# Chapter 5 Network Layer: Control Plane

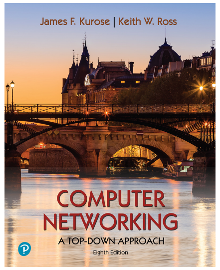
A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR






*Computer Networking: A Top-Down Approach*
8th edition
Jim Kurose, Keith Ross
Pearson, 2020



1

## Network layer control plane: our goals

- understand principles behind network control plane:
  - traditional routing algorithms
  - SDN controllers
  - network management, configuration
- instantiation, implementation in the Internet:
  - OSPF, BGP
  - OpenFlow, ODL and ONOS controllers
  - Internet Control Message Protocol: ICMP
  - SNMP, YANG/NETCONF



2



## 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



3

## 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:

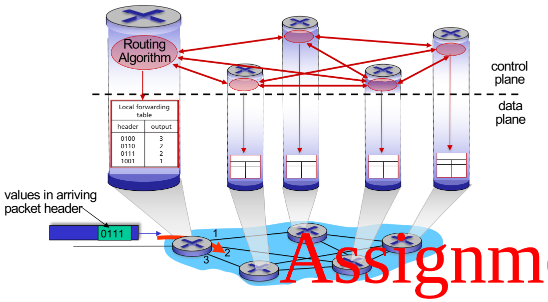
- per-router control (traditional)
- logically centralized control (software defined networking)



4

## Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane

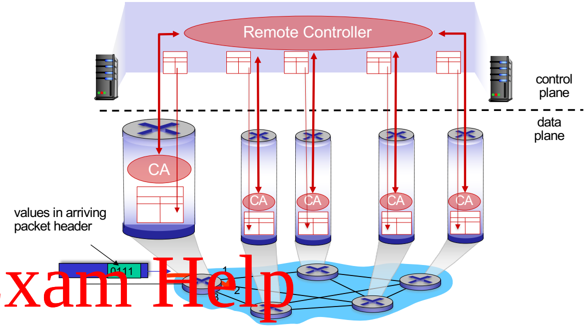







## Software-Defined Networking (SDN) control plane

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

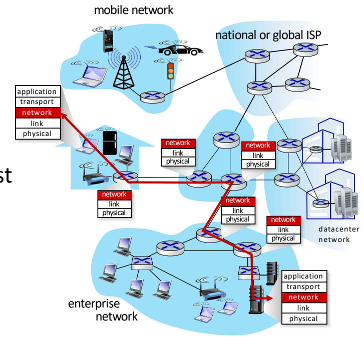




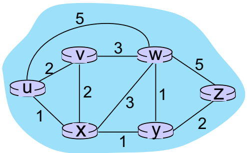
## Routing protocols

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)* }





9

## Routing algorithm classification

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

*How fast do routes change?*

**static:** routes change slowly over time

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

**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?*



10



## 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



11

## 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**

- $c_{x,y}$: direct link cost from node *x* to *y*; $= \infty$ 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



12

## Dijkstra's link-state routing algorithm

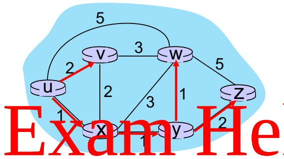
```
Initialization:
  N' = {u}                      /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u          /* u initially knows direct-path-cost only to  direct neighbors   */
       then D(v) = c_u,v        /* but may not be minimum cost!          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





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

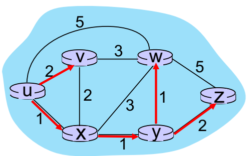
find *a* not in *N'* such that *D(a)* is a minimum
add *a* to *N'*
update *D(b)* for all *b* adjacent to *a* and not in *N'* :
$D(b) = \min ( D(b), D(a) + c_{a,b} )$



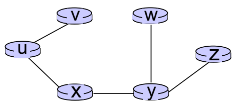




## Dijkstra's algorithm: an example




resulting least-cost-path tree from u:




resulting forwarding table in u:

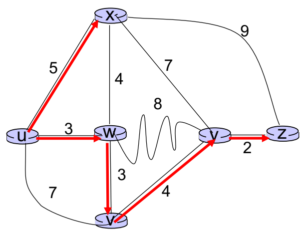
| destination | outgoing link | |
|---|---|---|
| v | (u,v) | route from *u* to *v* directly |
| x | (u,x) | route from u to all other destinations via *x* |
| y | (u,x) | |
| w | (u,x) | |
| x | (u,x) | |





## Dijkstra's algorithm: another 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 | 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 | | | | | |




notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

## 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:

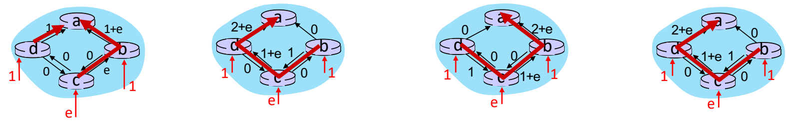
- each router must *broadcast* its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Network Layer: 5-17

17

## 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, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent




Network Layer: 5-18



18

## 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: 5-19

19

## 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:

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

*min* taken over all neighbors *v* of *x*

direct cost of link from *x* to *v*

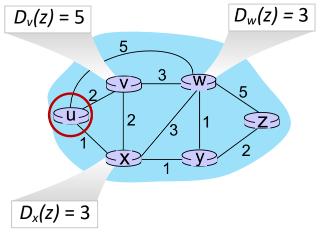
*v*'s estimated least-cost-path cost to *y*

Network Layer: 5-20

20

## Bellman-Ford Example

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




Bellman-Ford equation says:

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

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





21

## 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:

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

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



22



## 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!



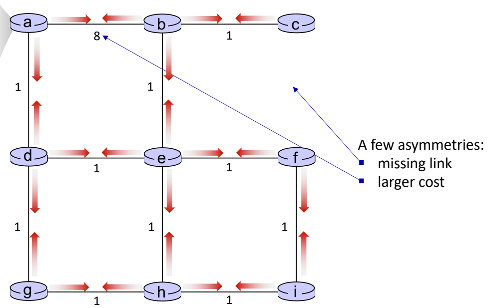
23

## Distance vector: example

t=0

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

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




A few asymmetries:

- missing link
- larger cost

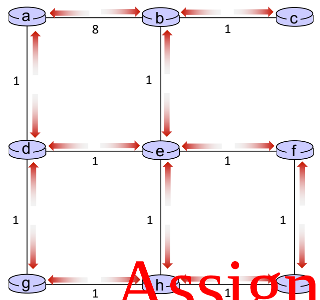


24

## 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




Network Layer: 5-25

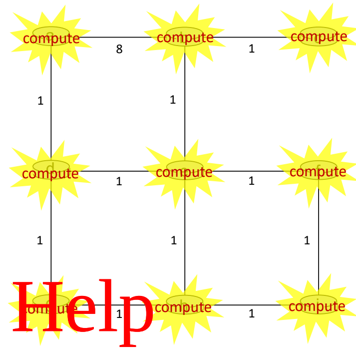
25



## 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




Network Layer: 5-26

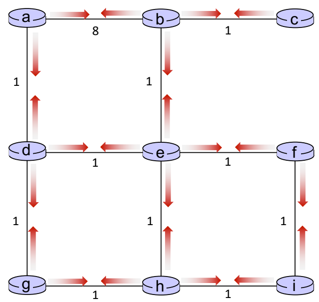
26



## 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



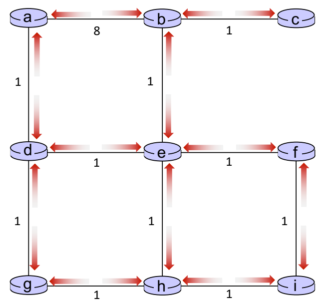

Network Layer: 5-27

27

## 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



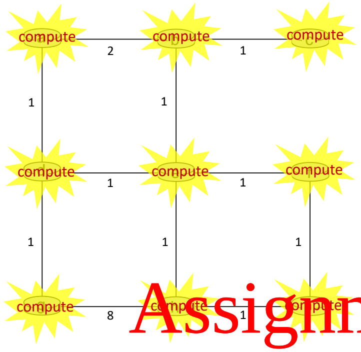

Network Layer: 5-28

28

## 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



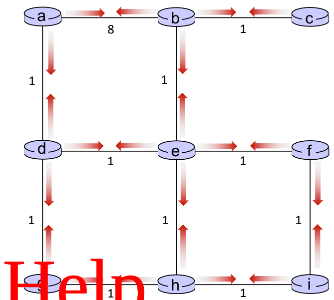

Network Layer: 5-29

29

## 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




Network Layer: 5-30

30



## Distance vector example: iteration

.... and so on

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

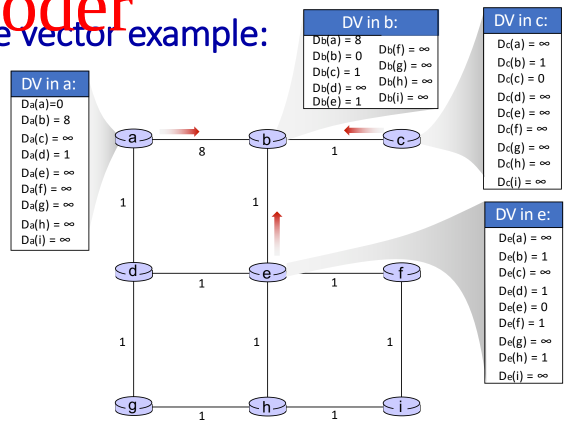
Network Layer: 5-31

31

## Distance vector example:

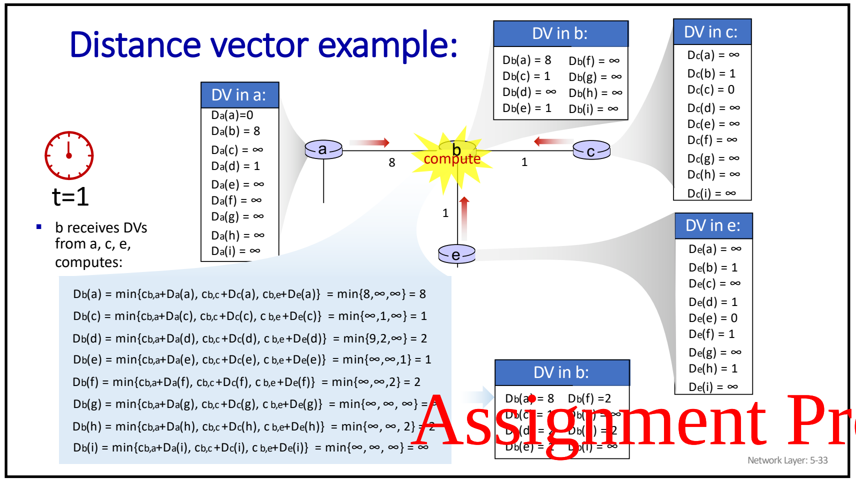
t=1

- b receives DVs from a, c, e



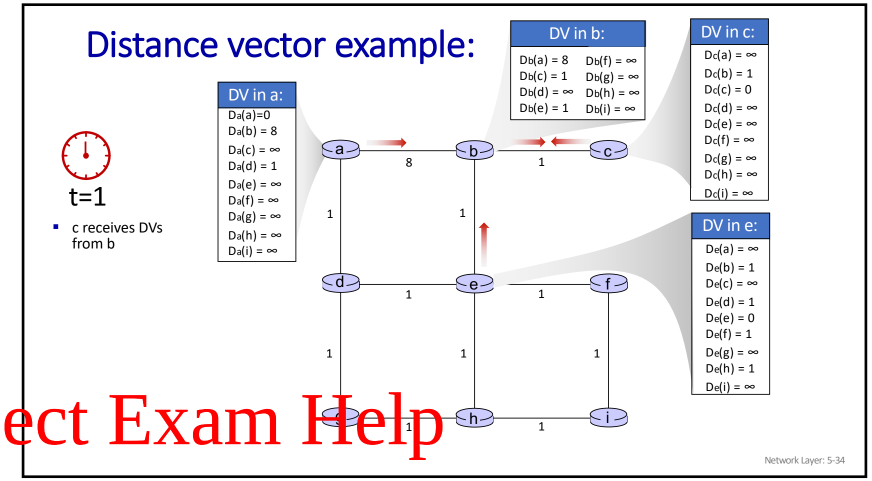

Network Layer: 5-32

32



Distance vector example:
t=1
b receives DVs from a, c, e, computes:
DV in a:
Da(a)=0
Da(b) = 8
Da(c) = ∞
Da(d) = 1
Da(e) = ∞
Da(f) = ∞
Da(g) = ∞
Da(h) = ∞
Da(i) = ∞
DV in b:
Db(a) = 8
Db(c) = 1
Db(d) = ∞
Db(e) = 1
Db(f) = ∞
Db(g) = ∞
Db(h) = ∞
Db(i) = ∞
DV in c:
Dc(a) = ∞
Dc(b) = 1
Dc(c) = 0
Dc(d) = ∞
Dc(e) = ∞
Dc(f) = ∞
Dc(g) = ∞
Dc(h) = ∞
Dc(i) = ∞
DV in e:
De(a) = ∞
De(b) = 1
De(c) = ∞
De(d) = 1
De(e) = 0
De(f) = 1
De(g) = ∞
De(h) = 1
De(i) = ∞
compute
Db(a) = min{cb,a+Da(a), cb,c+Dc(a), cb,e+De(a)} = min{8,∞,∞} = 8
Db(c) = min{cb,a+Da(c), cb,c+Dc(c), c b,e+De(c)} = min{∞,1,∞} = 1
Db(d) = min{cb,a+Da(d), cb,c+Dc(d), c b,e+De(d)} = min{9,2,∞} = 2
Db(e) = min{cb,a+Da(e), cb,c+Dc(e), c b,e+De(e)} = min{∞,∞,1} = 1
Db(f) = min{cb,a+Da(f), cb,c+Dc(f), c b,e+De(f)} = min{∞,∞,2} = 2
Db(g) = min{cb,a+Da(g), cb,c+Dc(g), c b,e+De(g)} = min{∞, ∞, ∞} = ∞
Db(h) = min{cb,a+Da(h), cb,c+Dc(h), c b,e+De(h)} = min{∞, ∞, 2} = 2
Db(i) = min{cb,a+Da(i), cb,c+Dc(i), c b,e+De(i)} = min{∞, ∞, ∞} = ∞
DV in b:
Db(a) = 8
Db(f) =2
Network Layer: 5-33

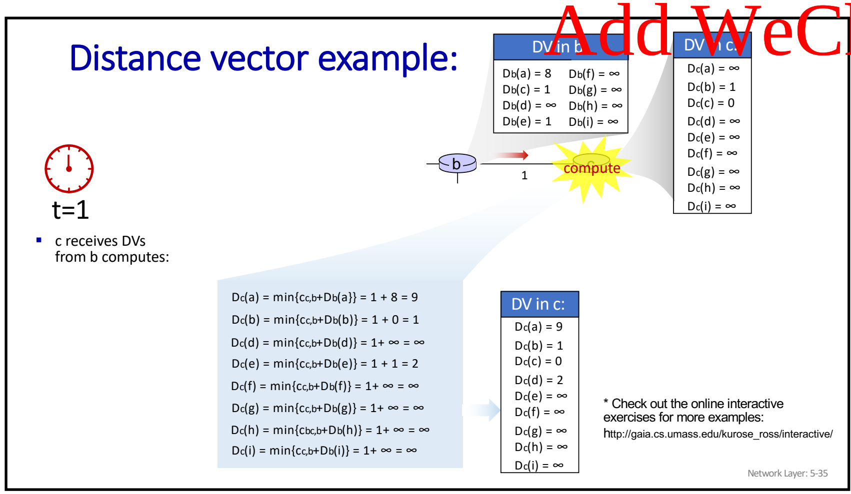

33



Distance vector example:
t=1
c receives DVs from b
DV in a:
Da(a)=0
Da(b) = 8
Da(c) = ∞
Da(d) = 1
Da(e) = ∞
Da(f) = ∞
Da(g) = ∞
Da(h) = ∞
Da(i) = ∞
DV in b:
Db(a) = 8
Db(c) = 1
Db(d) = ∞
Db(e) = 1
Db(f) = ∞
Db(g) = ∞
Db(h) = ∞
Db(i) = ∞
DV in c:
Dc(a) = ∞
Dc(b) = 1
Dc(c) = 0
Dc(d) = ∞
Dc(e) = ∞
Dc(f) = ∞
Dc(g) = ∞
Dc(h) = ∞
Dc(i) = ∞
DV in e:
De(a) = ∞
De(b) = 1
De(c) = ∞
De(d) = 1
De(e) = 0
De(f) = 1
De(g) = ∞
De(h) = 1
De(i) = ∞
Network Layer: 5-34

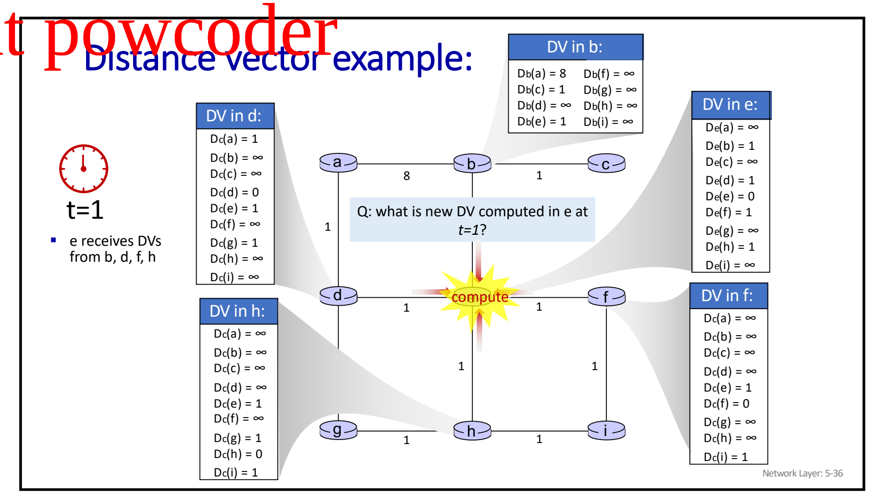

34





Distance vector example:
t=1
c receives DVs from b computes:
DV in b:
Db(a) = 8
Db(c) = 1
Db(d) = ∞
Db(e) = 1
Db(f) = ∞
Db(g) = ∞
Db(h) = ∞
Db(i) = ∞
DV in c:
Dc(a) = ∞
Dc(b) = 1
Dc(c) = 0
Dc(d) = ∞
Dc(e) = ∞
Dc(f) = ∞
Dc(g) = ∞
Dc(h) = ∞
Dc(i) = ∞
compute
Dc(a) = min{cc,b+Db(a}} = 1 + 8 = 9
Dc(b) = min{cc,b+Db(b)} = 1 + 0 = 1
Dc(d) = min{cc,b+Db(d)} = 1+ ∞ = ∞
Dc(e) = min{cc,b+Db(e)} = 1 + 1 = 2
Dc(f) = min{cc,b+Db(f)} = 1+ ∞ = ∞
Dc(g) = min{cc,b+Db(g)} = 1+ ∞ = ∞
Dc(h) = min{cbc,b+Db(h)} = 1+ ∞ = ∞
Dc(i) = min{cc,b+Db(i)} = 1+ ∞ = ∞
DV in c:
Dc(a) = 9
Dc(b) = 1
Dc(c) = 0
Dc(d) = 2
Dc(e) = ∞
Dc(f) = ∞
Dc(g) = ∞
Dc(h) = ∞
Dc(i) = ∞
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
Network Layer: 5-35


35



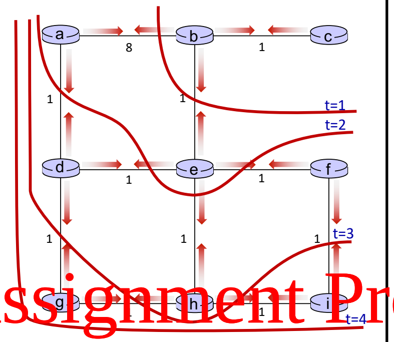
Distance vector example:
t=1
e receives DVs from b, d, f, h
DV in b:
Db(a) = 8
Db(c) = 1
Db(d) = ∞
Db(e) = 1
Db(f) = ∞
Db(g) = ∞
Db(h) = ∞
Db(i) = ∞
DV in d:
Dc(a) = 1
Dc(b) = ∞
Dc(c) = ∞
Dc(d) = 0
Dc(e) = 1
Dc(f) = ∞
Dc(g) = 1
Dc(h) = ∞
Dc(i) = ∞
DV in h:
Dc(a) = ∞
Dc(b) = ∞
Dc(c) = ∞
Dc(d) = ∞
Dc(e) = 1
Dc(f) = ∞
Dc(g) = 1
Dc(h) = 0
Dc(i) = 1
DV in e:
De(a) = ∞
De(b) = 1
De(c) = ∞
De(d) = 1
De(e) = 0
De(f) = 1
De(g) = ∞
De(h) = 1
De(i) = ∞
DV in f:
Dc(a) = ∞
Dc(b) = ∞
Dc(c) = ∞
Dc(d) = ∞
Dc(e) = 1
Dc(f) = 0
Dc(g) = ∞
Dc(h) = ∞
Dc(i) = 1
Q: what is new DV computed in e at t=1?
compute
Network Layer: 5-36


36

## 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

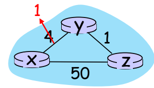



37

## 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




"good news travels fast"

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

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

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

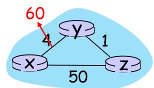


38



## 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!*



39

## 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



40

## 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

Network Layer: 5-41

41

## 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

Network Layer: 5-42

42



## Internet approach to scalable routing

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

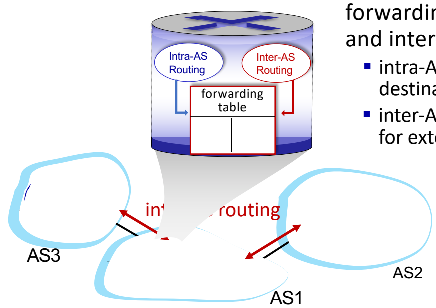
intra-AS (aka "intra-domain"): routing among *within same AS ("network")*
- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS (aka "inter-domain"): routing *among* AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

Network Layer: 5-43

43

## 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

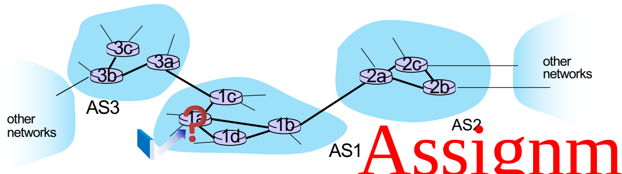
Network Layer: 5-44

44

## 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




Network Layer: 5-45

45



## Inter-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

Network Layer: 5-46

46



## OSPF (Open Shortest Path First) routing

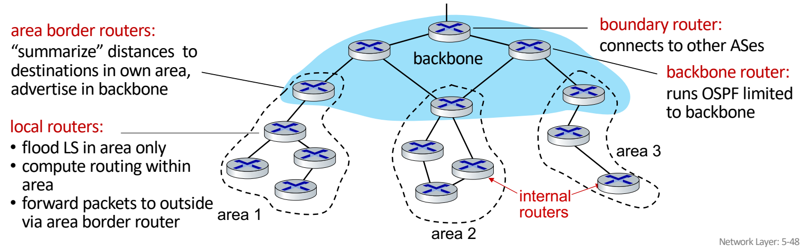
- "open": publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra's algorithm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

Network Layer: 5-47

47

## 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




Network Layer: 5-48

48

## 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





49

## Internet inter-AS routing: BGP

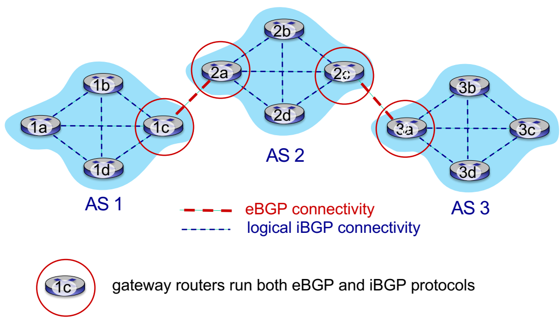
- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *"I am here, here is who I can reach, and how"*
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and *policy*



50



## eBGP, iBGP connections

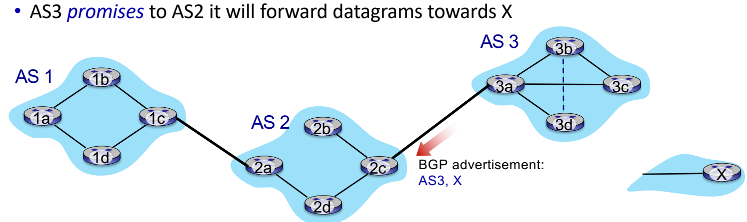





51

## BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X






52

## Path attributes and BGP routes

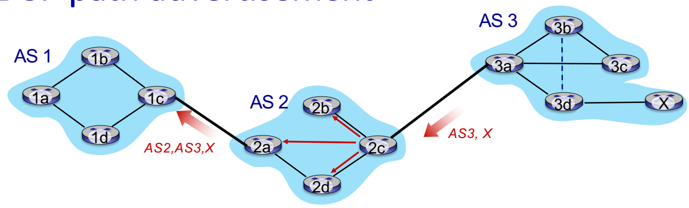
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes





53

## BGP path advertisement



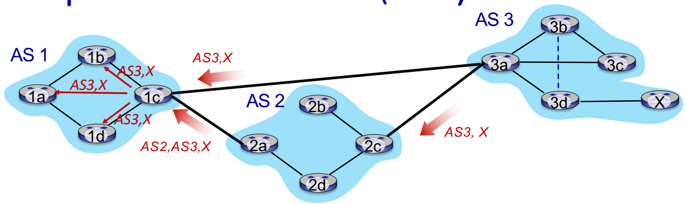

- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c



54



## BGP path advertisement (more)




gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- based on *policy,* AS1 gateway router 1c chooses path *AS3,X* and advertises path within AS1 via iBGP



55

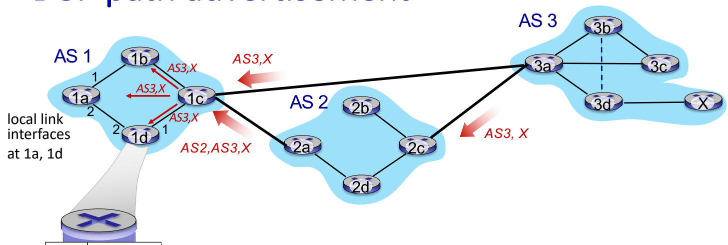
## BGP messages

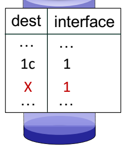
- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - OPEN: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - UPDATE: advertises new path (or withdraws old)
  - KEEPALIVE: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - NOTIFICATION: reports errors in previous msg; also used to close connection



56

## BGP path advertisement



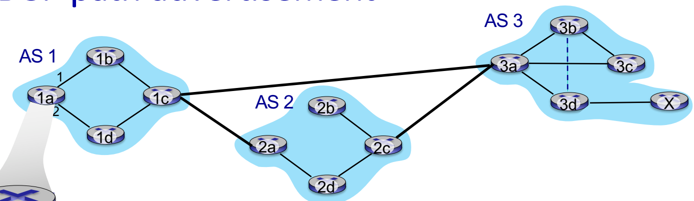

- recall: 1a, 1b, 1d learn via iBGP from 1c: "path to X goes through 1c"
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

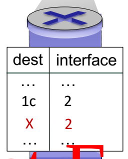




57

## BGP path advertisement




- recall: 1a, 1b, 1d learn via iBGP from 1c: "path to X goes through 1c"
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2



58



## Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

scale:

- hierarchical routing saves table size, reduced update traffic

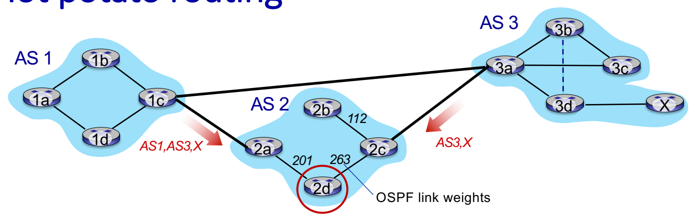
performance:

- intra-AS: can focus on performance
- inter-AS: policy dominates over performance



59

## Hot potato routing

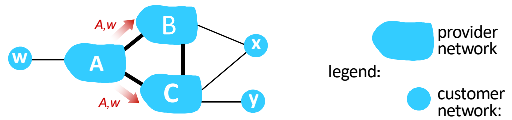



- 2d learns (via iBGP) it can route to X via 2a or 2c
- hot potato routing: choose local gateway that has least *intra-domain* cost (e.g., 2d chooses 2a, even though more AS hops to *X*): don't worry about inter-domain cost!



60

## BGP: achieving policy via advertisements



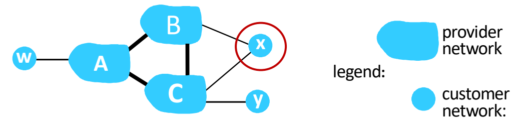

ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)

- A advertises path Aw to B and to C
- B *chooses not to advertise* BAw to C!
  - B gets no "revenue" for routing CBAw, since none of C, A, w are B's customers
  - C does *not* learn about CBAw path
- C will route CAw (not using B) to get to w



61

## BGP: achieving policy via advertisements (more)




ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)

- A,B,C are provider networks
- x,w,y are customer (of provider networks)
- x is dual-homed: attached to two networks
- *policy to enforce:* x does not want to route from B to C via x
  - .. so x will not advertise to B a route to C



62



## BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria



63

## 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



64

## Software defined networking (SDN)

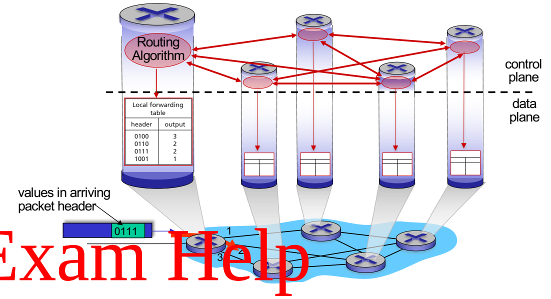
- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane





## Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to computer forwarding tables



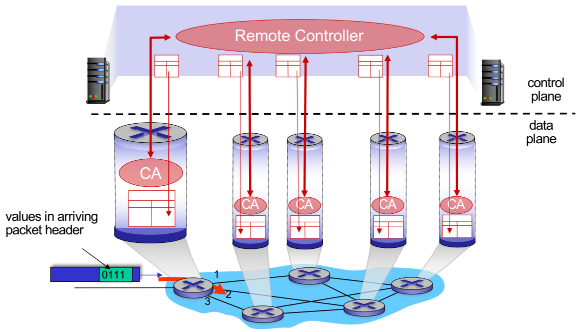







## Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers





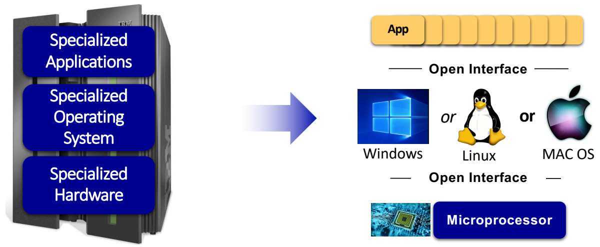



## Software defined networking (SDN)

*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - centralized "programming" easier: compute tables centrally and distribute
  - distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

## SDN analogy: mainframe to PC revolution




Vertically integrated
Closed, proprietary
Slow innovation
Small industry

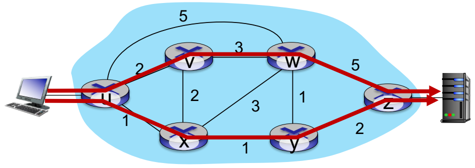
Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

Network Layer: 5-69

69

## Traffic engineering: difficult with traditional routing




Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

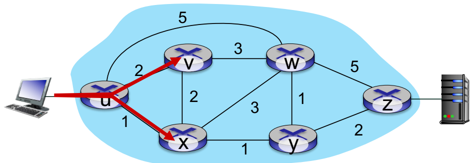
*Link weights are only control “knobs”: not much control!*

Network Layer: 5-70

70



## Traffic engineering: difficult with traditional routing

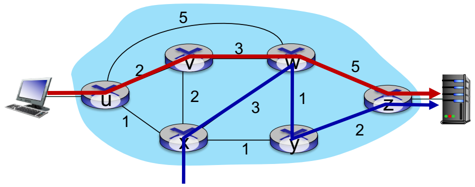



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?
A: can’t do it (or need a new routing algorithm)

Network Layer: 5-71

71

## Traffic engineering: difficult with traditional routing




Q: what if w wants to route blue and red traffic differently from w to z?

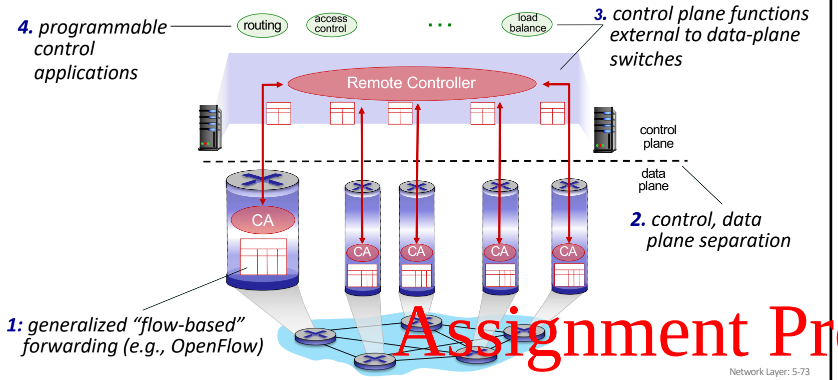
A: can’t do it (with destination-based forwarding, and LS, DV routing)

We learned in Chapter 4 that generalized forwarding and SDN can be used to achieve *any* routing desired

Network Layer: 5-72

72

## Software defined networking (SDN)



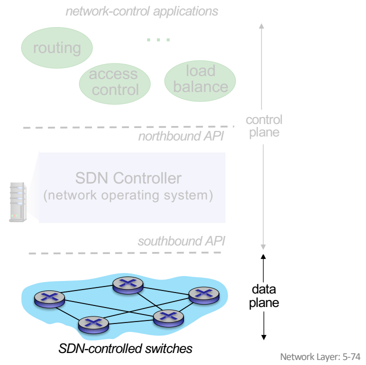



73

## Software defined networking (SDN)

### Data-plane switches:

- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)





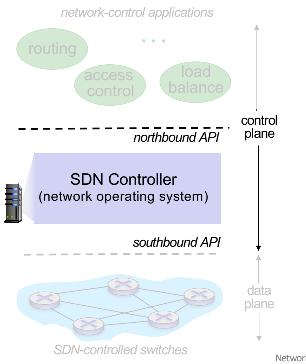

74



## Software defined networking (SDN)

### SDN controller (network OS):

- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



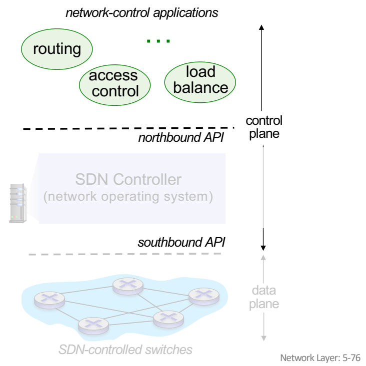



75

## Software defined networking (SDN)

### network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller





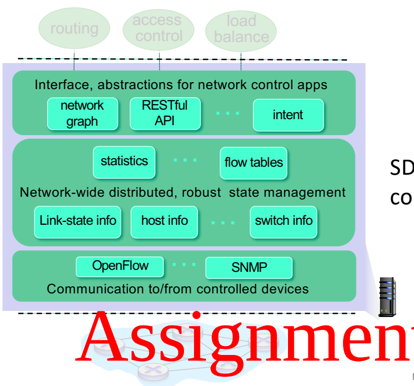

76

## Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management : state of networks links, switches, services: a *distributed database*

*communication*: communicate between SDN controller and controlled switches




SDN controller

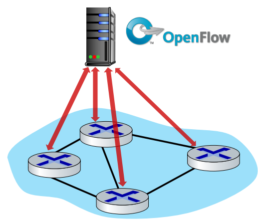
Network Layer: 5-77

77

## OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

OpenFlow Controller




Network Layer: 5-78

78

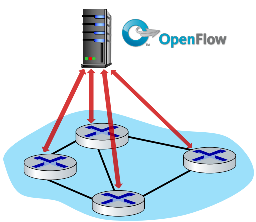


## OpenFlow: controller-to-switch messages

### Key controller-to-switch messages

- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port

OpenFlow Controller




Network Layer: 5-79

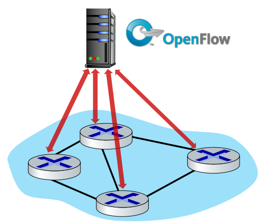
79

## OpenFlow: switch-to-controller messages

### Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.

OpenFlow Controller

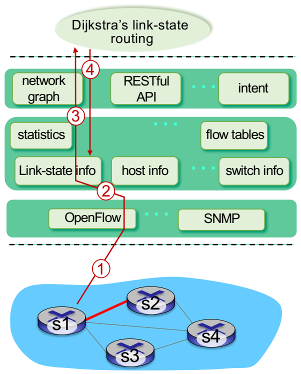



Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

Network Layer: 5-80

80

## SDN: control/data plane interaction example




① S1, experiencing link failure uses OpenFlow port status message to notify controller

② SDN controller receives OpenFlow message, updates link status info

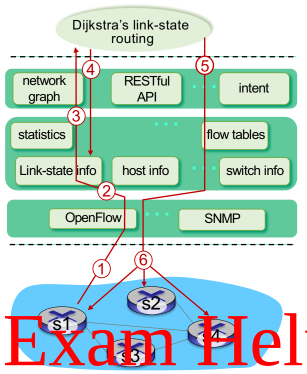
③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes



81

## SDN: control/data plane interaction example




⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

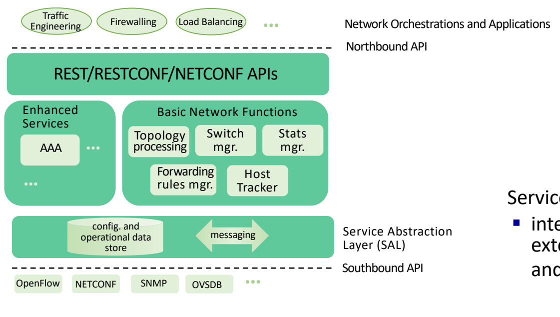
⑥ controller uses OpenFlow to install new tables in switches that need updating



82



## OpenDaylight (ODL) controller

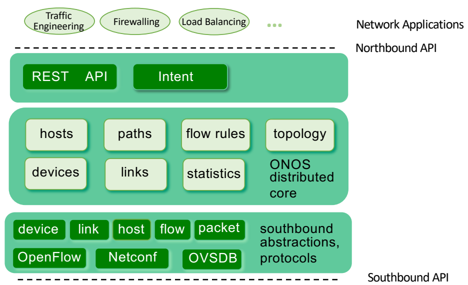



Service Abstraction Layer:
- interconnects internal, external applications and services



83

## ONOS controller




- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling



84

## SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: "baked in" from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling: beyond a single AS
- SDN critical in 5G cellular networks

Network Layer: 5-85

85

## SDN and the future of traditional network protocols

- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels

How will implementation of network functionality (SDN versus protocols) evolve?

Network Layer: 5-86

86



## 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

Network Layer: 5-87

87

## ICMP: internet control message protocol

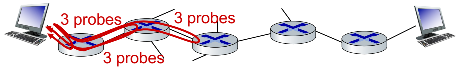
- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP messages carried in IP datagrams
- *ICMP message:* type, code plus first 8 bytes of IP datagram causing error

| 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 |

Network Layer: 4-88

88

## Traceroute and ICMP




- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

stopping criteria:
- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops

Network Layer: 4-89

89



## 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

Network Layer: 5-90

90



## What is network management?

- autonomous systems (aka "network"): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, configuration, control:
  - jet airplane, nuclear power plant, others?

"Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

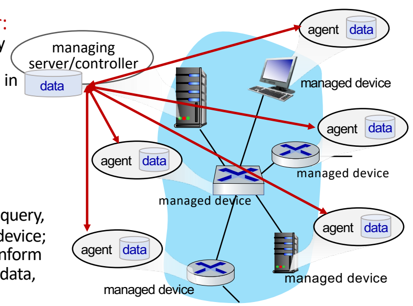
Network Layer: 5-91

91

## Components of network management

Managing server: application, typically with network managers (humans) in the loop

Network management protocol: used by managing server to query, configure, manage device; used by devices to inform managing server of data, events.




Managed device: equipment with manageable, configurable hardware, software components

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

Network Layer: 5-92

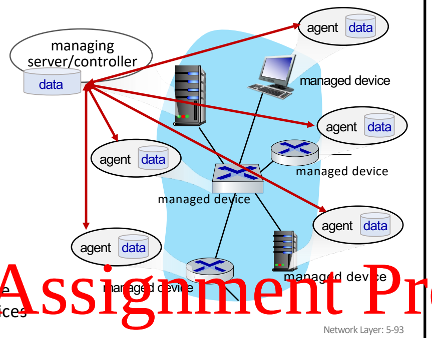
92

## Network operator approaches to management

CLI (Command Line Interface)
- operator issues (types, scripts) direct to individual devices (e.g., vis ssh)

SNMP/MIB
- operator queries/sets devices data (MIB) using Simple Network Management Protocol (SNMP)

NETCONF/YANG
- more abstract, network-wide, holistic
- emphasis on multi-device configuration management.
- YANG: data modeling language
- NETCONF: communicate YANG-compatible actions/data to/from/among remote devices



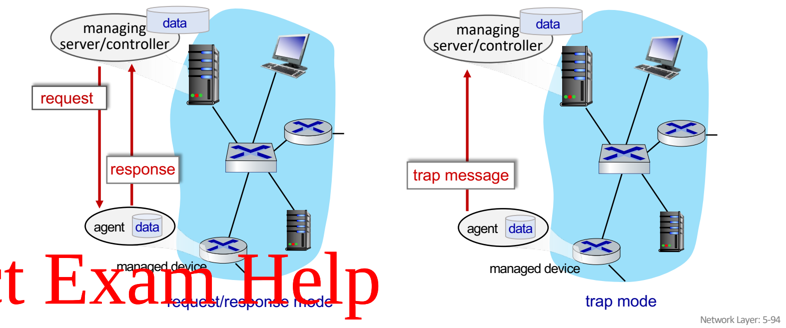

Network Layer: 5-93

93



## SNMP protocol

Two ways to convey MIB info, commands:




Network Layer: 5-94

94



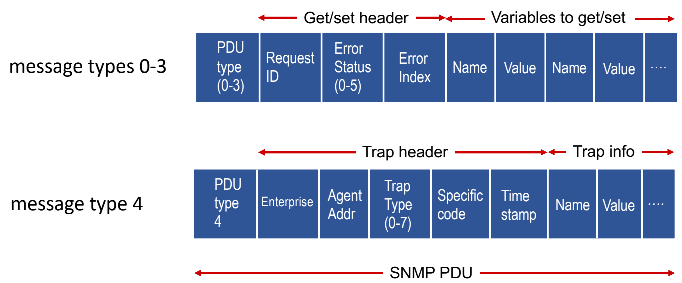
## SNMP protocol: message types

| Message type | Function |
|---|---|
| GetRequest<br>GetNextRequest<br>GetBulkRequest | manager-to-agent: "get me data" (data instance, next data in list, block of data). |
| SetRequest | manager-to-agent: set MIB value |
| Response | Agent-to-manager: value, response to Request |
| Trap | Agent-to-manager: inform manager of exceptional event |

Network Layer: 5-95

95

## SNMP protocol: message formats




Network Layer: 5-96

96

## SNMP: Management Information Base (MIB)


- managed device's operational (and some configuration) data
- gathered into device MIB module
  - 400 MIB modules defined in RFC's; many more vendor-specific MIBs
- Structure of Management Information (SMI): data definition language
- example MIB variables for UDP protocol:

| Object ID | Name | Type | Comments |
|---|---|---|---|
| 1.3.6.1.2.1.7.1 | UDPInDatagrams | 32-bit counter | total # datagrams delivered |
| 1.3.6.1.2.1.7.2 | UDPNoPorts | 32-bit counter | # undeliverable datagrams (no application at port) |
| 1.3.6.1.2.1.7.3 | UDInErrors | 32-bit counter | # undeliverable datagrams (all other reasons) |
| 1.3.6.1.2.1.7.4 | UDPOutDatagrams | 32-bit counter | total # datagrams sent |
| 1.3.6.1.2.1.7.5 | udpTable | SEQUENCE | one entry for each port currently in use |



97



## NETCONF overview

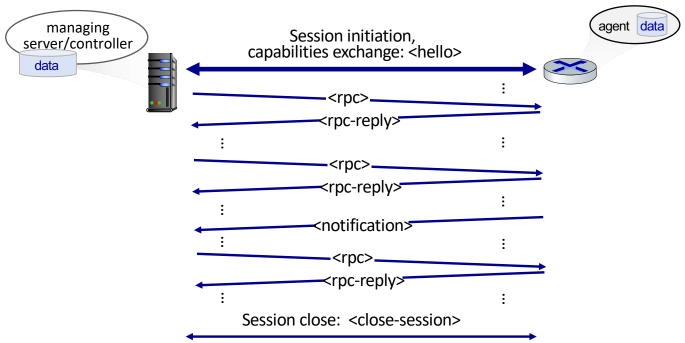
- goal: actively manage/configure devices network-wide
- operates between managing server and managed network devices
  - actions: retrieve, set, modify, activate configurations
  - atomic-commit actions over multiple devices
  - query operational data and statistics
  - subscribe to notifications from devices
- remote procedure call (RPC) paradigm
  - NETCONF protocol messages encoded in XML
  - exchanged over secure, reliable transport (e.g., TLS) protocol



98



## NETCONF initialization, exchange, close






99

## Selected NETCONF Operations

| NETCONF | Operation Description |
|---|---|
| <get-config> | Retrieve all or part of a given configuration. A device may have multiple configurations. |
| <get> | Retrieve all or part of both configuration state and operational state data. |
| <edit-config> | Change specified (possibly running) configuration at managed device. Managed device <rpc-reply> contains <ok> or <rpcerror> with rollback. |
| <lock>, <unlock> | Lock (unlock) configuration datastore at managed device (to lock out NETCONF, SNMP, or CLIs commands from other sources). |
| <create-subscription>, <notification> | Enable event notification subscription from managed device |



100

## Sample NETCONF RPC message

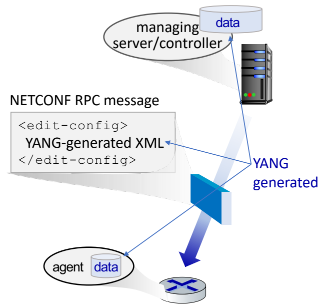
```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101"  note message id
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>        change a configuration
    <target>
      <running/>  change the running configuration
    </target>
    <config>
      <top xmlns="http://example.com/schema/
      1.2/config">
         <interface>
            <name>Ethernet0/0</name>  change MTU of Ethernet 0/0 interface to 1500
            <mtu>1500</mtu>
         </interface>
      </top>
    </config>
  </edit-config>
</rpc>
```



101

## YANG

- data modeling language used to specify structure, syntax, semantics of NETCONF network management data
  - built-in data types, like SMI
- XML document describing device, capabilities can be generated from YANG description
- can express constraints among data that must be satisfied by a valid NETCONF configuration
  - ensure NETCONF configurations satisfy correctness, consistency constraints






102



## Network layer: Summary

we've learned a lot!

- approaches to network control plane
  - per-router control (traditional)
  - logically centralized control (software defined networking)
- traditional routing algorithms
  - implementation in Internet: OSPF , BGP
- SDN controllers
  - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

*next stop: link layer!*



103

## Network layer, control plane: Done!

- 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



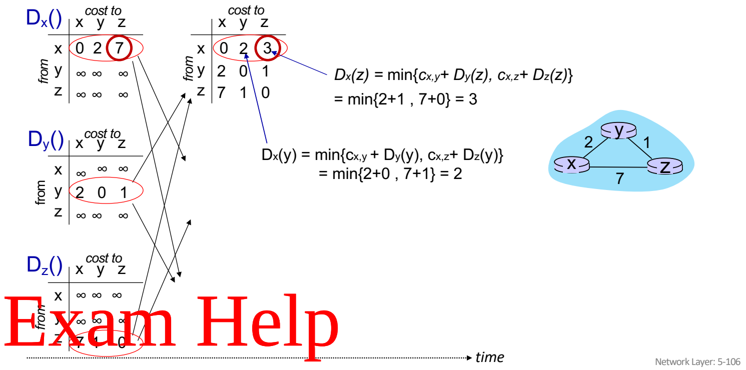
104

## Additional Chapter 5 slides



105

## Distance vector: another example

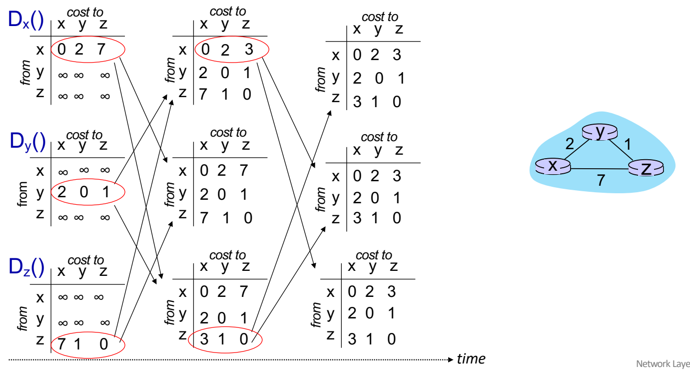





106



## Distance vector: another example






107