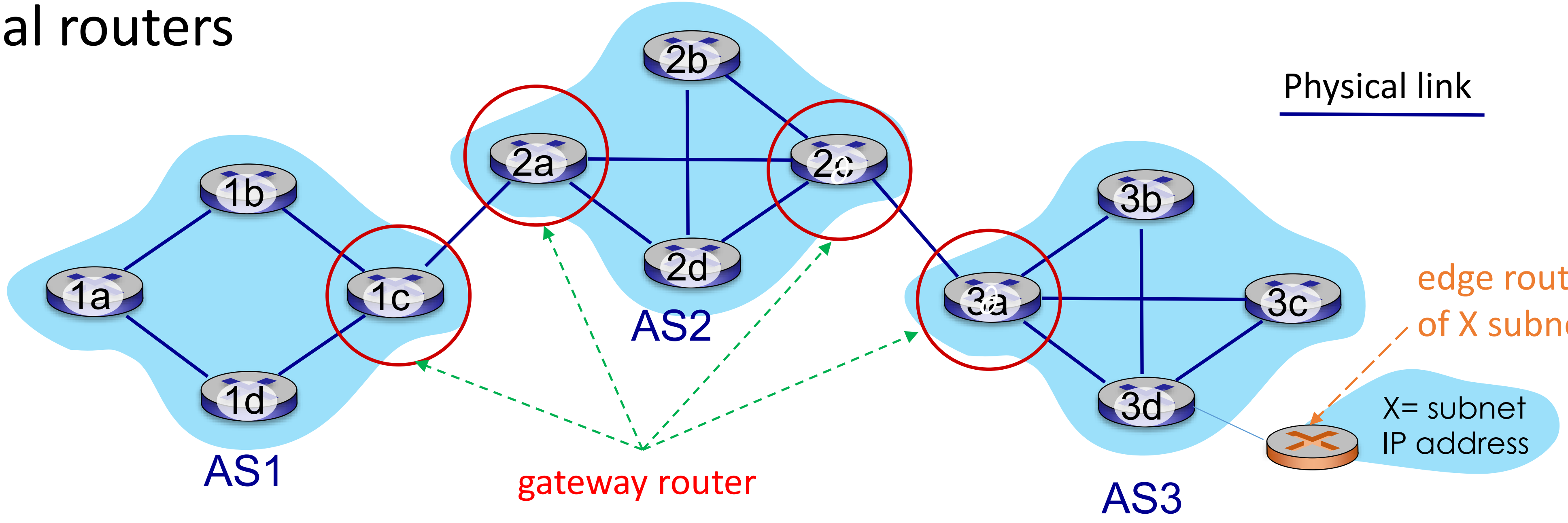
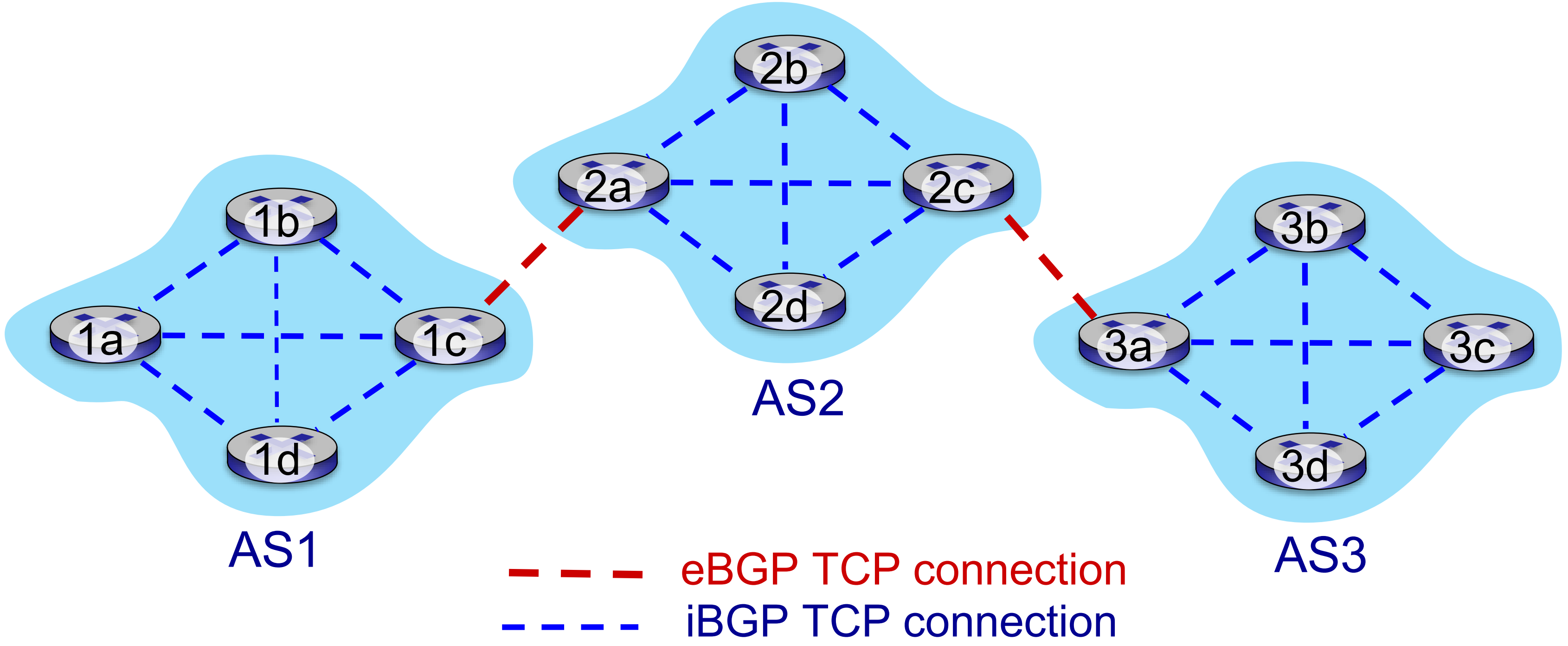


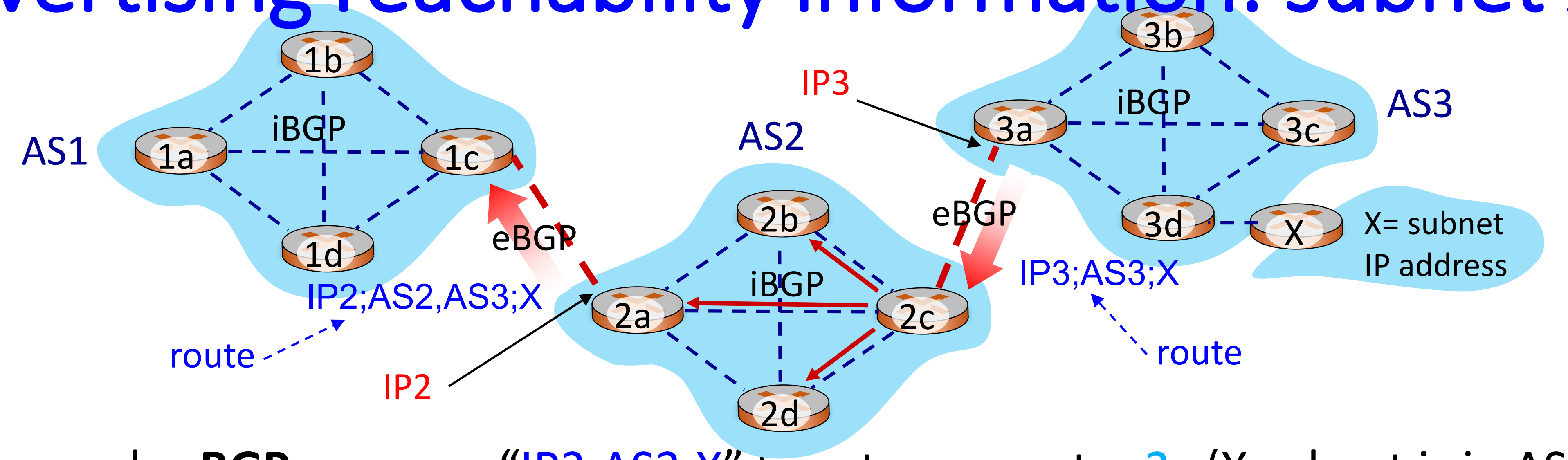
Figure 5.8 Network with three autonomous systems. AS3 includes a subnet with prefix X

5.4.2 Advertising BGP Route Information

- **BGP protocol:** external BGP (eBGP) and internal BGP (iBGP)
- eBGP and iBGP use TCP to setup **BGP connections** between Routers



Advertising reachability information: subnet X



1. 3a sends **eBGP** message “IP3;AS3;X” to gateway router 2c (X subnet is in AS3)
2. 2c sends **iBGP** message “IP3;AS3;X” to all of other routers in AS2, including to gateway router 2a (X is reachable from 2c, and route is IP3;AS3;X)
3. 2a sends **eBGP** message “IP2;AS2,AS3;X” to gateway router 1c (X is reachable using route: IP2;AS2,AS3;X)
4. 1c uses **iBGP** to send message “IP2;AS2,AS3;X” to all the routers in AS1 (X is …)

Multipath to a subnet

- Figure 5.10, which is original network in Figure 5.8, with an additional physical link from router 1c to router 3a

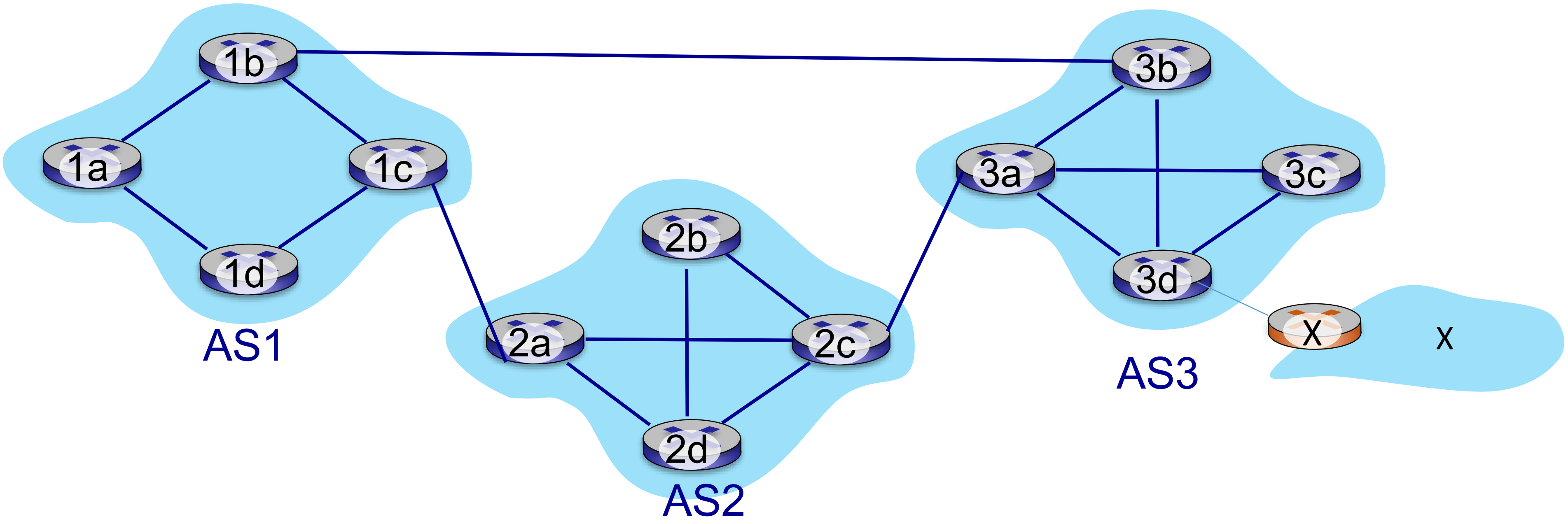
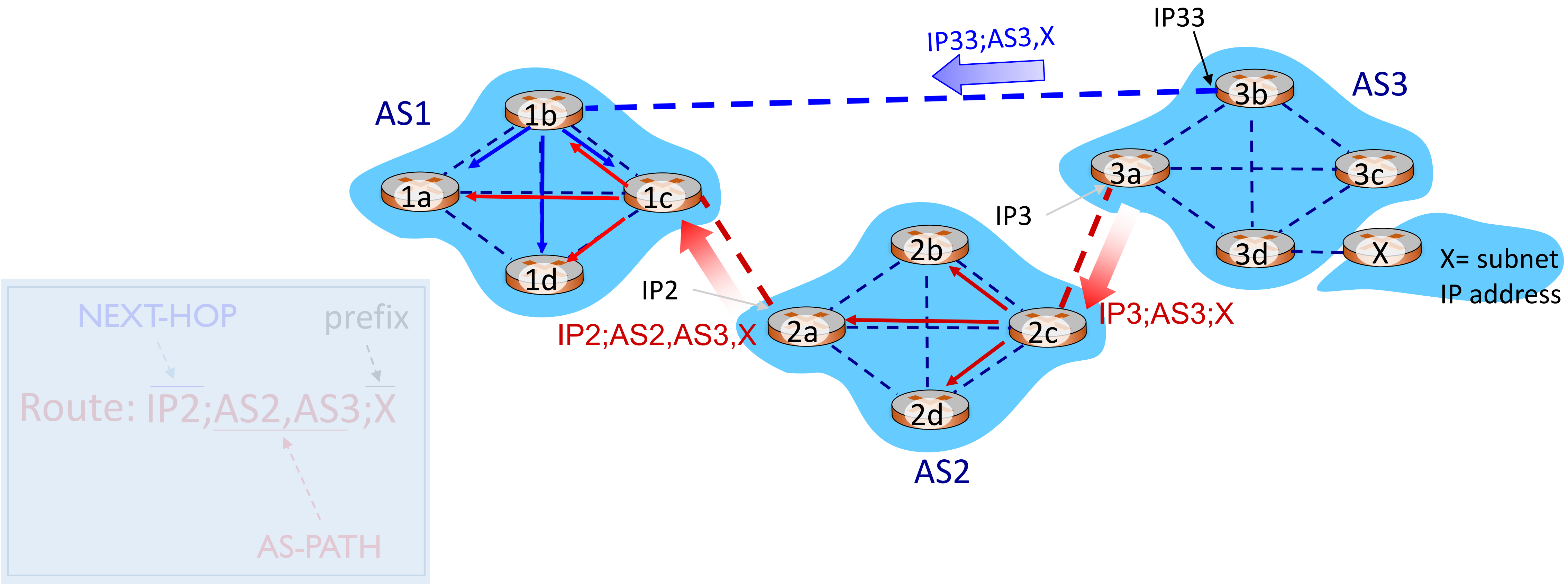


Figure 5.10 Network augmented with peering link between AS1 and AS3

Multipath to a subnet

- Two paths to subnet X: IP2;AS2,AS3;X and IP33;AS3,X

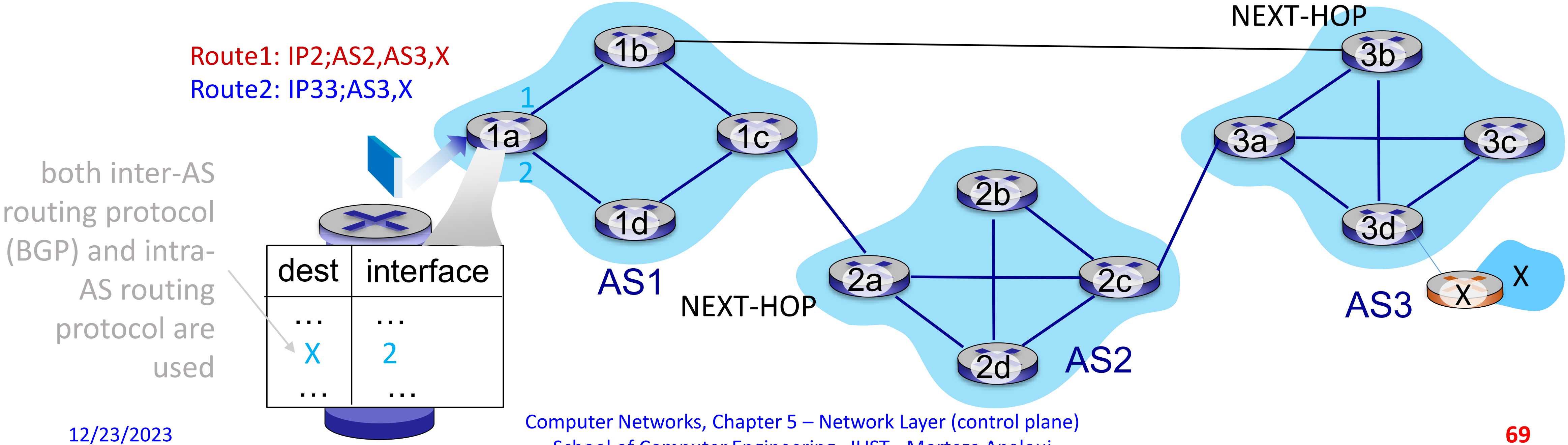


5.4.3 Determining the Best Routes

- How does a router choose among paths and then configure its forwarding table accordingly?
- We begin with one of simplest routing algorithms, namely, hot potato routing

Hot Potato Routing

- 1a receives a packet with destination to X subnet. 1a knows the destination is out of AS1
- **Hot potato routing** uses reachability information (there two 2 paths to X, one is through 1b and another is through 1c). Then, **Intra-AS routing** calculates least costs path to router 2a and to router 3b, finally, selects route with smallest of these least-cost paths
- If cost to 2a be less than to 3b then packet will send out of interface 2 and **NEXT-HOP in AS-PATH** will be 2a



Hot Potato Routing

- Hot-potato routing: get packets out of AS with least cost possible without worrying about cost of remaining portions of path outside of AS to destination
- It is a selfish algorithm. It tries to reduce cost in its own AS while ignoring other components of end-to-end costs outside its AS
- In practice, BGP uses an algorithm that is more complicated than hot potato routing, but nevertheless incorporates hot potato routing

- Note: two routers in same AS may choose two different AS paths to same prefix

BGP path selection Algorithm

- When there are several path to a subnet, BGP invokes following elimination rules until one route remains:

1. First eliminate gateway routes that are not complied with **local preference** values (policy)
2. From remaining gateway routes (all with same highest local preference value), route with shortest AS-PATH is selected. BGP uses a DV algorithm **Cost (distance) is number of AS hops**
3. From selected gateway routes (all with same highest local preference value and same AS-PATH length), **hot potato routing** is used
4. If more than one route still remains, router uses BGP identifiers to select route; see [Stewart 1999]

14 Day BGP Profile:

16-May-2022 00:00 - 29-May-2022 23:59 (UTC+1000)

Number of BGP Update Messages:	21125623
Number of Prefix Updates:	8933485
Number of Prefix Withdrawals:	310073
Average Prefixes per BGP Update:	0.44
Average BGP Update Messages per second:	17.47
Average Prefix Updates per second:	7.64
Peak BGP Update Message Rate per second:	133842
Peak Prefix Update Rate per second:	10005
Peak Prefix Withdraw Rate per second:	77340
Prefix (subnet) Count:	937235
Updated Prefix (subnet) Count:	937234
Stable Prefix (subnet) Count:	1
Origin AS Count:	73462
Updated Origin AS Count:	73427
Stable Origin AS Count:	35
Unique Path Count:	546595
Updated Path Count:	460796
Stable Path Count:	85799

5.4.5 Routing Policy

- Gateway routes are first selected according to local-preference **attribute**, whose value is fixed by policy of local AS

Basic concepts of BGP routing, simple example:

- Figure 5.13: six interconnected ASs: A, B, C, W, X, and Y
- X is a **multi-homed access ISP**, since it is connected to two different providers

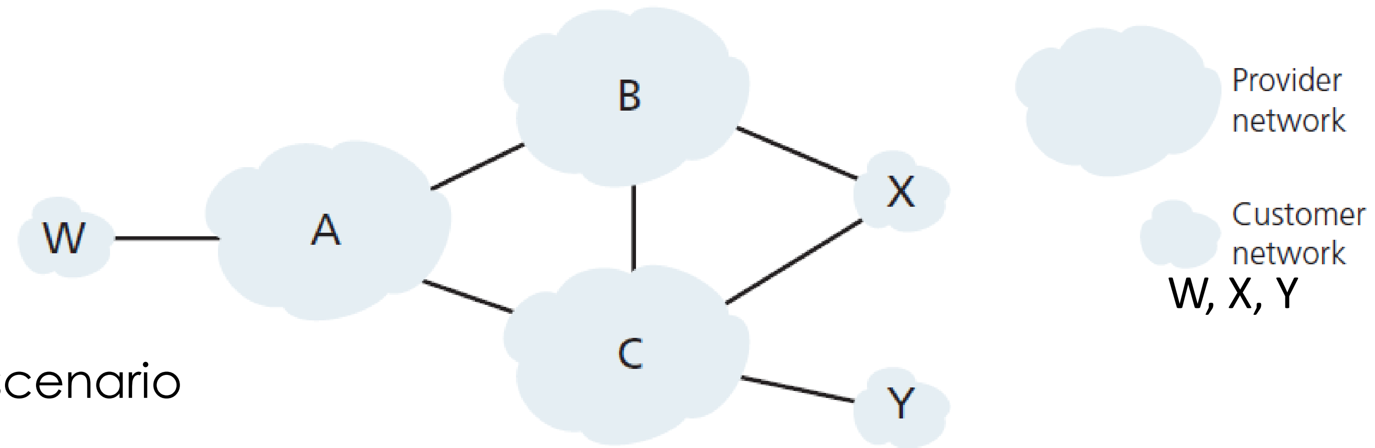
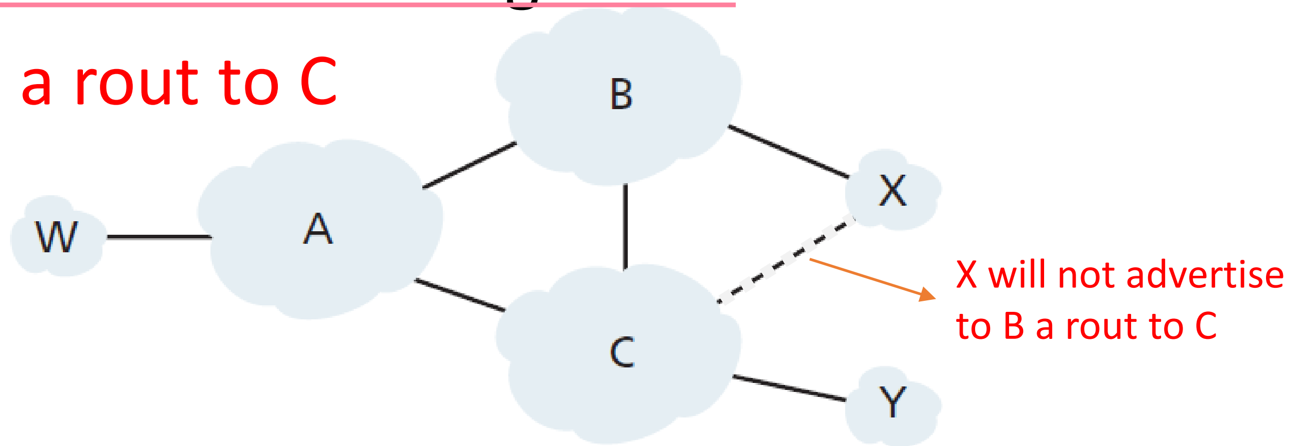


Figure 5.13 A simple BGP policy scenario

Policy in customer ISP networks

Routing policy reflects commercial relationships among ISPs

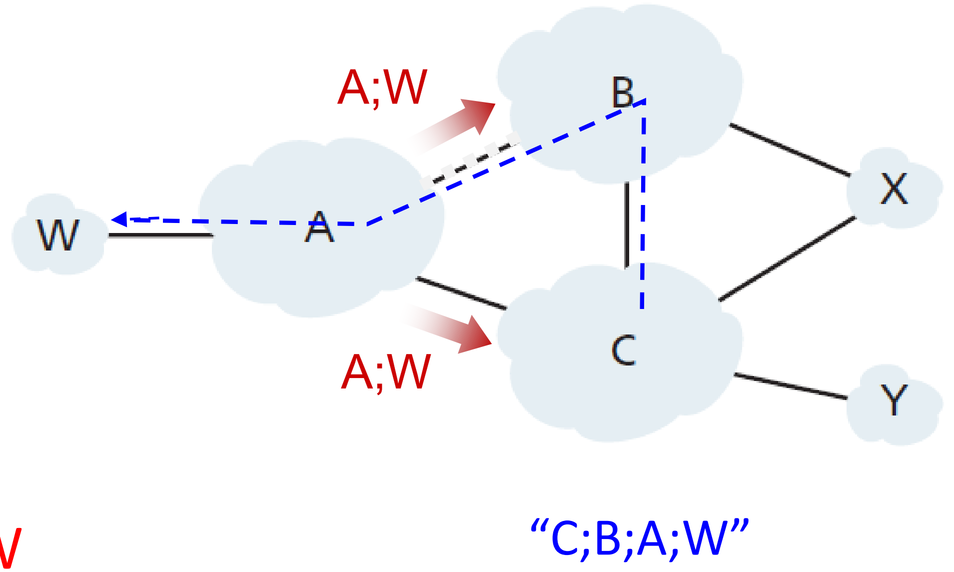
- Assume:
  - A, B, and C, directly send traffic to each other, and provide full BGP information to their customer networks
  - All traffic entering a customer ISP must **has destination on that customer ISP**
  - All traffic leaving a customer ISP must **have originated in that customer ISP**
- **Q**: How will X be prevented from forwarding traffic from B to C?
- **A**: X will not advertise to B a rout to C



Policy in provider ISPs

B policy to enforce:

- A advertises path "A;W" to B and to C
- B gets no "revenue" for routing "C;B;A;W", since none of C, A, W are B's customers
- B chooses not to advertise "B;A;W" to C
- C does not learn about "C;B;A;W" path
- C will route "C;A;W" (not using B) to get to W



C does not learn about "C;B;A;W" path

- Routing policy reflects commercial relationships among ISPs

5.4.6 Putting the Pieces Together: Obtaining Internet Presence

- **Suppose you have created a company**
- You like to have a public Web server that describes your products and services, a mail server for employees, and an authoritative DNS server

1. First obtain Internet connectivity by contracting with, and connecting to, a local ISP. An edge router in your network is connected to a router in your local ISP
   - This connection might be a DSL connection through existing telephone infrastructure, a leased line to ISP's router, or one of many other access solutions (Chapter 1)
2. Your local ISP provides you an IP address range, for example, a /24 address range consisting of 256 addresses

- You assign one of IP addresses to your Web server, one to your mail server, one to your DNS server, one to your gateway router, and other IP addresses to other servers and networking devices in your company's network

Obtaining Internet Presence

3. You will also contract with an Internet registrar to obtain a domain name for your company
    - Example, if your company's name is, imy Inc., you will naturally try to obtain domain name imy.com or imy.ir
4. Your company must also obtain presence in DNS system, because customers will want to contact your ADNS server to obtain IP addresses of your servers
    - So, You provide your registrar with IP address and domain name of your authoritative DNS server. Your registrar will then put an entry for your DNS server in .ir top-level-domain servers

Obtaining Internet Presence

- Each router in Internet needs to know about existence of your company's /24 prefix (or some aggregate entry)

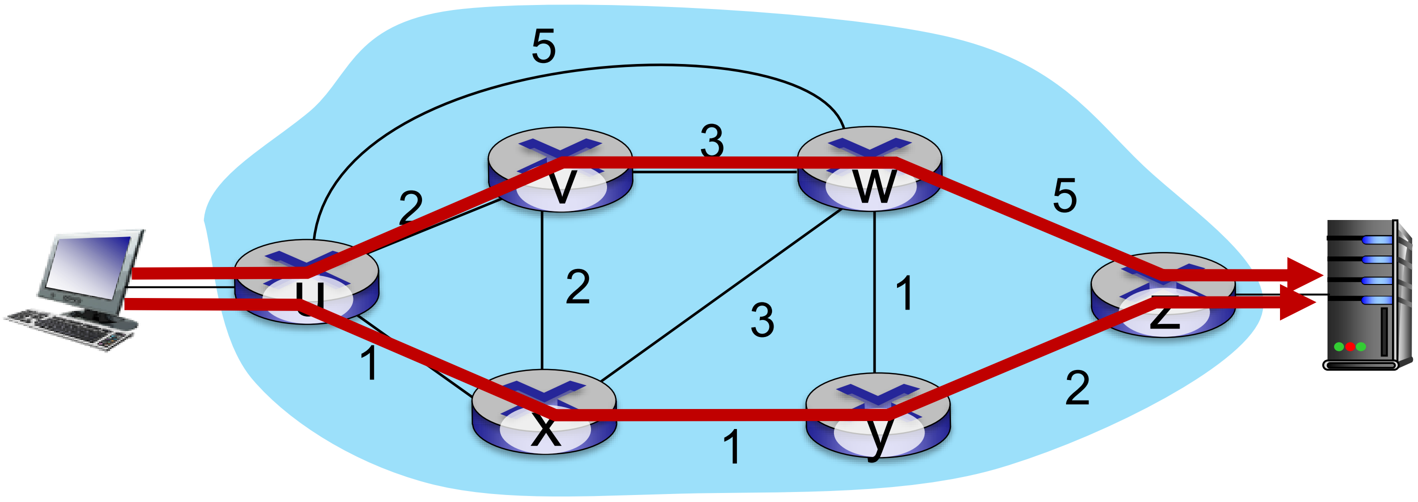
5. Your local ISP will use BGP to advertise your prefix to ISPs to which it connects
    - Those ISPs will then, in turn, use BGP to propagate advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers

Contents

Example: Traffic engineering

Traditional routing:

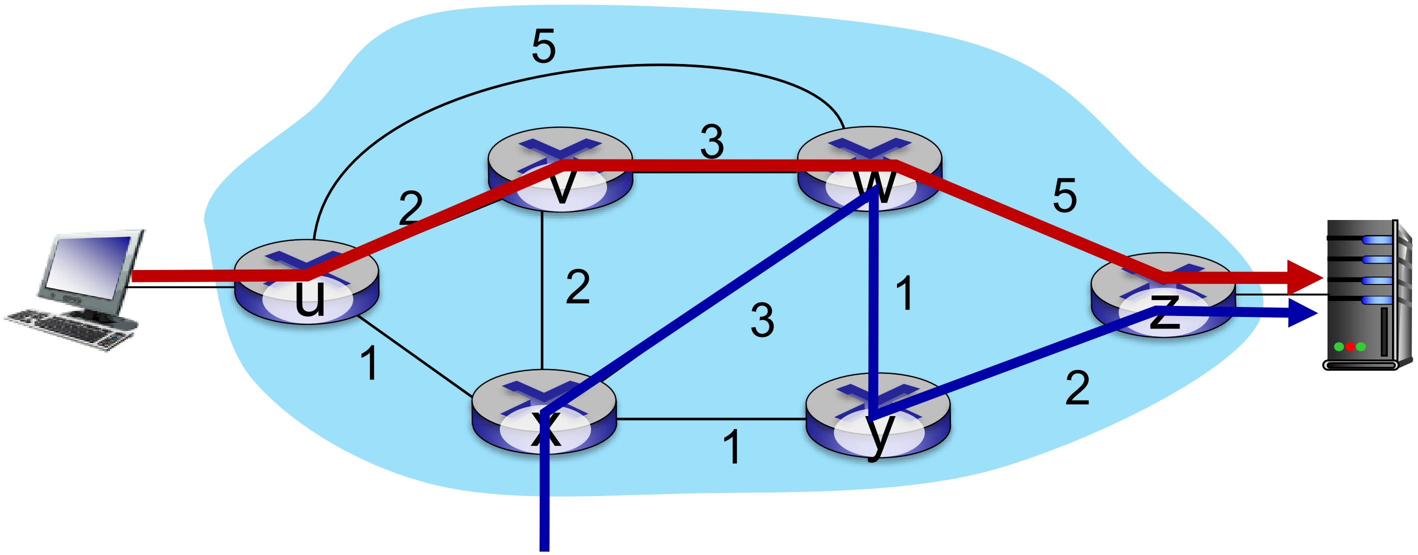
- What if network operator wants u-to-z traffic to flow along uvwz, rather than uxyz? Re-define link weights
- Traditional routing can not do it



Example: Traffic engineering

Traditional routing:

- What if w wants to route blue and red traffic differently from w to z?
  can't do it



5.5 The SDN Control Plane

Four key characteristics of an SDN architecture:

1. Flow-based forwarding

   forwarding by packet switches can be based on any number of header field values in transport-layer, network-layer, or link-layer header

2. Separation of data plane and control plane

   Data plane: packet switches, relatively simple but fast devices, that execute "match plus action" rules in their flow tables. Control plane: servers and software that **determine and manage** switches' **flow tables**

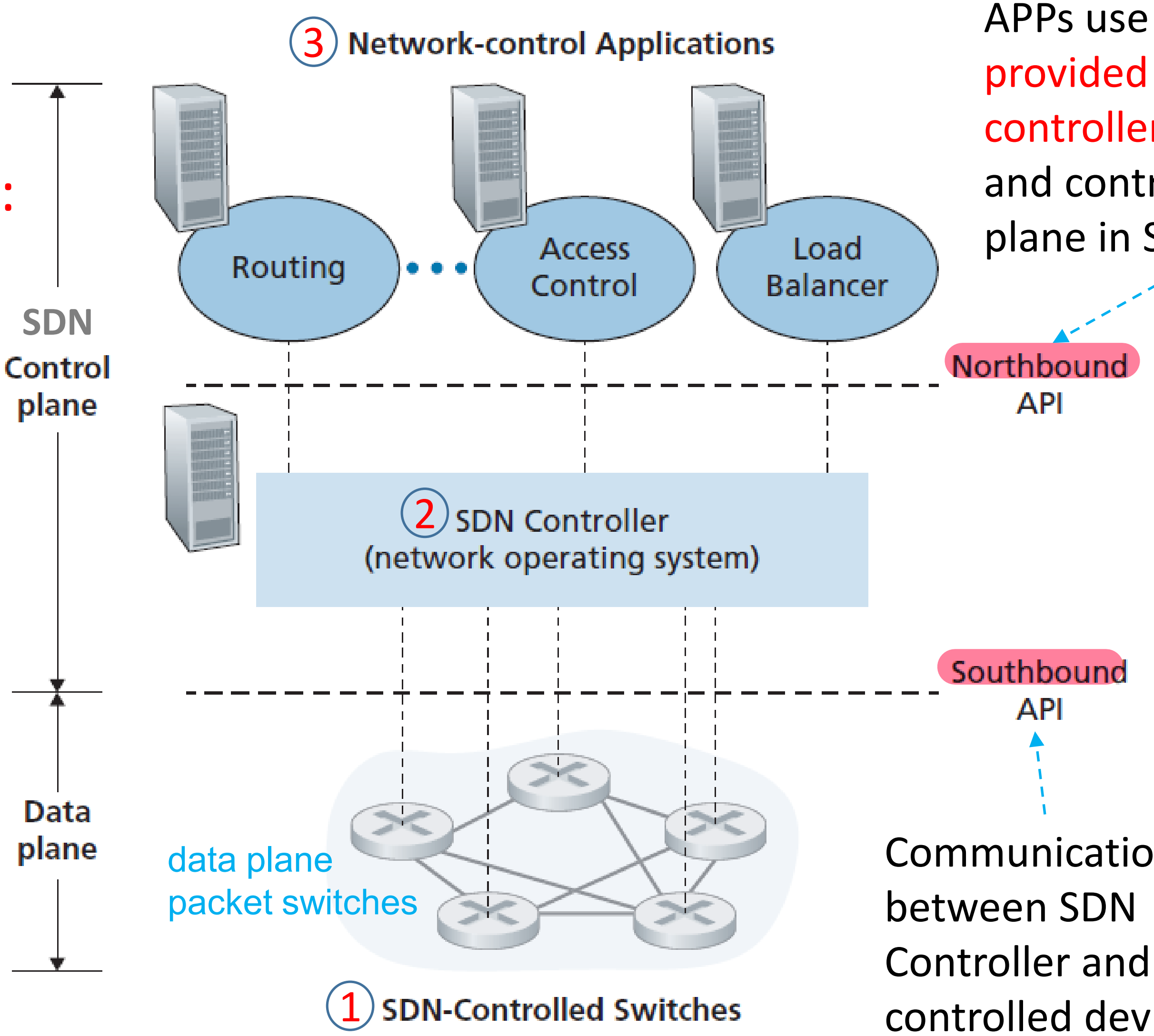
3. Network control functions: a **controller** software external to data-plane switches

4. A programmable network: routing, firewall, access control, …

Figure 5.14

Components of SDN architecture:

1. SDN-controlled switches
2. SDN controller
3. Network-control applications



Network-control APPs use APIs provided by SDN controller to specify and control data plane in Switches

Communication between SDN Controller and controlled devices

Unbundling of network functionality

- SDN represents a significant “unbundling”
- Entities that may each be provided by different vendors:
- Packet switch
- SDN controller
- Network-control APPs

- Traditional networking, a single vendor provides: a switch/router together with its embedded control plane software and protocol

5.5 The SDN Control Plane

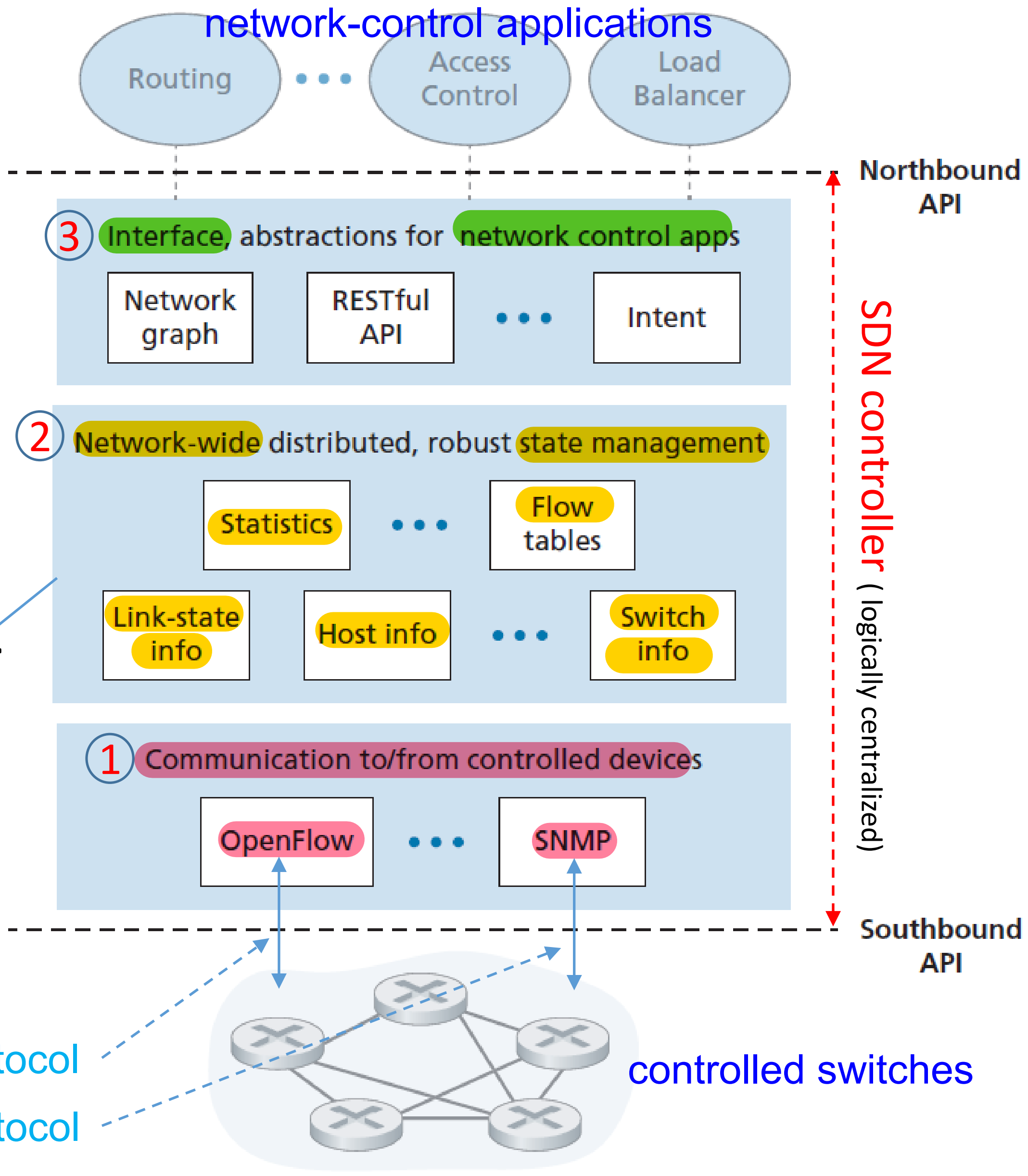
- SDN control plane divides broadly into two components
  - Controller
  - Network-control APPs
- Controller is implemented in 3 layers (Figure 5.15)
- Controller controls devices such as **switches**, **host**, **links**, or others

Figure 5.15 - controller

1. Communication layer: communicating between SDN controller and controlled network devices (packet switches, hosts, …) using appropriate protocols via Southbound APIs
2. Network-wide state-management layer
3. Interface to network-control applications layer via Northbound APIs

Managers of:

- Flow table
- Link-state
- Statistics
- Host information
- Switch Information
- …



5.5.1 SDN Control Plane: SDN Controller

1- Communication layer:

- Protocols are used to transfer **information** between controller and devices (OpenFlow protocol)
- **Information:**
  - Controller to devices: packets, flow table, …
  - Devices to controller: packets, locally-observed events, (**event:** an attached link has gone down, **event:** a device has joined network, **event:** a heartbeat indicating a device is up and operational), …
  - **Events** provide SDN controller with an up-to-date view of network's state
- Controller's "southbound" API: Communication between controller and controlled devices

SDN Controller

2- Network-wide state-management layer:

- **Controller have up to date information** about state of networks' hosts, links, switches, and other SDN-controlled devices
- Controller might maintain a **copy of flow tables**. A switch's flow table contains **counters** whose values might also be profitably used by network-control applications; these values should thus be available to applications
- These pieces of information all constitute examples of network-wide "state" maintained by SDN controller
- A **data base** is used to maintain all states and information

SDN Controller

3- Interface to network-control application layer:

- Controller interacts with network-control applications through its "northbound" API
- This API allows network-control applications to **read/write** network state/info and flow tables within state-management layer
- Applications can register **to be notified** when state-change events occur, so they can take actions in response to network event notifications sent from SDN-controlled devices
- Different types of APIs may be provided; we'll see that popular SDN controllers communicate with their applications using a REST API (Representational State Transfer) request-response interface

SDN controller: logically centralized

- SDN controller is “logically centralized,” (it is viewed by SDN-controlled devices and network-control applications **as a single, monolithic service**)
- In practice, SDN controller and its **data base** is implemented by a distributed set of servers for **fault tolerance**, **high availability**, or for **performance** reasons
  - Semantics of controller’s internal operations (e.g., maintaining logical time ordering of events, consistency, consensus, and more) must be considered
  - Such concerns are common across many different distributed systems
- Open-Daylight and ONOS controllers emphasis on architecting a logically centralized but physically distributed controller platform that provides scalable services and high availability to controlled devices and network-control applications alike

5.5.2 OpenFlow Protocol

- OpenFlow protocol operates between an SDN controller and an SDN-controlled switch or other device implementing Open-Flow API (Section 4.4)
- OpenFlow protocol operates over TCP, default port number: 6653

1- Important **messages from controller** to controlled switches are:

- Configuration: Controller query and **set a switch's configuration** parameters
- Modify-State: Controller add/delete or modify **entries in switch's flow table**, and set switch interface properties
- Read-State: Controller **collects statistics** and counter values from switch's flow table and ports (interfaces)
- Send-Packet: Controller **sends a specific packet out** of a specified port at controlled switch. The message itself contains packet to be sent

OpenFlow Protocol

2- Important **messages from controlled** switches to controller:

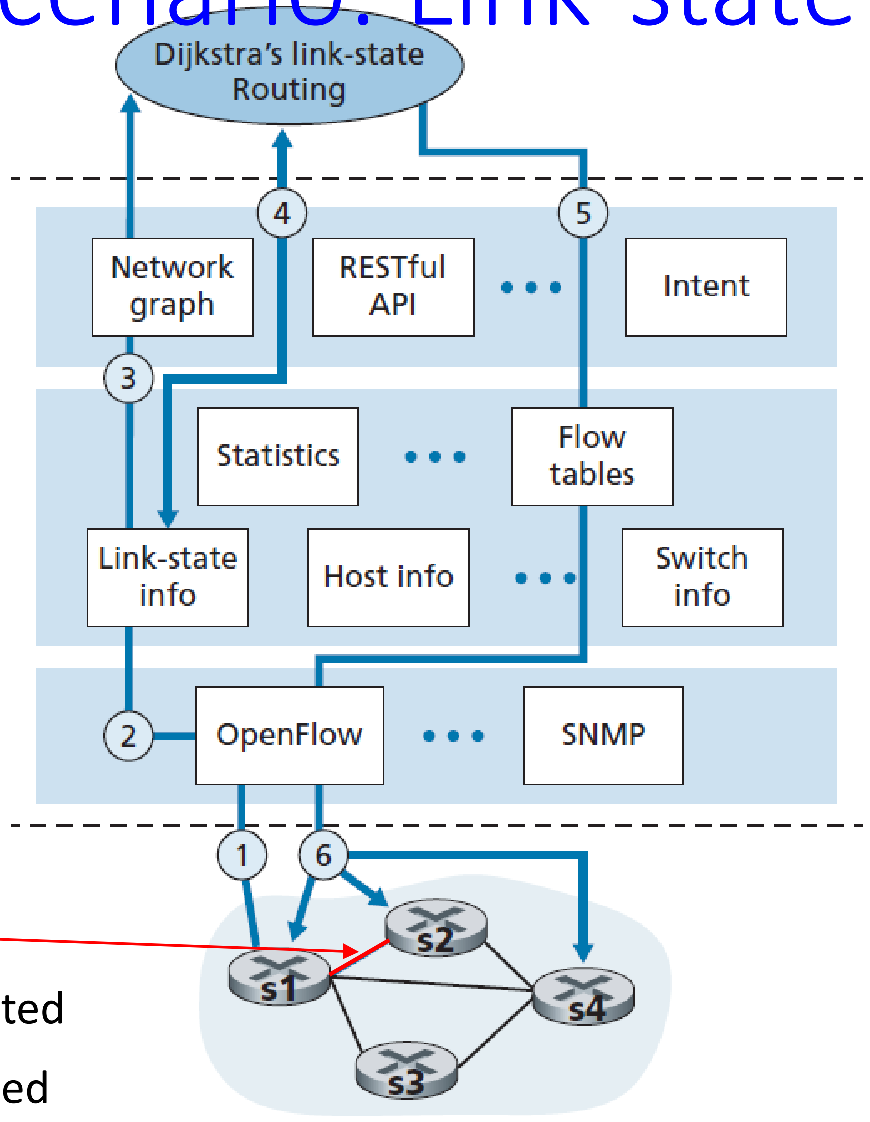
- Flow-Removed: Message informs controller that a flow table entry has been removed, for example by a timeout or as result of a received **modify-state** message
- Port-status: Switch informs controller of a change in port (interface) status
- Packet-in:
  - A **packet arriving** at a switch port and **not matching any flow table** entry is **sent to** controller for additional processing
  - **Matched packets** may be sent to controller, as an action to be taken on a match

5.5.3 Data and Control Plane Interaction: An Example

- Figure 5.16, Dijkstra’s algorithm is used to determine shortest path routes
- SDN scenario has two important differences from earlier per-router-control
  - **Dijkstra’s algorithm is executed as a separate application**, **outside of packet switches** (as a network-control application)
  - Packet switches send link updates **(link state) to SDN** controller and not to each other

Figure 5.16 SDN controller scenario: Link-state change

- Assume link between switch s1 and s2 goes down, so, flow forwarding rules at **s1**, **s2**, and **s4** are affected, but s3's operation is unchanged
- Assume **OpenFlow** is used as communication **layer protocol**, and APP in control plane performs no other function other than link-state routing



Suppose:

- s1 - s2 link goes down
- at **s1**, **s2**, and **s4** are affected
- s3's operation is unchanged

A link goes down

1. s1 notifies controller of link-state change using OpenFlow port-status message
2. Controller receives message, and notifies link-state manager, which updates a link-state database
3. Network-control application has registered to be notified when link state changes. It receives notification of link-state change
4. Application **interacts** with **link-state manager** to get updated link state. It then <u>computes new least-cost paths</u>
5. Application **interacts** with **Flow table manager**, which determines flow tables to be updated
6. **Flow table manager** uses **OpenFlow protocol** to update flow table entries at affected switches, s1, s2, and s4

5.5.4 SDN: Past and Future

- In 2004, separation of network's data and control planes is proposed
- Intense interest in SDN is a relatively recent phenomenon
- Ethane project [2007] pioneered notion of match-plus-action flow tables, a centralized controller that managed flow admission and routing, and forwarding of unmatched packets from switch to controller
- Ethane project quickly evolved into OpenFlow project

SDN: Past and Future

- Numerous research efforts are aimed at developing future SDN architectures and capabilities
- A generalization of SDN known as network functions virtualization (NFV) aims at replacement of sophisticated middleboxes with simple servers, switching, and storage
- A second area of important research seeks to extend SDN concepts from intra-AS setting to inter-AS setting

Contents

5.6 ICMP: The Internet Control Message Protocol [RFC792]

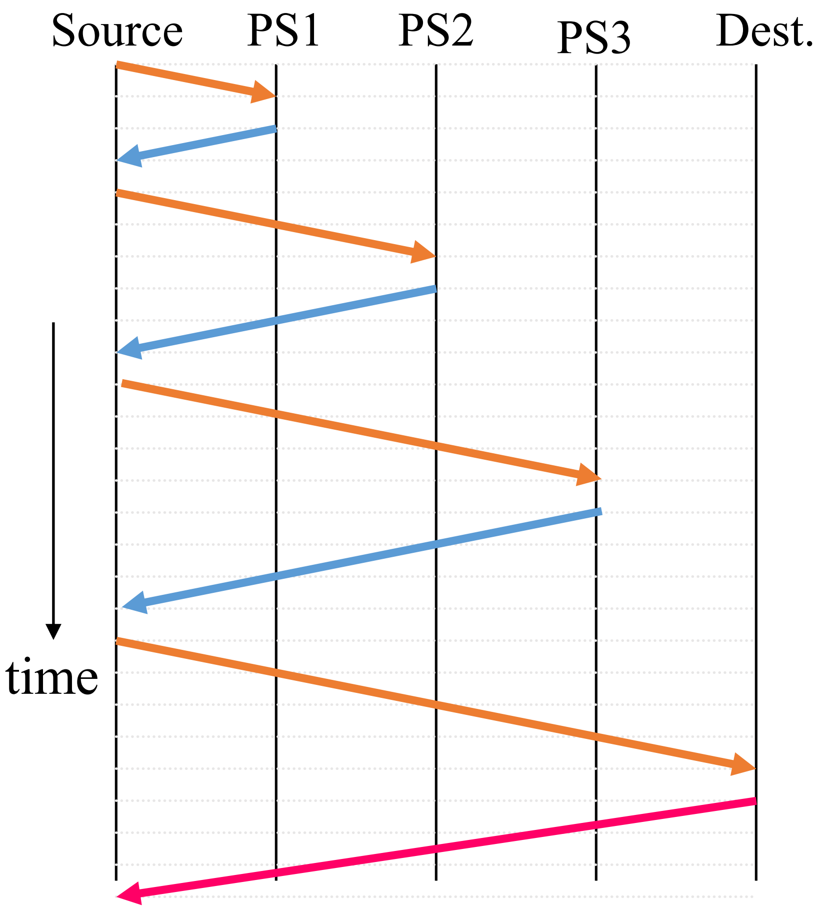
- ICMP is used by **hosts** and **routers** to communicate network-layer information to each other:
  - error reporting
  - echo request/echo reply (used by ping)
- ICMP messages: a type and a code field, and entire IPv4 header, plus first 8 bytes of IP datagram that caused ICMP message to be generated in first place (sender can determine datagram that caused error)
- **ICMP messages** are carried inside **IP datagrams** (upper-layer protocol =1)

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control-not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Figure 5.19 Selected ICMP message types

Traceroute and ICMP

- Source sends sets of UDP segments to destination
  - 1^{st} set has TTL =1, 2^{nd} set has TTL=2, etc.
- datagram in n_{th} set arrives to n_{th} router:
  - router discards datagram and sends source ICMP message (TTL expired: type 11, code 0)
  - **ICMP reply message possibly includes name of router & IP address**
- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops



when ICMP message arrives at source, source record RTTs

ICMP

- Example, in an HTTP session, you may have encountered "Destination network unreachable"
- At some point, an **IP router was unable to find a path to destination host**, so, **router created and sent an ICMP message** to your host indicating error
- ICMPv6 has been defined for IPv6 in RFC 4443
- In addition to reorganizing existing ICMP type and code definitions, ICMPv6 also added new types and codes required by new IPv6 functionality. These include "Packet Too Big" type and an "unrecognized IPv6 options" error code

Contents

5.7 Network Management and SNMP, NETCONF/YANG

- A network consists of 100 or 1000 of complex, interacting pieces of hardware and software:
  - links, switches, routers, hosts, other devices, many protocols
- Network administrator: keep network "up and running"
- Network management tools and approaches: help network administrator
  - **Monitor**
  - **Manage**
  - **Control network**

What is network management?

Network management hardware, software, and human elements be:

- Deployed
- Integrated
- coordinated

5.7.1 The Network Management Framework

Managing server: managing applications, typically with network managers in loop

Managed device: equipment (including its software) with **manageable, configurable hardware, software components**

Network management agent: Software, communicates with managing server, taking local actions at managed device under command and control of managing server

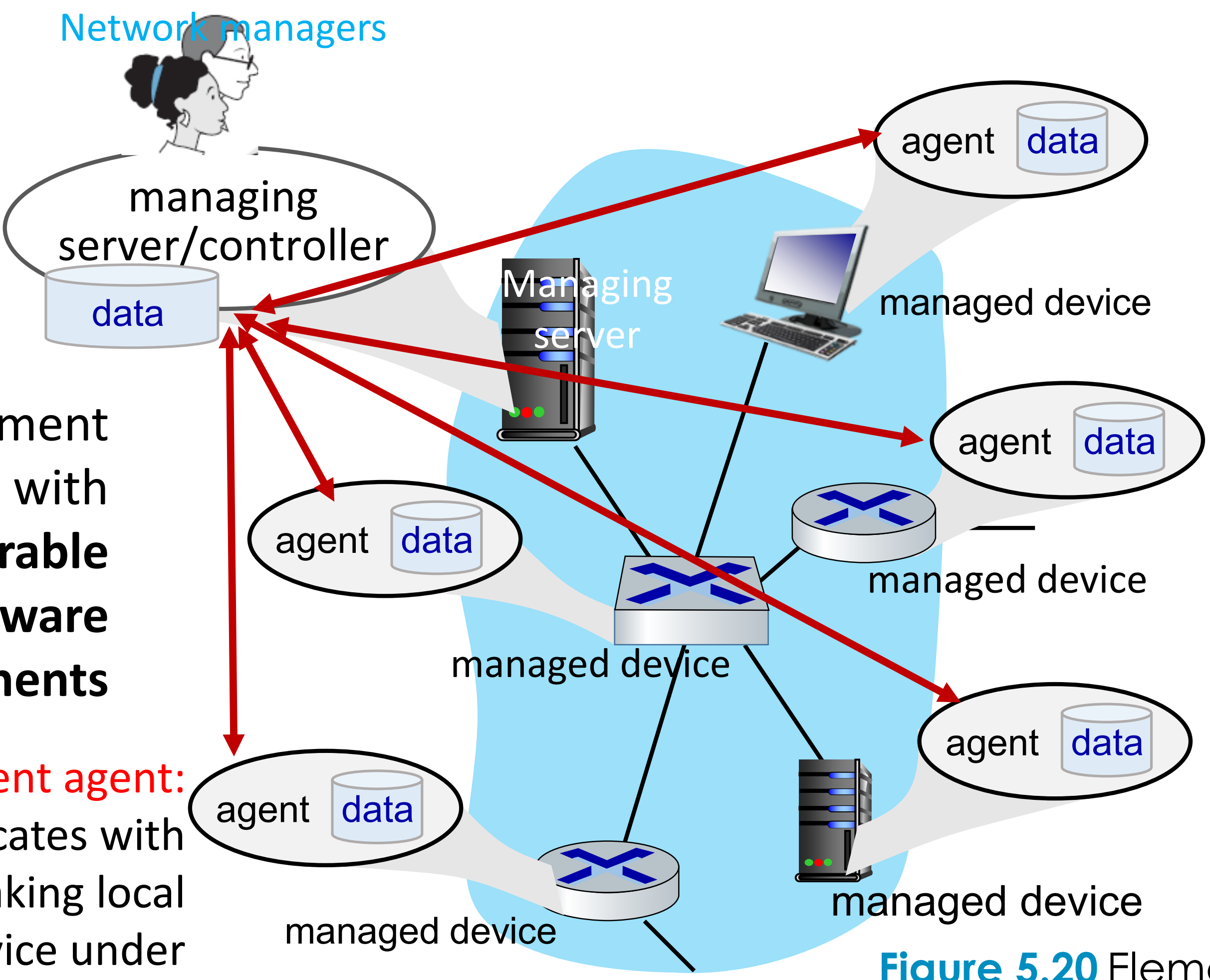


Figure 5.20 Elements of network management

Data: device **state**, configuration data, operational data, device statistics

Network management protocol: used by managing server and agent for: **query/reply**, **notify events**, **configure**, **manage device, data**

Practical ways of doing network management

1. Command Line Interface (CLI)

- Direct commands to device
- Typed either directly on a managed device's **console**, or
- Over a **Telnet** or **secure shell** (SSH) connection, possibly via scripting

- CLI commands is **prone to errors**
- CLI is difficult to **automate** or **efficiently sc**ale for large networks

Practical ways of doing network management

2. Management software based on **SNMP/MIB**

- Management software sends query or set data contained in a device’s Management Information Base (MIB) objects using **Simple Network Management Protocol (SNMP)**

- Information in MIB are either
  - **Device-** and **vendor-specific**, or
  - **Device-independent** (e.g., number of IP datagrams discarded at a router due to errors in an IP datagram header, or number of UDP segments received at a host) are
- A network operator uses Management software to **query** and **monitor operational** state and device **statistics**, and then **use CLI** to actively control/configure device
- Both CLI and SNMP/MIB manage devices individually

The Management Information Base (MIB)

- Examples of MIB objects:
  - A counter, such as number of IP datagrams discarded at a router due to errors in an IP datagram header
  - Number of carrier sense errors in an Ethernet interface card
  - Version of software running on a DNS server
  - Status information such as whether a particular device is functioning correctly
  - Protocol-specific information such as a routing path to a destination
  - …
- Related MIB objects are gathered into MIB modules
- Over 400 MIB modules (IETF), many more device- and vendor-specific MIBs
- MIB objects: specified in a data description language known as SMI (Structure of Management Information) [RFC 2578; RFC 2579; RFC 2580]

Example: MIB object, ipSystem-StatsInDelivers

- Counter 32 data type:
  - Counter32 is one of basic data types defined in SMI
  - A 32-bit read-only counter, keeps track of number of IP datagrams received at managed device and were successfully delivered to an upper-layer protocol

```
ipSystemStatsInDelivers OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
       "The total number of datagrams successfully de-
       livered to IPuser-protocols (including ICMP).

       When tracking interface statistics, the coun-
       ter of the interface to which these datagrams
       were addressed is incremented. This interface
       might not be the same as the input interface
       for some of the datagrams.

      Discontinuities in the value of this counter can
      occur at re-initialization of the management
      system, and at other times as indicated by the
      value of ipSystemStatsDiscontinuityTime."
 ::= { ipSystemStatsEntry 18 }
```

Practical ways of doing network management

3. Management software based on **YANG/NETCONF**

- It takes a more abstract, network- wide, and holistic view toward network management, with a much stronger emphasis on configuration management, including specifying correctness constraints and providing management operations over multiple controlled devices
  - **YANG** [RFC 6020] is a data modeling language used to model configuration and operational data (YANG: Yet Another Next Generation)
  - **NETCONF** protocol [RFC 6241] is used to communicate YANG-compatible actions and data to/from/among remote devices (NETCONF: Network Configuration)

5.7.2 The Simple Network Management Protocol (SNMP)

- Simple Network Management Protocol version 3 (SNMPv3) [RFC 3410], **application-layer protocol**, <u>control and information messages</u> between a managing server and an agent
- SNMP's modes:
  - Request-response mode: SNMP managing server sends a request to an SNMP agent, who receives request, performs some action, and sends a reply to request
    - Requests: **query** (retrieve) or **modify** (set) MIB object values
  - Trap mode: Agent sends an **trap message** to a managing server Trap messages are used to **notify** a managing server of an **exceptional situation** (e.g., a link interface going up or down) that has resulted in changes to MIB object values

SNMPv3 –PDU format (message format)

- Format of PDU is shown in Figure 5.21.

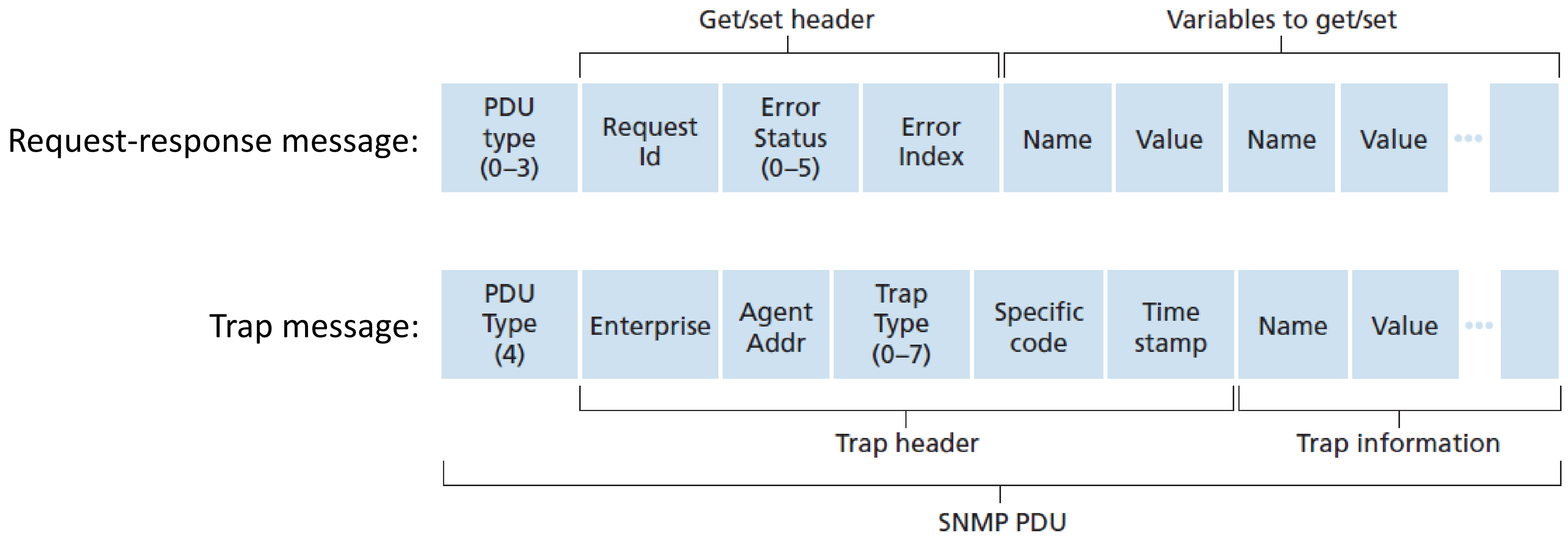


Figure 5.21 SNMP PDU format

Table 5.2 SNMPv3 Protocol Data Unit (message) types

Request-response mode

Type	PDU Type	Sender-receiver	Description
0	GetRequest	manager-to-agent	get value of one or more MIB object instances
0	GetNextRequest	manager-to-agent	get value of next MIB object instance in list or table
0	GetBulkRequest	manager-to-agent	get values in large block of data, for example, values in a large table
1	**InformRequest**	manager-to-manager	inform remote managing entity of MIB values remote to its access
2	SetRequest	manager-to-agent	set value of one or more MIB object instances
3	Response	agent-to-manager or manager-to-manager	generated in response to a request (GetRequest, GetNextRequest, GetBulkRequest, InformRequest, SetRequest)
4	Trap	agent-to-manager	inform manager of an exceptional event numbers

SNMPv3 –Protocol Data Units types

- `GetRequest` can request an arbitrary set of MIB values
- Multiple `GetNextRequests` can be used to sequence through a list or table of MIB objects
- `GetBulkRequest` allows a large block of data to be returned (a table), avoiding multiple `GetRequest` or `GetNextRequest` messages

- In all three cases, agent responds with a Response PDU containing **object identifiers** and their associated **values**

SNMPv3 –Protocol Data Units types

- `SetRequest` is used by a managing server to set value of one or more MIB objects in a managed device
  - Agent replies with a `Response` with the "noError" error status to confirm that value has indeed been set
- `InformRequest` is used by a **managing server** to notify another **managing server** of MIB information that is remote to receiving server
- `Response` is typically sent from a managed device to managing server in response to a request message from that server, returning requested information
- `SNMPv3 trap` is not generated in response to a request but in response to an event for which **managing server requires notification**
  - A received trap request has no required response from a managing server

More on SNMPv3

- SNMP uses unreliable UDP
- Request ID in PDU (Figure 5.21) is used by managing server to number its request; agent's response uses request ID, so, managing server detects lost requests or replies
- Time-out Timer in server: It is up to managing server to decide whether to retransmit a request if no response is received after a given amount of time
  - Managing server "needs to act responsibly in respect to **frequency** and duration of retransmissions"
- "SNMPv3 can be thought of as SNMPv2 with additional security and administration capabilities" [RFC 3410]
- Role of security in SNMPv3 is important for monitoring and control

5.7.3 The Network Configuration Protocol (NETCONF) [RFC 6241] and YANG

- NETCONF protocol: between managing server and managed network devices
- Providing messaging to:
    1. Retrieve, set, and modify configuration data at managed devices
    2. Query operational data and statistics at managed devices
    3. Subscribe to notifications generated by managed devices
- Managing server controls a managed device by sending it configurations, **structured XML document**, and activating a configuration at managed device
- NETCONF: is working over a secure, connection-oriented session such as TLS over TCP

Figure 5.22

1. Managing server establishes a secure connection to managed device (server initiates connection!)
2. Managing server and managed device exchange <hello> messages, declaring their "capabilities"
3. Interactions take form of a remote procedure call, <rpc> and <rpc-response> messages
4. Session is closed with <session-close message>

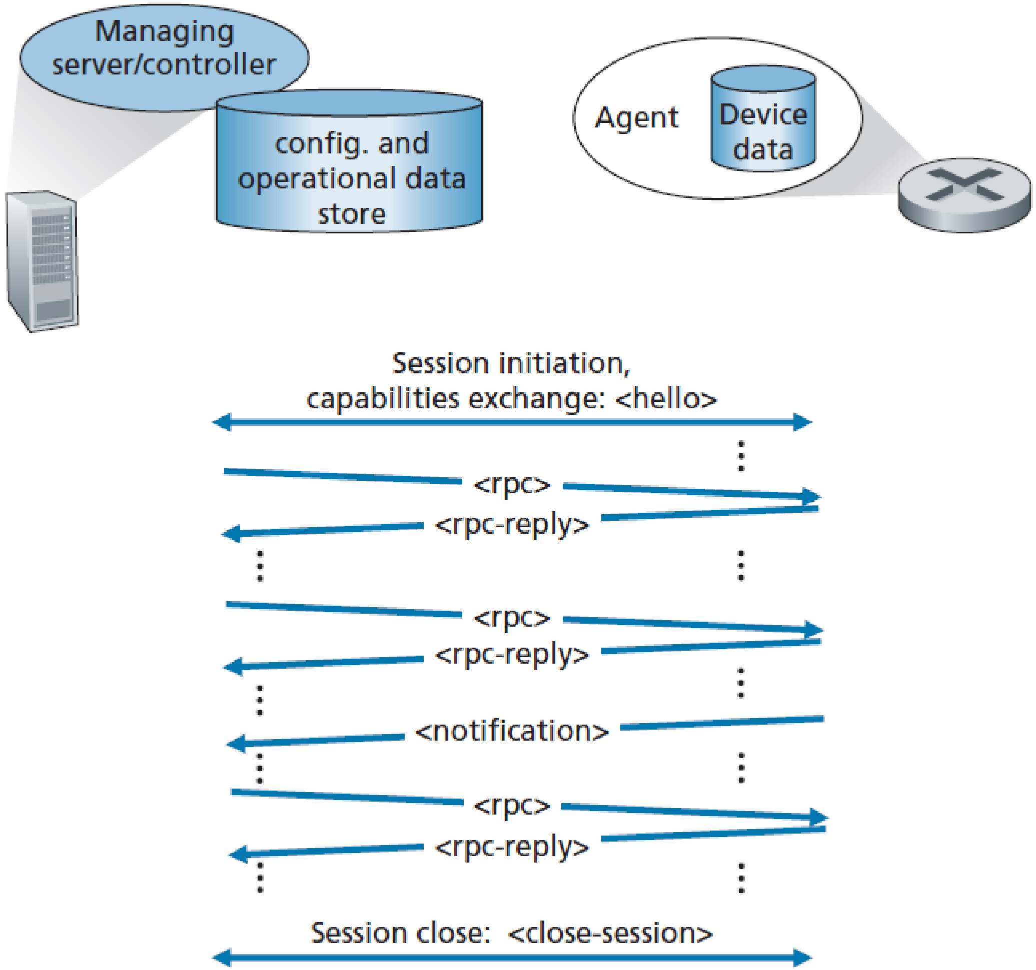


Figure 5.22 NETCONF session between managing server/controller and managed device

NETCONF

For purpose of:

- Device configurations,
- Operational data and statistics
- Subscribe to device notifications

Flowing <rpc> and <rpc-response> messages are used:

- <retrieve>
- <set>
- <query>
- <modify>

For purpose of:

- Device notification

Following messages are proactively sent from managed device to managing server:

- <notification>

Table 5.3 Selected NETCONF protocol operations

NETCONF Operation	Description
<get-config>	Retrieve all or part of a given configuration. A device may have multiple configurations. There is always a running/ configuration that describes devices current (running) configuration
<get>	Retrieve all or part of both configuration state and operational state data
<edit-config>	Change all or part of a specified configuration at managed device. If running/configuration is specified, then device's current (running) configuration will be changed. If managed device was able to satisfy request, an <rpc-reply> is sent containing an <ok> element; otherwise <rpcerror> response is returned. On error, device's configuration state can be rolled-back to its previous state
<lock>, <unlock>	<lock> (<unlock>) operation allows managing server to lock (unlock) entire configuration data store system of a managed device. Locks are intended to be short-lived and allow a managing server to make a change without fear of interaction with other NETCONF, SNMP, or CLIs commands from other sources
<create-subscription> <notification>	This operation initiates an event notification subscription that will send asynchronous event <notification> for specified events of interest from managed device to managing server, until subscription is terminated

NETCONF operations

- Table 5.3: important NETCONF protocol operations
- Operations: retrieving (<get>) operational state data, event notification
- Using NETCONF operations, we can create a set of sophisticated management transactions that
  - complete atomically and successfully on a set of devices
  - Such multi-device transactions concentrate on configuration of network as a whole rather than individual devices

XML message vs Header-Body message

- This is first time we've seen protocol messages formatted as an XML document (rather than traditional message with header fields and message body)

Example1 – <get> command

- XML document sent from managing server to managed device is a NETCONF <get> command requesting all device configuration and operational data. With this command, server can learn about device's configuration:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <rpc message-id="101"
3.       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
4.     <get/>
5.  </rpc>
```

rpc messages

NETCONF <get> command

(Line numbers here just for pedagogical purposes)

- Few people can completely parse XML directly
- NETCONF command is relatively human-readable

Example1 – <reply> command

- Reply from device contains a matching ID number (101), and all of device's configuration data, starting in line 4, ultimately with a closing </rpc-reply>

```
1.   <?xml version="1.0" encoding="UTF-8"?>
2.   <rpc-reply message-id="101"
3.        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
4.     <!--...all configuration data returned... -->
...
</rpc-reply>
```

Example 2 – <edit-config> command

- XML document sent from managing server to managed device sets Maximum Transmission Unit (MTU) of an interface named “Ethernet0/0” to 1500 bytes
- NETCONF <edit-config> command, spanning lines 04–15

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101"
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04   <edit-config>
05     <target>
06       <running/>
07     </target>
08     <config>
09       <top xmlns="http://example.com/schema/
         1.2/config">
10         <interface>
11           <name>Ethernet0/0</name>
12           <mtu>1500</mtu>
13         </interface>
14       </top>
15     </config>
16   </edit-config>
17 </rpc>
```

running device configuration will be changed

MTU size=1500 Byte to be set of Ethernet0/0 interface

Example2

- Once managed device has changed interface's MTU size in configuration, it responds back to managing server with an OK reply (line 04 below), again within an XML document:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc-reply message-id="101"
03    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04       <ok/>
05       </rpc-reply>
```

YANG (Yet Another Next Generation)

- YANG: data modeling language used to precisely specify structure, syntax, and semantics of network management **data used by NETCONF**
- (in much same way that SMI is used to specify MIBs in SNMP)
- All YANG definitions are contained in modules
- XML document describing a device and its capabilities can be generated from a YANG module
- YANG features a small set of built-in data types (as in case of SMI) and also allows data modelers to express constraints that must be satisfied by a valid NETCONF configuration
- YANG is also used to specify NETCONF notifications

Contents

5.8 Summary

- Network layer: network layer's data plane in Chapter 4 and network layer's control plane in Chapter 5
- Control plane
  - is a network-wide logic
  - controls
    - how a datagram is forwarded among routers along an end-to-end path from source host to destination host
    - how network-layer components and services are configured and managed

Summary

- Two broad approaches towards building a control plane:
  - Per-router control
  - Software-defined networking (OpenFlow)

- Two fundamental routing algorithms for computing least cost paths in a graph:
  - link-state routing (OSPF)
  - distance-vector routing (BGP)

  Both algorithms find application in both per-router control and in SDN control

Summary

- Managing an IP network:
  - ICMP (Internet Control Message Protocol)
  - Network management using SNMP and NETCONF/YANG

Appendix

SETTING OSPF LINK WEIGHTS

- Link weights reflect cost of using a link and routing algorithm serves to minimize overall cost
- In practice, network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals
- For example, suppose a network operator has an estimate of traffic flow entering network at each ingress point and destined for each egress point
- Operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes maximum utilization over all of network’s links
- But with a routing algorithm such as OSPF, operator’s main “knobs” for tuning routing of flows through network are link weights
- Thus, in order to achieve goal of minimizing maximum link utilization, operator must find set of link weights that achieves this goal
- This is a reversal of cause and effect relationship, desired routing of flows is known, and OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

- Answer to this question gets at heart of differences between goals of routing within an AS and among ASs:
- Policy: Among ASs, policy issues dominate
- It may well be important that traffic originating in a given AS not be able to pass through another specific AS.
- Similarly, a given AS may well want to control what transit traffic it carries between other ASs
- Within an AS, everything is nominally under same administrative control, and thus policy issues play a much less important role in choosing routes within the AS

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

- Scale: Ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing
- Within an AS, scalability is less of a concern
- If a single ISP becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into areas.)

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

- Performance: Because inter-AS routing is so policy oriented, quality (for example, performance) of routes used is often of secondary concern
- Indeed, we saw that among ASs, there is not even notion of cost (other than AS hop count) associated with routes
- Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on level of performance realized on a route

GOOGLE’S SOFTWARE-DEFINED GLOBAL NETWORK

- Google deploys a dedicated wide-area network (called B4) that interconnects its data centers and server clusters
- B4 has a Google-designed SDN control plane built on OpenFlow
- B4 is able to drive WAN links at near 70% utilization over long run (a two to three fold increase over typical link utilizations) and split application flows among multiple paths based on application priority and existing flow demands
- B4 is well-suited for SDN:
  - Google controls all devices from servers in data centers to routers in network core
  - Most **bandwidth-intensive applications** are large-scale data copies between sites that can defer to **higher-priority interactive applications** during times of resource congestion
  - With only a few dozen data centers being connected, centralized control is feasible

GOOGLE'S SOFTWARE-DEFINED GLOBAL NETWORK

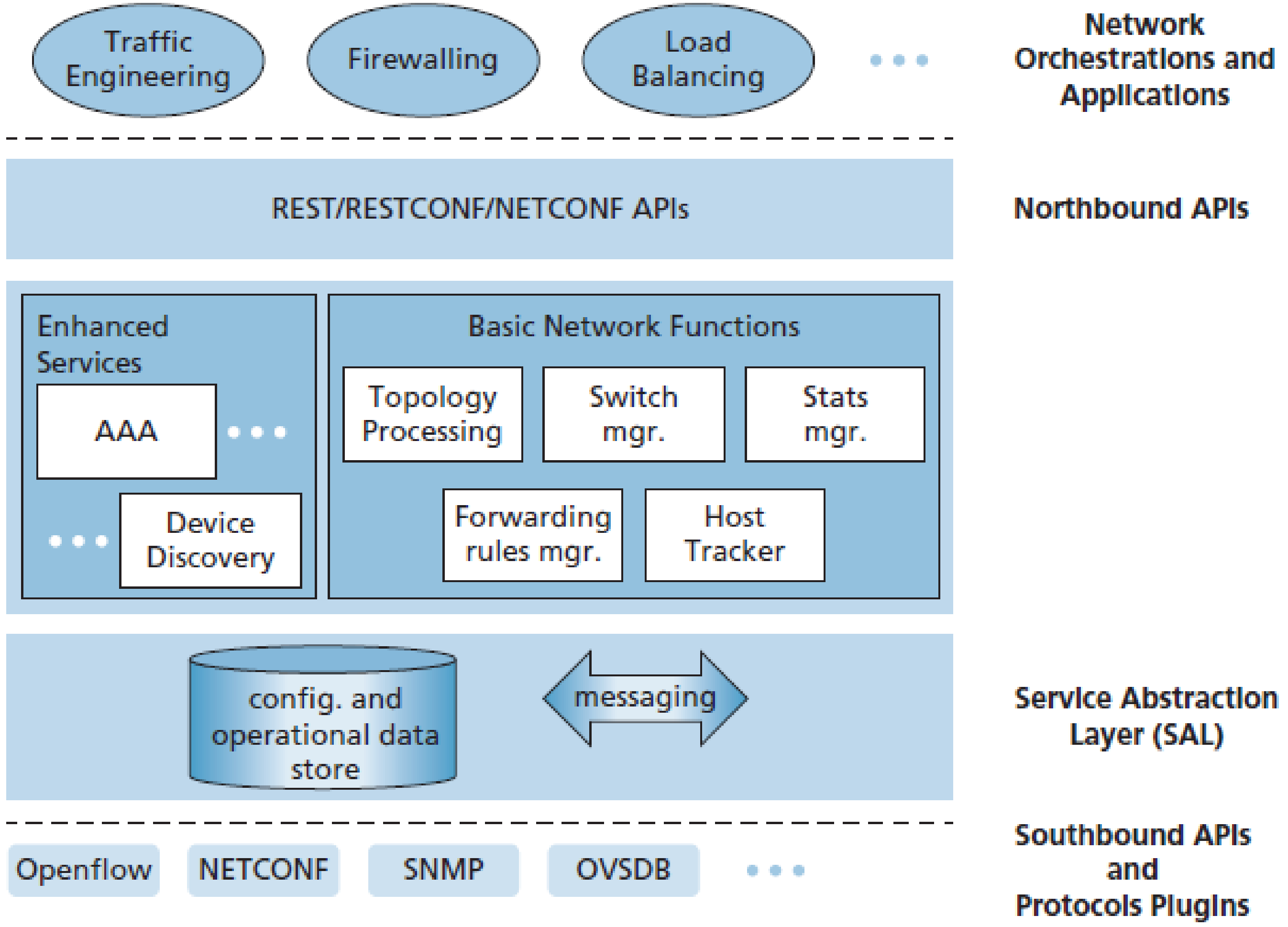
- B4 uses custom-built switches, each implementing a slightly extended version of OpenFlow, with a local Open Flow Agent (OFA)
- OFA connects to an Open Flow Controller (OFC) in network control server (NCS), using a separate "out of band" network, distinct from network that carries data-center traffic between data centers
- In B4, OFC performs state management functions, keeping node and link status in a Network Information Base (NIB)
- Google's implementation of the OFC is based on the ONIX SDN controller
- Two routing protocols, BGP (for routing between data centers) and IS-IS (a close relative of OSPF, for routing within a data center), are implemented
- Paxos is used to execute hot replicas of NCS components to protect against failure

GOOGLE’S SOFTWARE-DEFINED GLOBAL NETWORK

- A traffic engineering network-control application, sitting logically above set of network control servers, interacts with these servers to provide global, network-wide bandwidth provisioning for groups of application flows
- With B4, SDN made an important leap forward into operational networks of a global network provider

SDN CONTROLLER CASE STUDIES: THE OPENDAYLIGHT AND ONOS CONTROLLERS

- **Figure 5.17** A simplified view of OpenDaylight controller



OpenDaylight Controller

- Most recently, OpenDaylight (ODL) controller and ONOS controller have found considerable industry support
- They are both open-source and are being developed in partnership with Linux Foundation
- ODL's Basic Network Functions are at heart of controller, (network-wide state management capabilities)
- Service Abstraction Layer (SAL) allows controller components and applications
  - to invoke each other's services
  - to access configuration and operational data
  - subscribe to events they generate

OpenDaylight Controller

- SAL provides a uniform abstract interface to specific protocols operating between ODL controller and controlled devices
- These protocols include OpenFlow, and Simple Network Management Protocol (SNMP) and Network Configuration (NETCONF) protocol
- Open vSwitch Database Management Protocol (OVSDB) is used to manage data center switching, an important application area for SDN technology

OpenDaylight Controller

- Network Orchestrations and Applications determine how data-plane forwarding and other services, such as firewalling and load balancing, are accomplished in controlled devices
- ODL provides two ways in which **applications can interoperate with native controller services** (and hence devices) and with each other:
  - API-Driven (AD-SAL) approach (Figure 5.17): applications communicate with controller modules using a REST request-response API running over HTTP. Initial releases of ODL controller provided only AD-SAL
  - Model-Driven (MD-SAL) approach. Here, YANG data modeling language [RFC 6020] defines models of device, protocol, and network configuration and operational state data. Devices are then configured and managed by manipulating this data using NETCONF protocol

ONOS Controller

- 3 layers can be identified in the ONOS controller:
- Northbound abstractions and protocols
- Distributed core
- Southbound abstractions and protocols

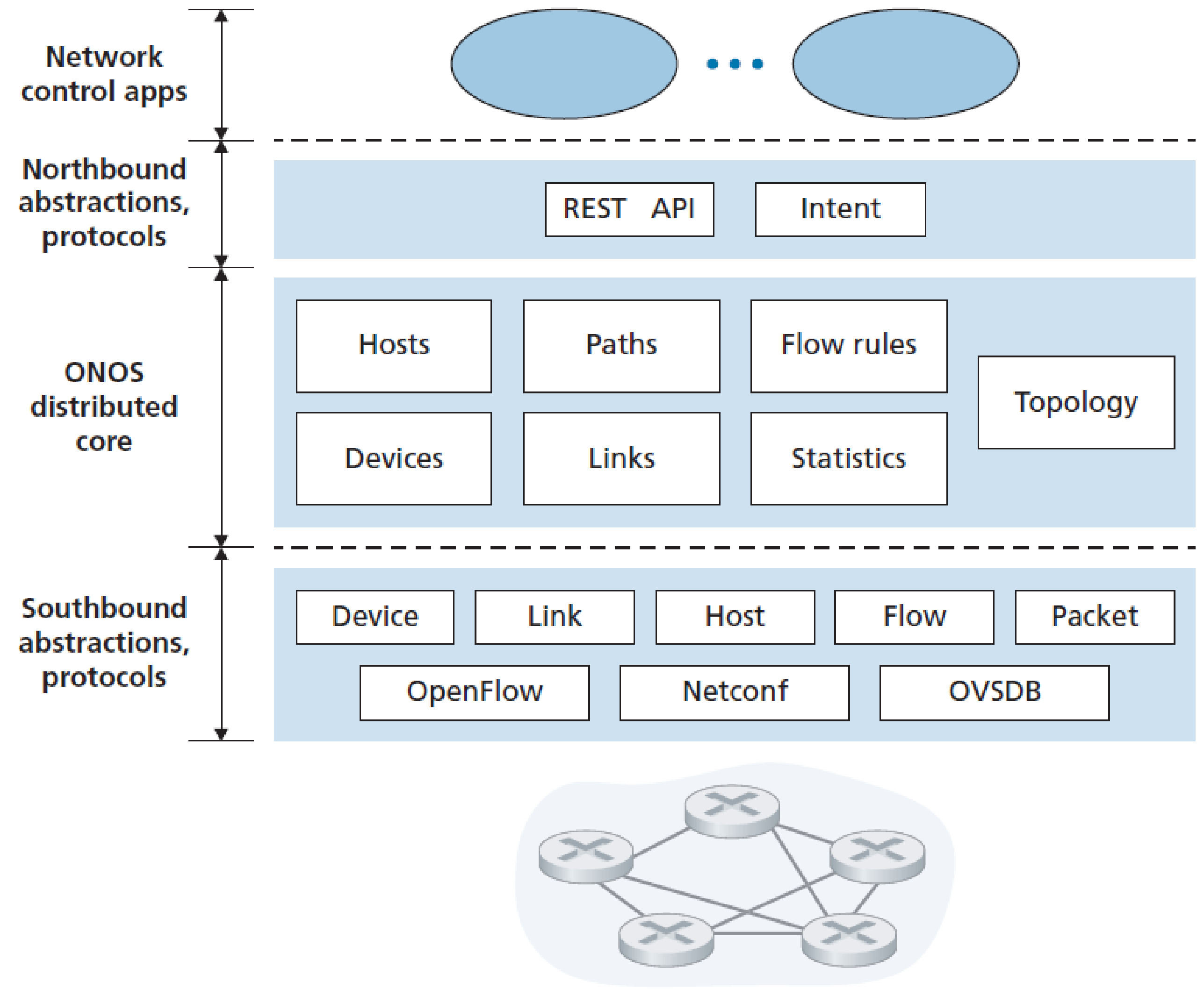


Figure 5.18 ONOS controller architecture