

Chapter 5

Network Layer: Control Plane

A note on the use of these PowerPoint slides:

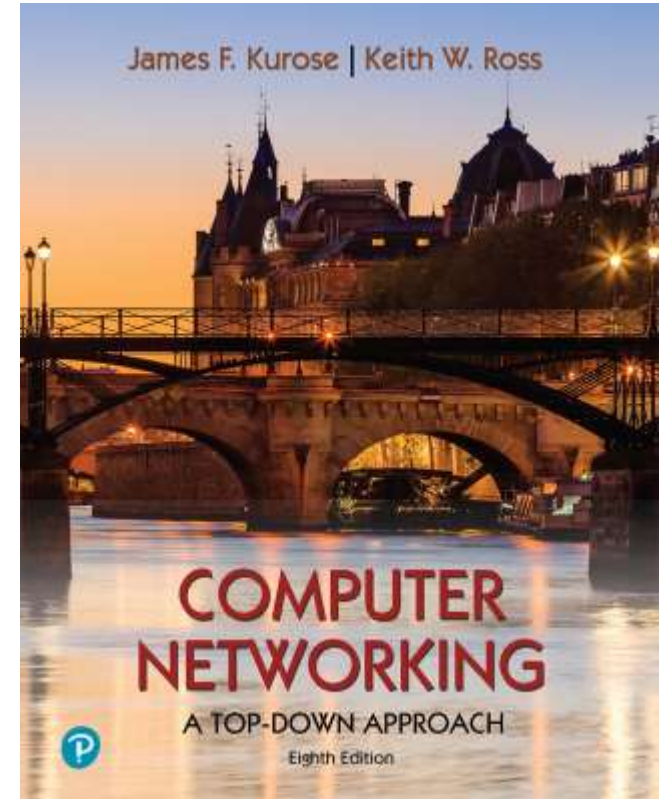
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use,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

All material copyright 1996-2023
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top-Down Approach*

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

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

Network layer: “control plane” roadmap

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



- network management, configuration
 - SNMP
 - NETCONF/YANG

Network-layer functions

- **forwarding**: move packets from router's input to appropriate router output
- **routing**: determine route taken by packets from source to destination

data plane

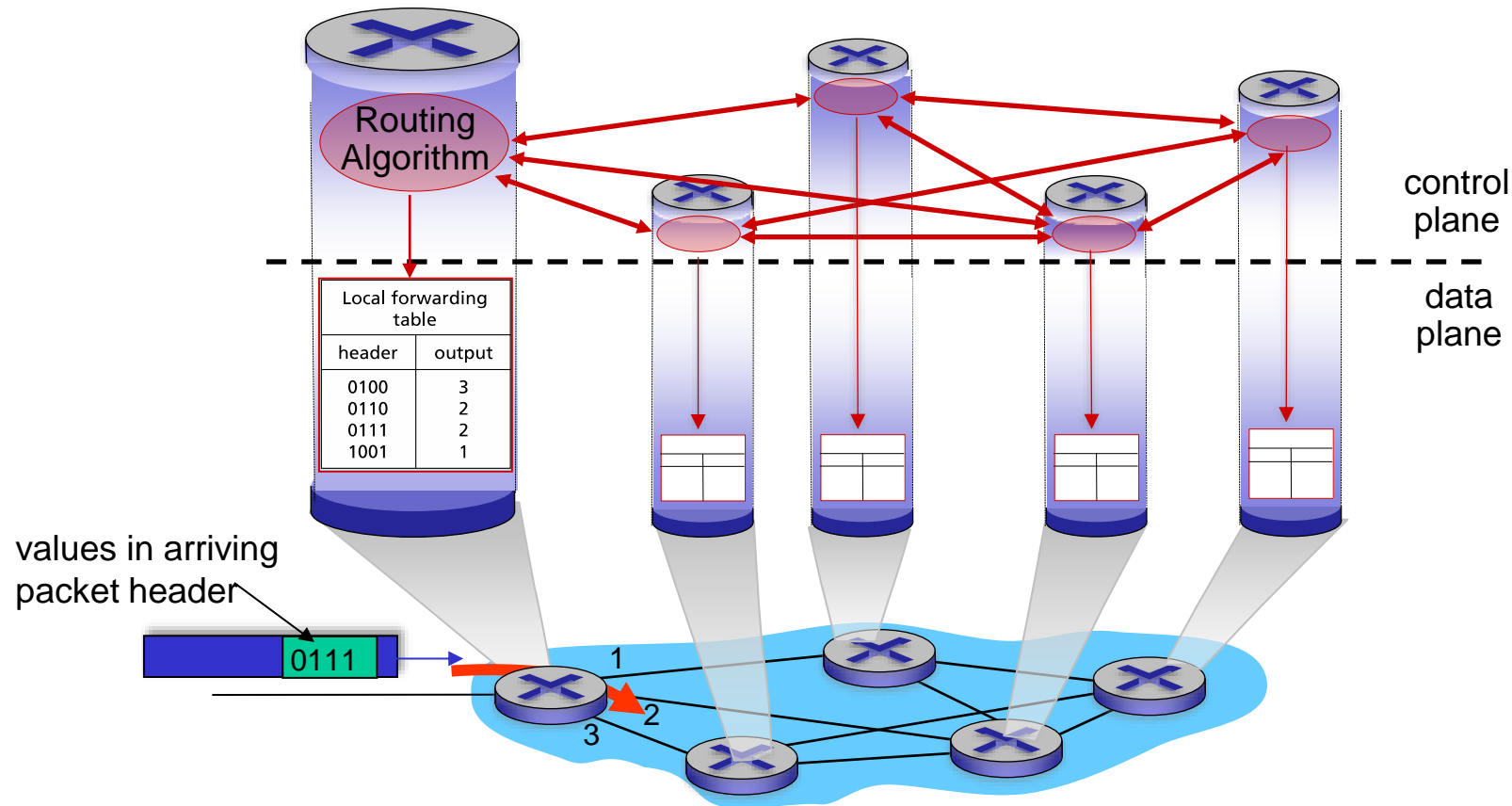
control plane

Two approaches to structuring network control plane:

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

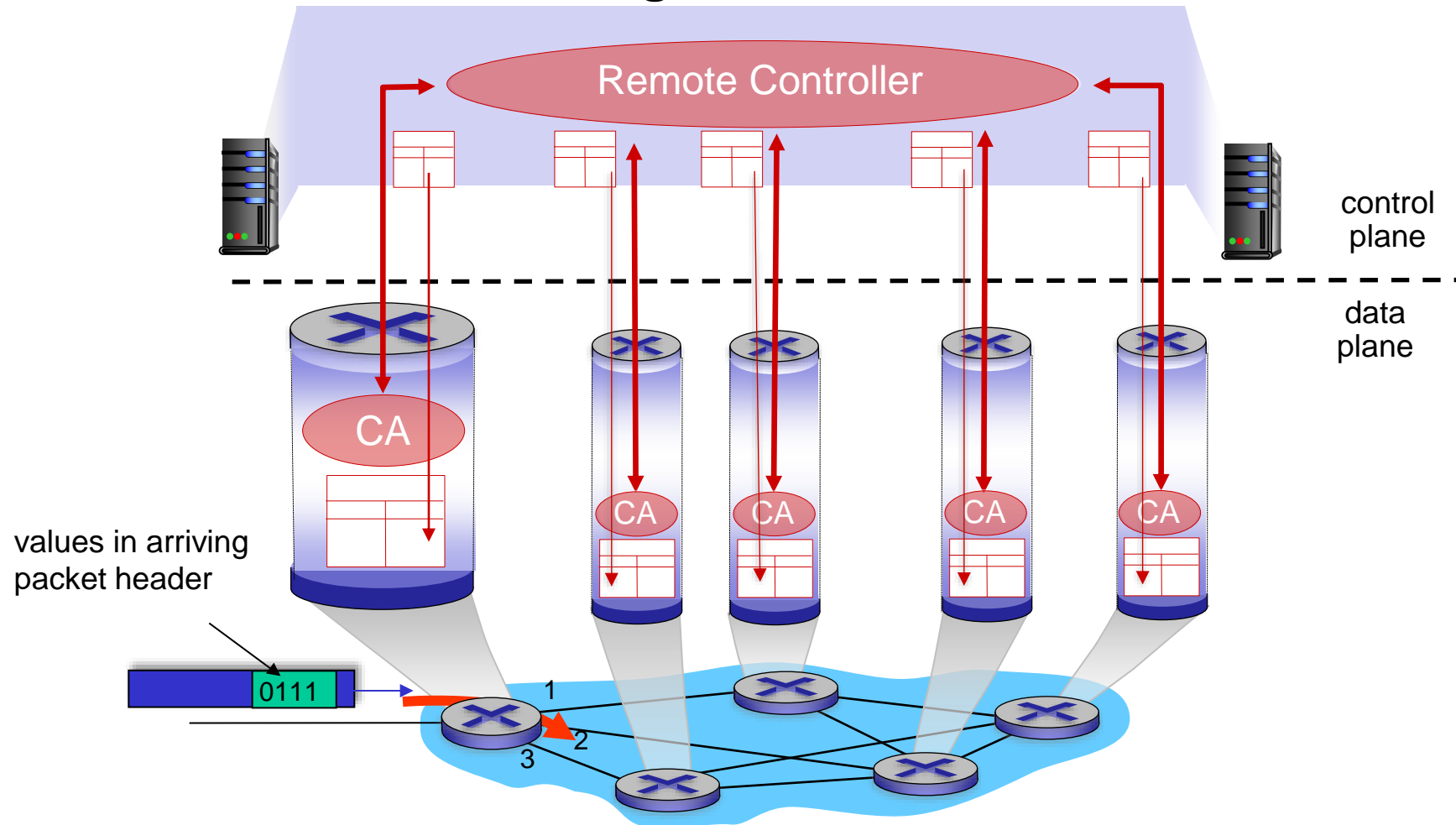
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane. OSPF and BGP protocols.

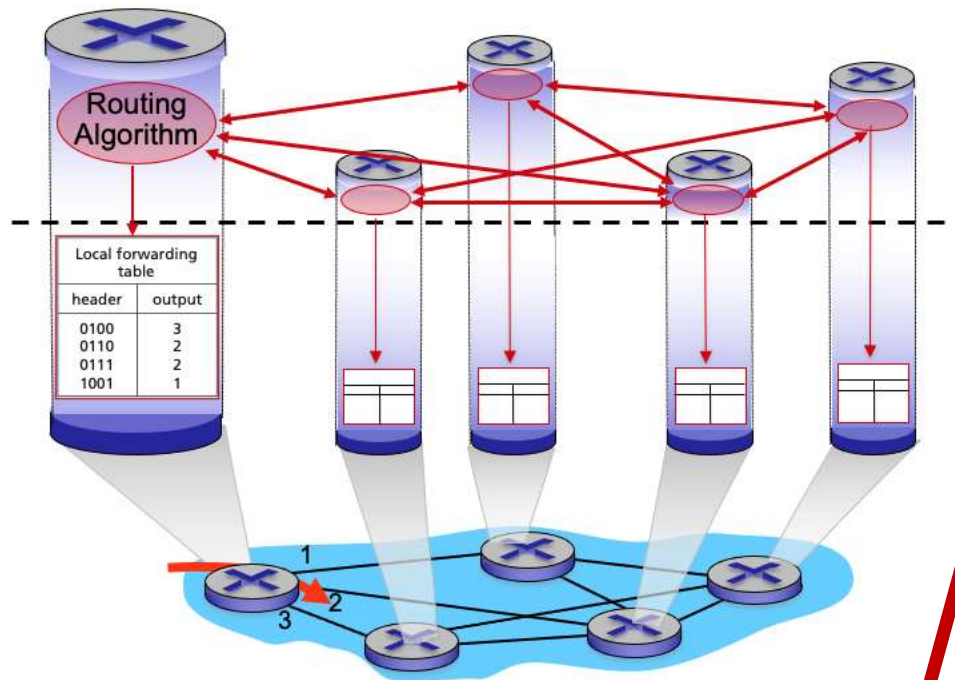


Software-Defined Networking (SDN) control plane

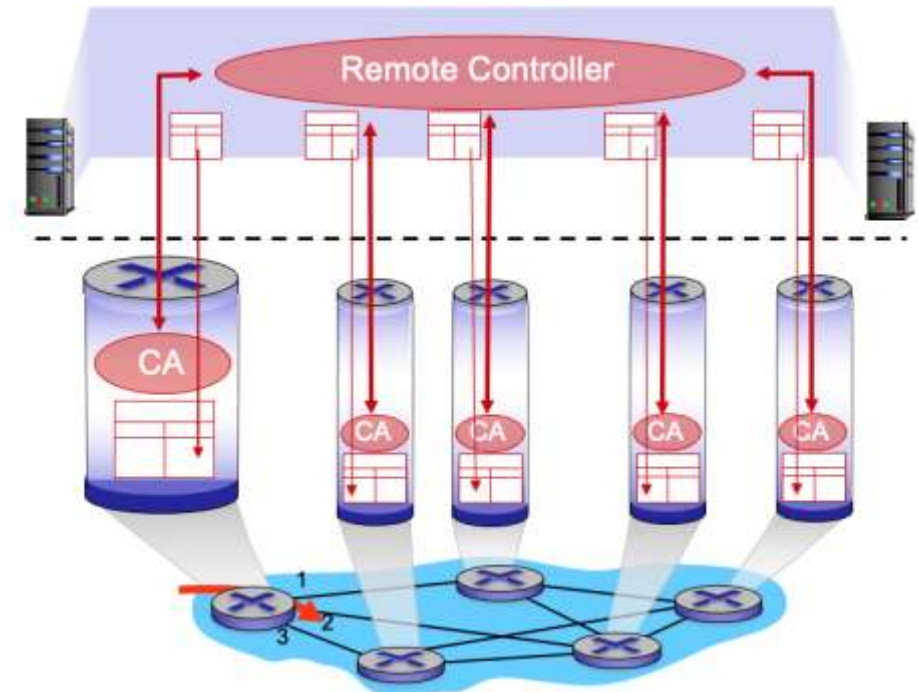
Remote controller computes, installs forwarding tables in routers via control agents



Per-router control plane



SDN control plane



Network layer: “control plane” roadmap

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

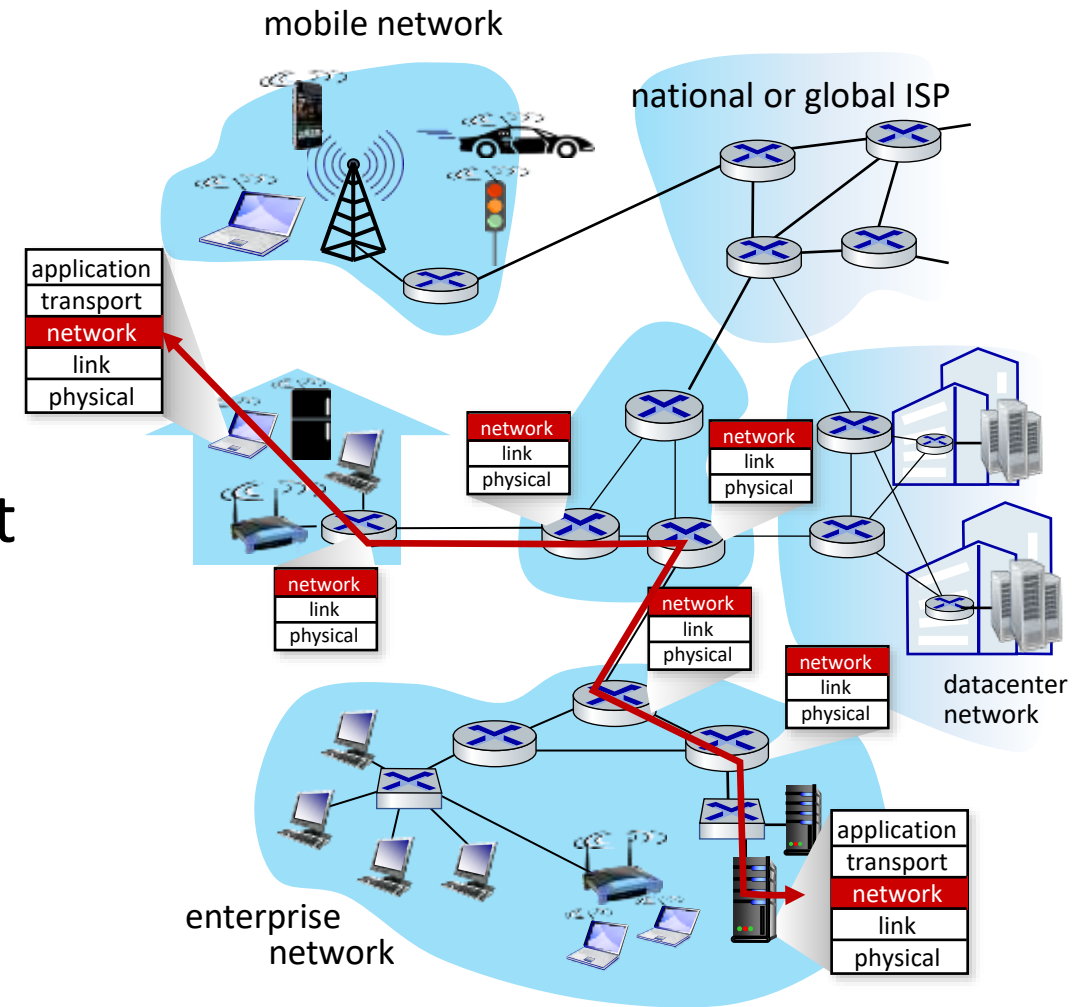


- network management, configuration
 - SNMP
 - NETCONF/YANG

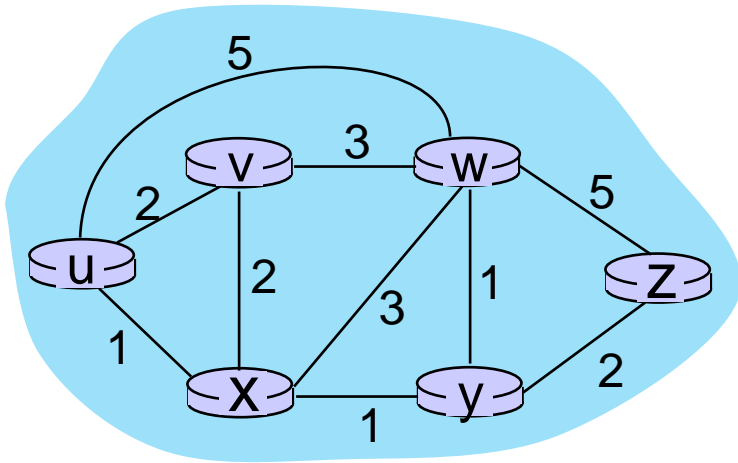
Routing protocols

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

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!



Graph abstraction: link costs



$c_{a,b}$: cost of *direct* link connecting a and b

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

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

graph: $G = (N, E)$

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

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

Routing Algorithms Classification: Centralized versus Decentralized

- **Centralized Routing Algorithm** - computes the least-cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site or could be replicated in the routing component of each and every router.
- The key distinguishing feature here is that the algorithm has complete information about connectivity and link costs. Algorithms with global state information are often referred to as **link-state** (LS) algorithms, since the algorithm must be aware of the cost of each link in the network.

Routing Algorithms Classification: Centralized versus Decentralized

- **Decentralized Routing Algorithms** - the calculation of the least-cost path is carried out in an iterative, distributed manner by the routers. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the 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 the least-cost path to a destination or set of destinations. The decentralized routing algorithm is called a **distance-vector (DV) algorithm**, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network. Such decentralized algorithms, with interactive message exchange between neighboring routers is perhaps more naturally suited to control planes where the routers interact directly with each other

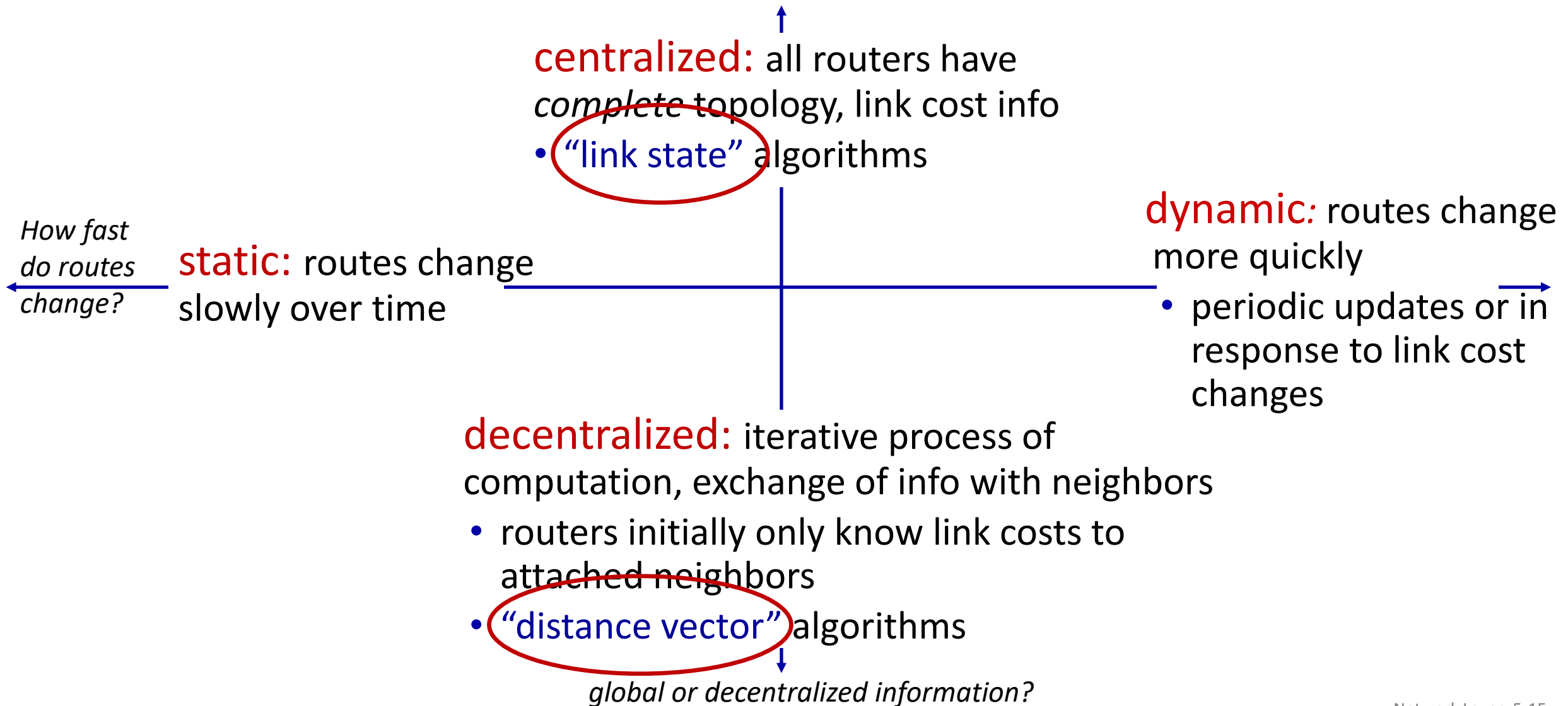
Routing Algorithms Classification: Static versus Dynamic

- **Static Routing Algorithms:** routes change very slowly over time, as a result of human intervention.
- **Dynamic Routing Algorithms:** change the routing paths as the network traffic loads or topology changes.
 - Can be run either periodically or in response to topology or link cost savings
 - More responsive to network changes, also more susceptible to problems such as routing loops

Routing Algorithms Classification: Load-Sensitive or Load-Insensitive

- **Load-Sensitive** – link costs vary dynamically to reflect the current level of congestion in the 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.
- **Load – Insensitive** – consistent with today's Internet routing algorithms (OSPF, BGP, RIP), as a link's cost does not explicitly reflect its current level of congestion.

Routing algorithm classification



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

The Link-State (Centralized) Routing Algorithms

- All links and all costs are known which means available as input to the LS algorithm
- This is accomplished by having each node broadcast link-state packets to all other nodes in the network with identities and costs for its attached links. **Link-state broadcast algorithm**
- Result: all nodes have an identical and complete view of the network
- **Dijkstra's algorithm** computes the least-cost path from one node to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes

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

Network layer: “control plane” roadmap

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



- network management, configuration
 - SNMP
 - NETCONF/YANG

Distance vector algorithm

- LS algorithm is an algorithm using global information, the **distance- vector (DV) algorithm is iterative, asynchronous, and distributed.**
- It is distributed in that each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors.
- It is iterative in that this process continues on until no more information is exchanged between neighbors. (Interestingly, the algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.)
- The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.

Distance vector algorithm

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

Bellman-Ford equation

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

Then:

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

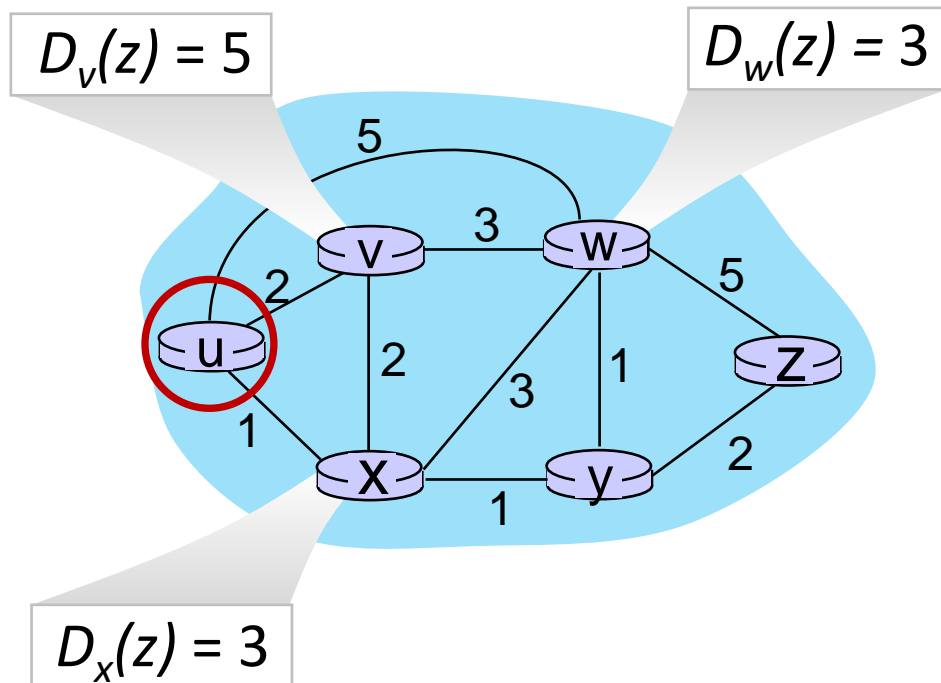
v 's estimated least-cost-path cost to y

\min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

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



Bellman-Ford equation says:

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

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

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 DV is used by others: error propagate thru network

Network layer: “control plane” roadmap

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



- network management, configuration
 - SNMP
 - NETCONF/YANG

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... not true in practice

scale: billions of destinations:

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

administrative autonomy:

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

Internet approach to scalable routing

- aggregate routers into regions known as “autonomous systems” (AS) (a.k.a. “domains”); with each AS consisting of a group of router that are under the same administrative control.
- The routing algorithm running within an autonomous system is called an intra-autonomous system routing algorithm

Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

- **RIP: Routing Information Protocol** [RFC 1723]
 - classic DV: DVs exchanged every 30 secs
 - no longer widely used
- **EIGRP: Enhanced Interior Gateway Routing Protocol**
 - DV based
 - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- **OSPF: Open Shortest Path First** [RFC 2328]
 - link-state routing
 - Current version 2

OSPF (Open Shortest Path First) routing

- OSPF is a **link-state** protocol that uses Dijkstra's least-path algorithm.
- Each router constructs a complete topological map (graph) of the entire autonomous system.
- Each router that locally runs shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node.
- Individual link costs are configured by the network administrator.
 - The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links.

OSPF Routing

- Router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring routers.
- Router broadcasts link-state information whenever there is a change in a link's state (change in cost)
- Link's state periodically (at least once every 30 minutes), even if no change

OSPF Routing

- Security: Exchanges between OSPF routers (link-state updates), can be authenticated
- With **Authentication**, only trusted routers can participate in the OSPF protocol within autonomous system; hence, un-authorized devices will not be able to interject their potentially malicious entries into router tables.
- By default, OSPF packets between routers **are not authenticated, and could be forged.**
- Router send OSPF packet it includes the password in plaintext. MD5 hash algorithm helps with authentication, yet, clear text password still remain.
- Sequence numbers are also used with MD5 authentication to protect against replay attacks.

OSPF Routing

- Multiple same-cost paths. When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used.
- Integrated support for unicast (one-to-one transmission) and multicast routing. Multicast OSPF (MOSPF) provides an extension to OSPF to provide for multicasting (one-to-many kind of transmissions) routing.
- Support for hierarchy within a single AS. OSPF AS can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in the area broadcasting its link-state to all other routers in the area.

Hierarchical OSPF

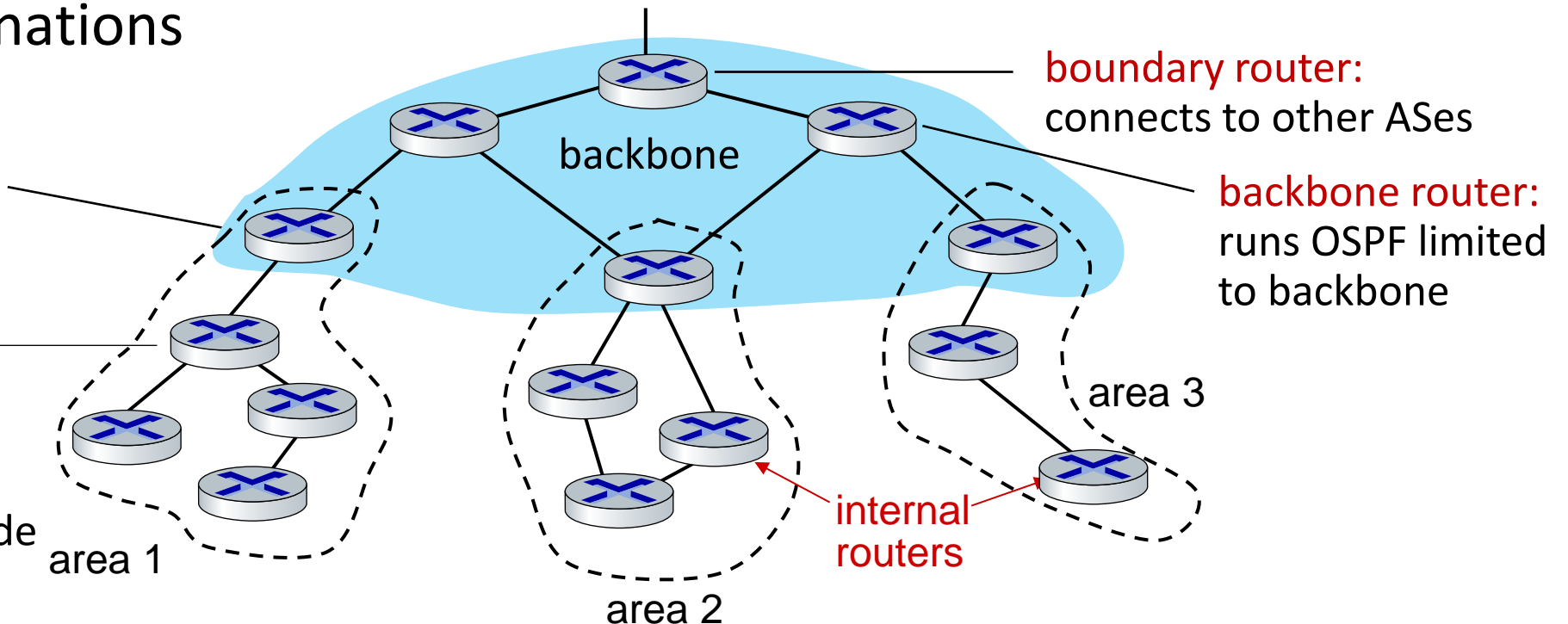
- **two-level hierarchy:** local area, **backbone**, routes traffic between the other areas in the AS.
 - link-state advertisements flooded only in area, or backbone
 - each node has detailed area topology; only knows direction to reach other destinations

area border routers:

“summarize” distances to destinations in own area, advertise in backbone

local routers:

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



Network layer: “control plane” roadmap

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

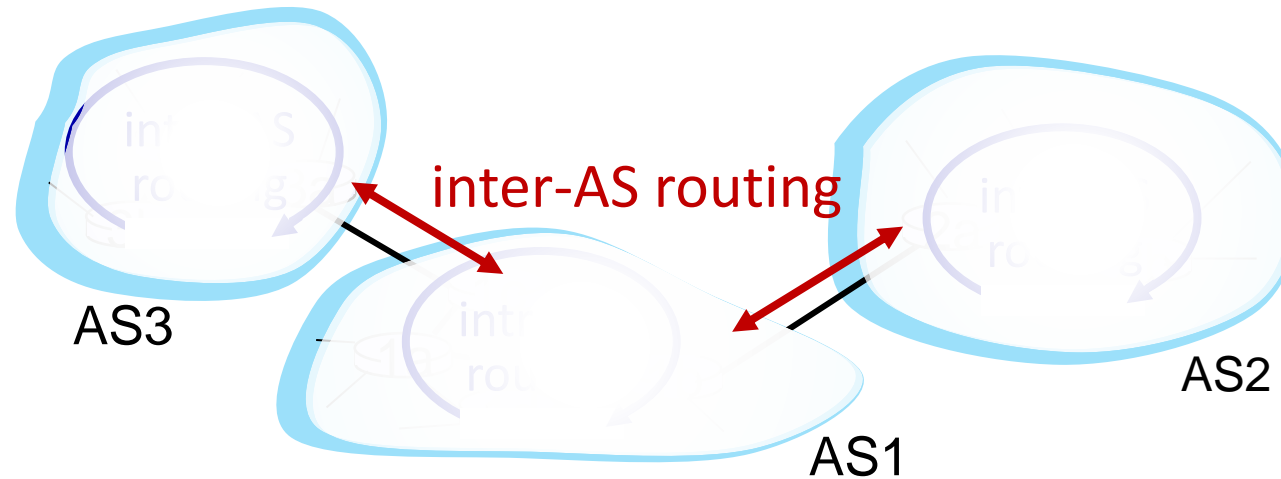


- network management, configuration
 - SNMP
 - NETCONF/YANG

Inter-Autonomous System Routing Protocol

- OSPF – **intra-AS routing protocol**
- when routing a packet between a source and destination within the same AS, the route the packet follows is determined by the intra-AS routing protocol.
- The route for packet across multiple AS's would require – **inter-autonomous system routing protocol**.
- All As's run the same inter-AS routing protocol. **Boarder Gateway Protocol – BGP**.
 - One of the most important Internet protocols, at parr with IP protocol. It “glues” thousands of ISPs together.
 - BGP is a decentralized and asynchronous protocol in the vein of distance-vector routing (decentralized)

Interconnected ASes



- ✓ **intra-AS (aka “intra-domain”)**: routing among routers *within same* AS (“*network*”)
- ➡ **inter-AS (aka “inter-domain”)**: routing *among* AS’s

Role of BGP

- Recall every router has a forwarding table, which plays the central role in the process of forwarding arriving packets to outbound router links.
- Destinations within the same AS, the entries in the router's forwarding table are determined by the AS's intra-AS routing protocol
- Destinations outside of the AS?
 - BGP
- In BGP, packets are not routed to a specific destination address, but instead to **CIDRized** (subnet mask or masks).
 - i.e. 138.16.68/22

BGP

- Obtain prefix (mask) reachability information from neighboring As's. BGP allows each subnet to advertise its existence to the rest of the Internet. If BGP wouldn't exist, then it would be like isolated AS islands.
- Determine the best route for/to subnets. A router may learn about two or more different routes to a specific prefix. The router will locally run a BGP route-selection procedure (using the prefix/subnet reachability information it obtained via neighboring routers).
- The best route will be determined based on the policy as well as the reachability information

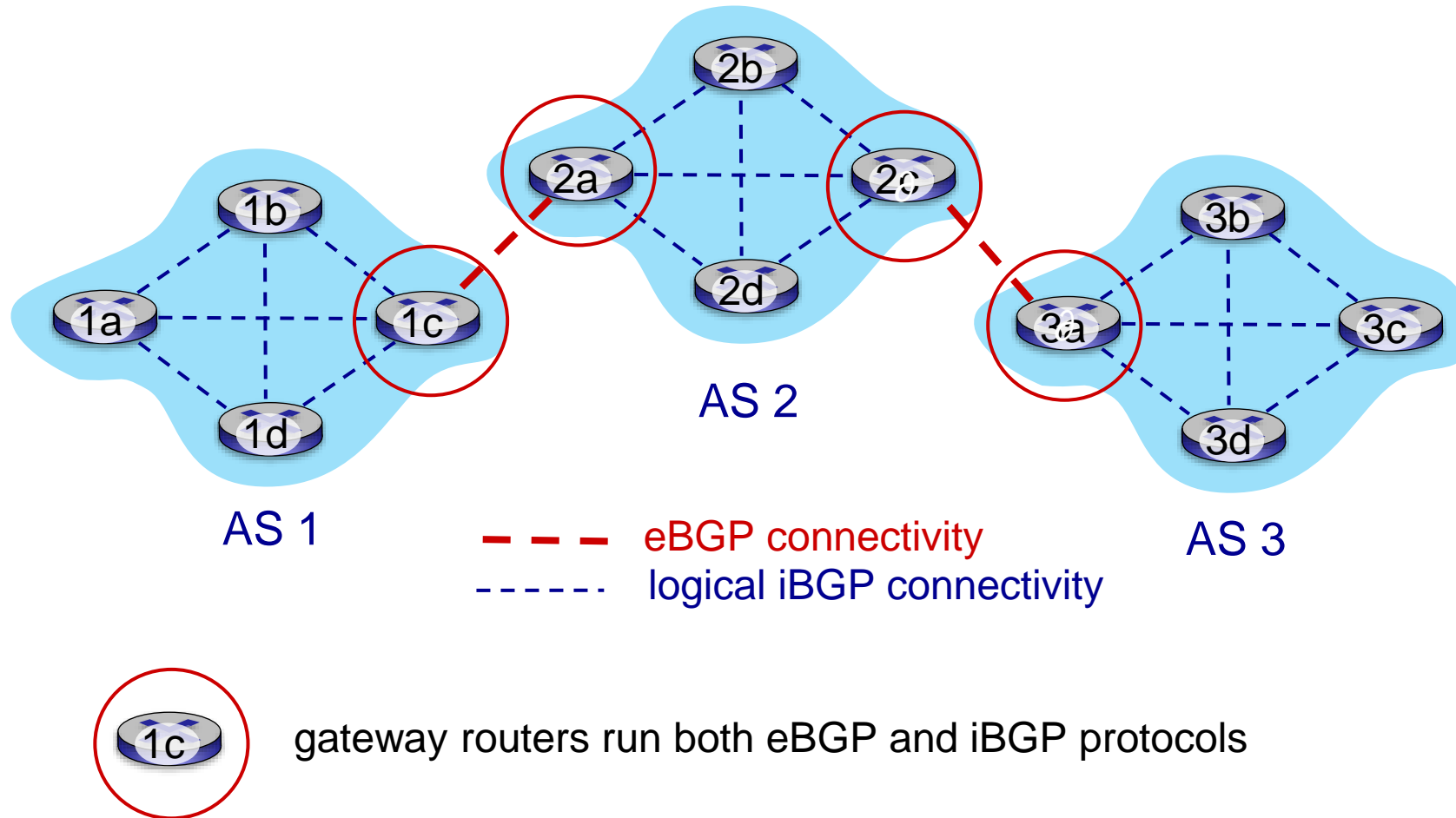
Advertising BGP Route Information

- Example with 3 Autonomous Systems
- AS3 includes a subnet with prefix x.
- For each AS, each router is either a **gateway router** or an **internal router**
 - Gateway router is a router on the edge of an AS that directly connects to one or more routers in the AS's.
 - Internal Router connects only to hosts and routers within its won AS.

Internet inter-AS routing: BGP

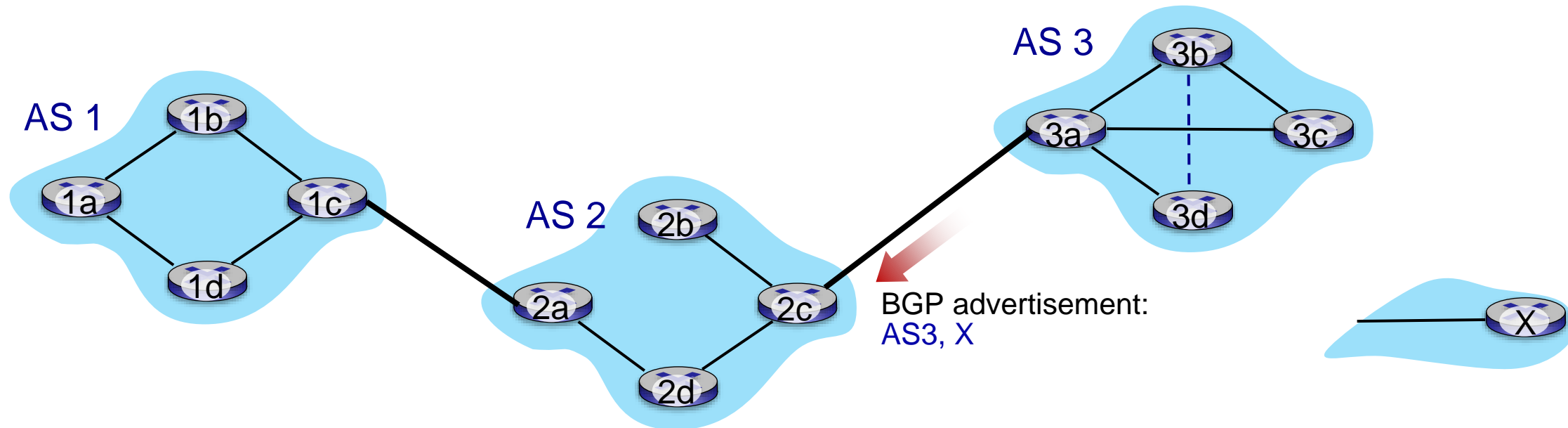
- In BGP, pairs of routers exchange routing information over TCP connections using port 179. Each TCP connection along with all the BGP “messages” sent over the connection is called **BGP connection**
- BGP connection that spans two AS's is called an external BGP (**eBGP**)
- BGP session between routers in the same AS is called an internal BGP (**iBGP**)

eBGP, iBGP connections



BGP basics

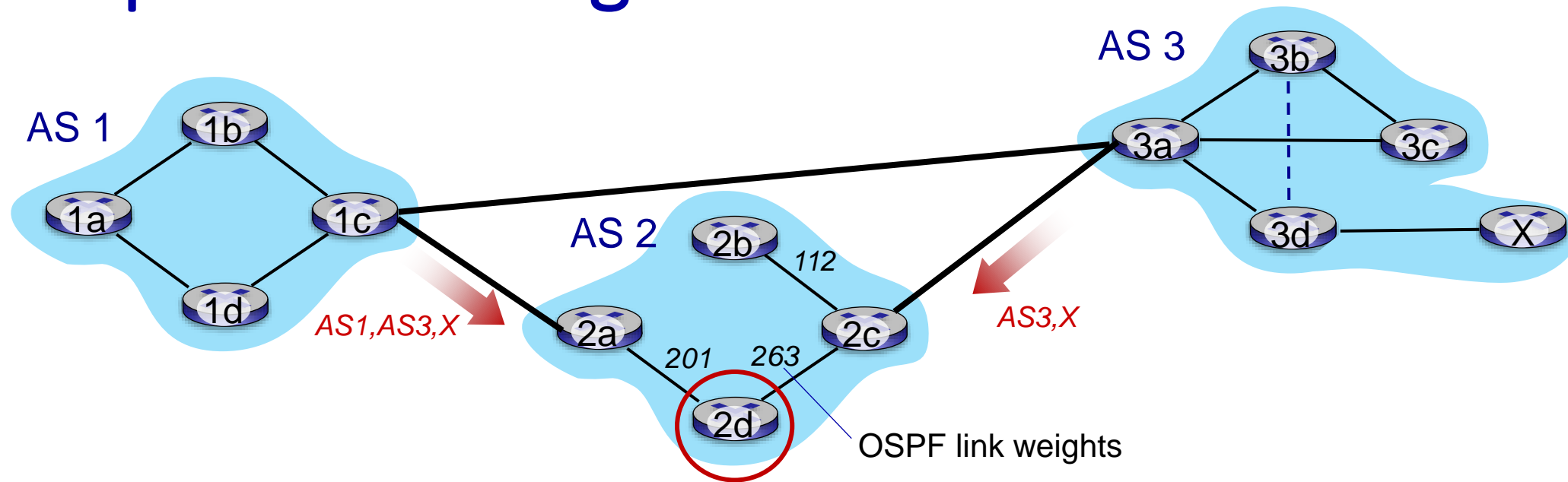
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
 - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway 3a advertises *path AS3,X* to AS2 gateway 2c:
 - AS3 *promises* to AS2 it will forward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



Path attributes and BGP routes

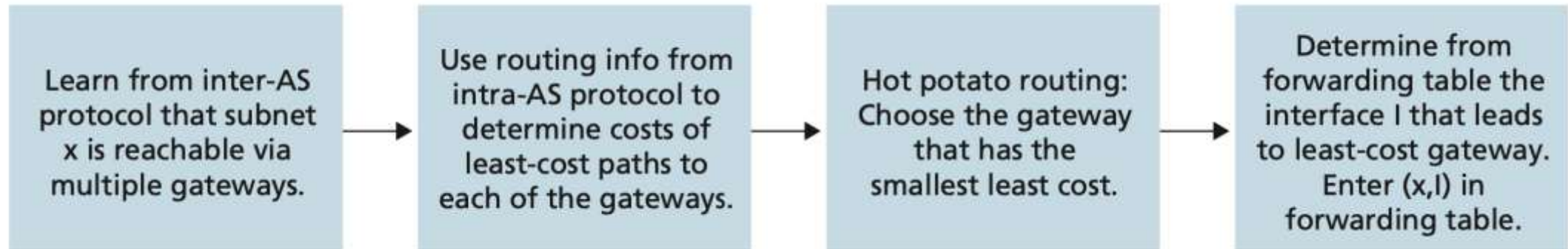
- BGP advertised route: prefix + attributes
 - prefix: destination being advertised
 - two important attributes:
 - **AS-PATH**: list of AS's through which prefix advertisement has passed
 - **NEXT-HOP**: indicates specific IP address of the router interface that begins the

Hot potato routing



- Consider 1b router to subdomain X. learns via iBGP to router 2a, and the least-cost to the router 3d.
- **hot potato routing**: choose local gateway that has least cost to the NEXT-HOP router

Hot Potato routing Lifecycle



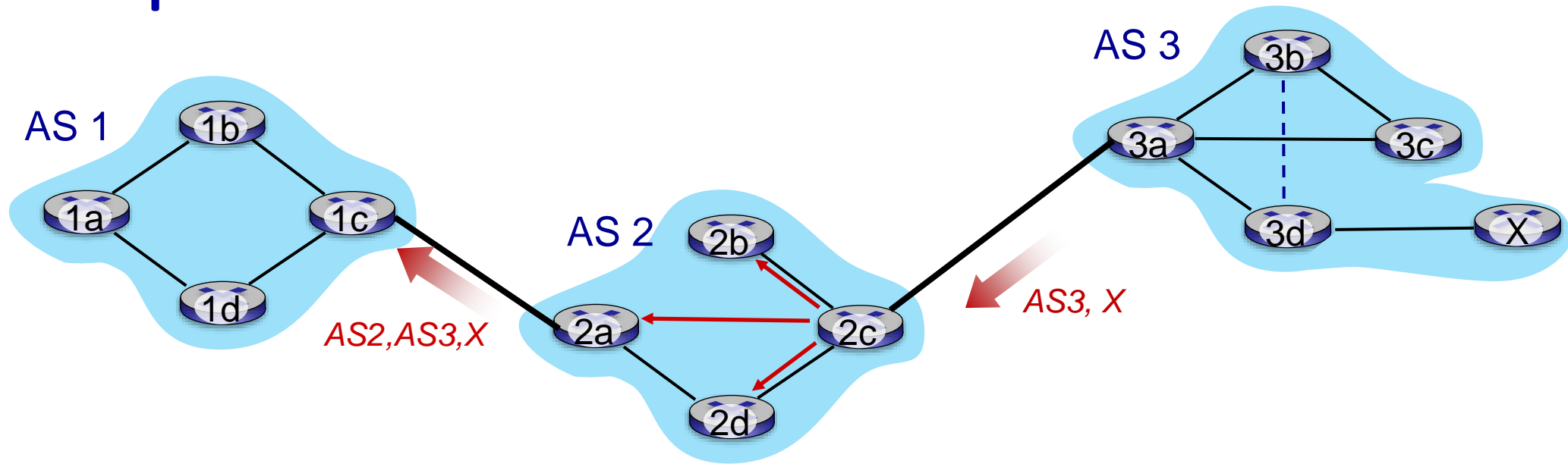
BGP routing

- In practice, BGP uses an algorithm more complicated than hot potato routing.
- For any destination subdomain (prefix), the input into BGP's route –selection algorithm is the set of all routes to that sub-domain that have been learned and accepted by the router.
- If there are 2 or more routes then BGP sequentially invokes the following elimination rules, until one route remains:
 - Local Preference (decided by local admin)
 - The route with the shortest AS-PATH is selected
 - Hot potato routine is used, route with the closest NEXT-HOP

BGP protocol messages

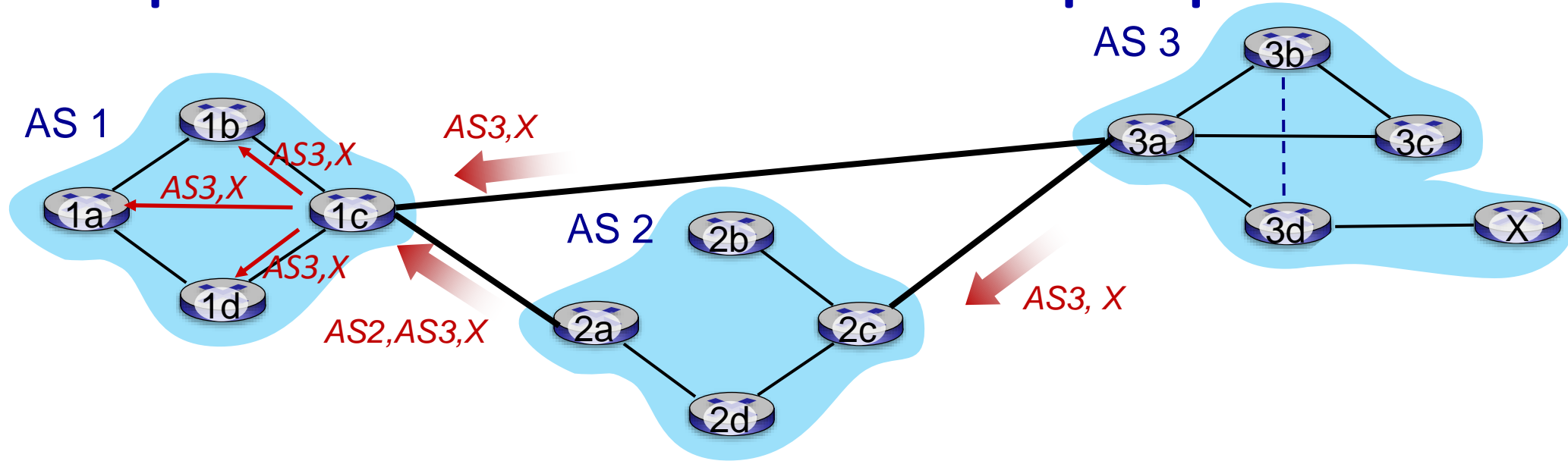
- BGP messages exchanged between peers over TCP connection
- BGP messages [RFC 4371]:
 - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
 - **UPDATE**: advertises new path (or withdraws old)
 - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
 - **NOTIFICATION**: reports errors in previous msg; also used to close connection

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

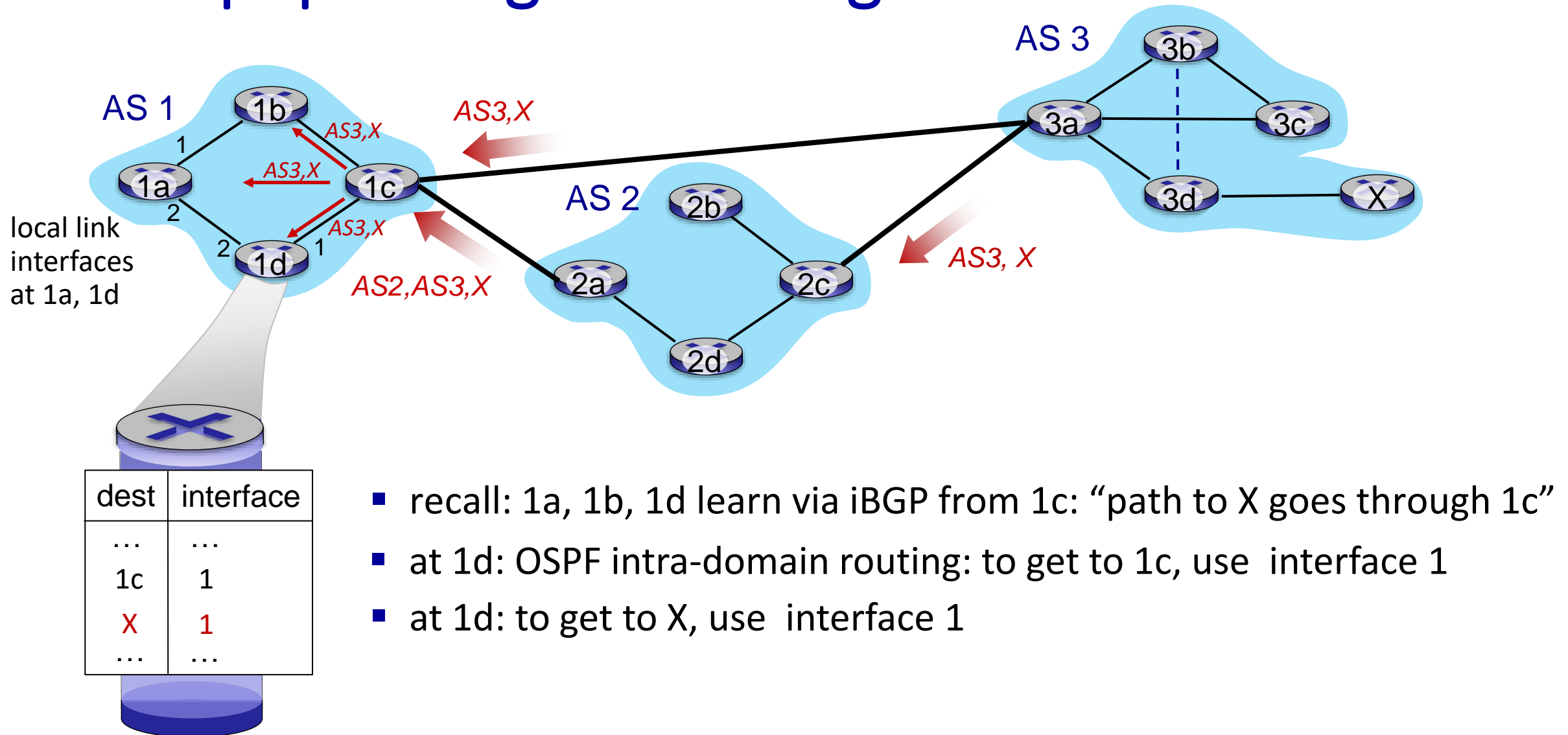
BGP path advertisement: multiple paths



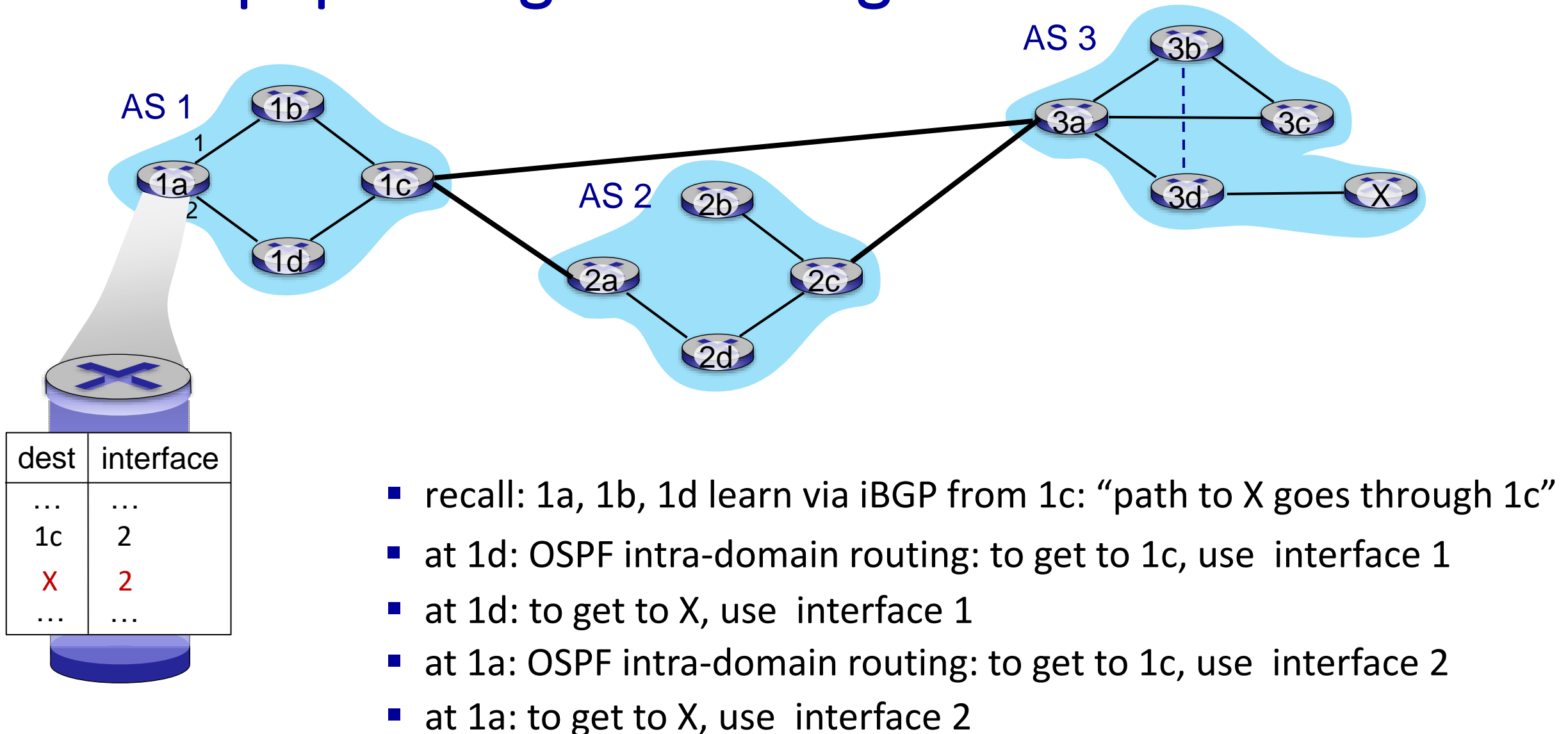
gateway router may learn about **multiple** paths to destination:

- 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

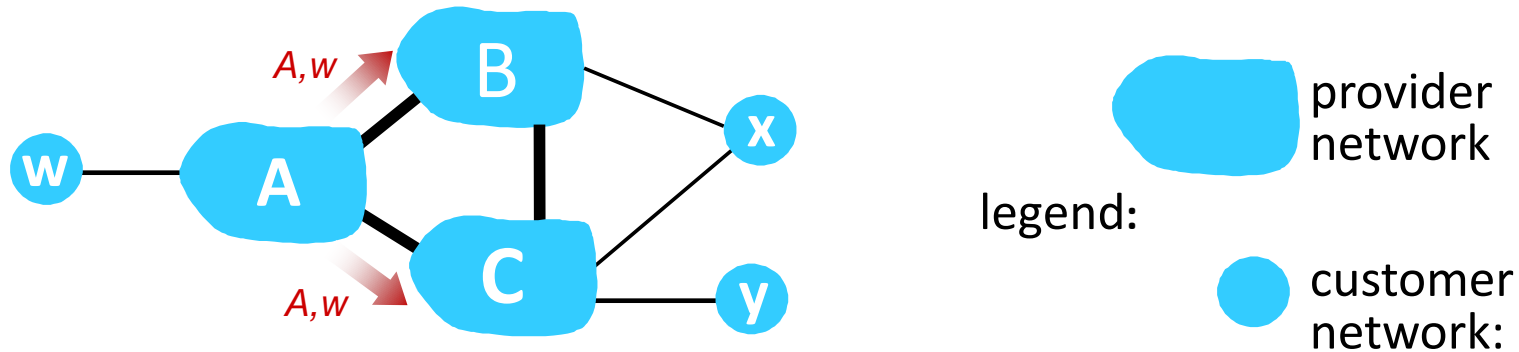
BGP: populating forwarding tables



BGP: populating forwarding tables



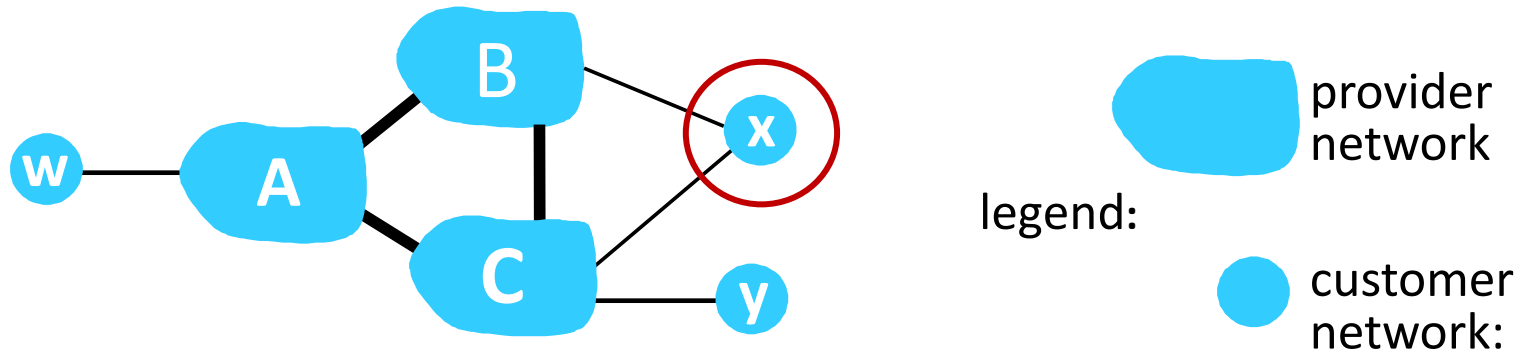
BGP: achieving policy via advertisements



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* BA_w to C!
 - B gets no “revenue” for routing CBA_w, since none of C, A, w are B’s customers
 - C does *not* learn about CBA_w path
- C will route CA_w (not using B) to get to w

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

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

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

Network layer: “control plane” roadmap

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



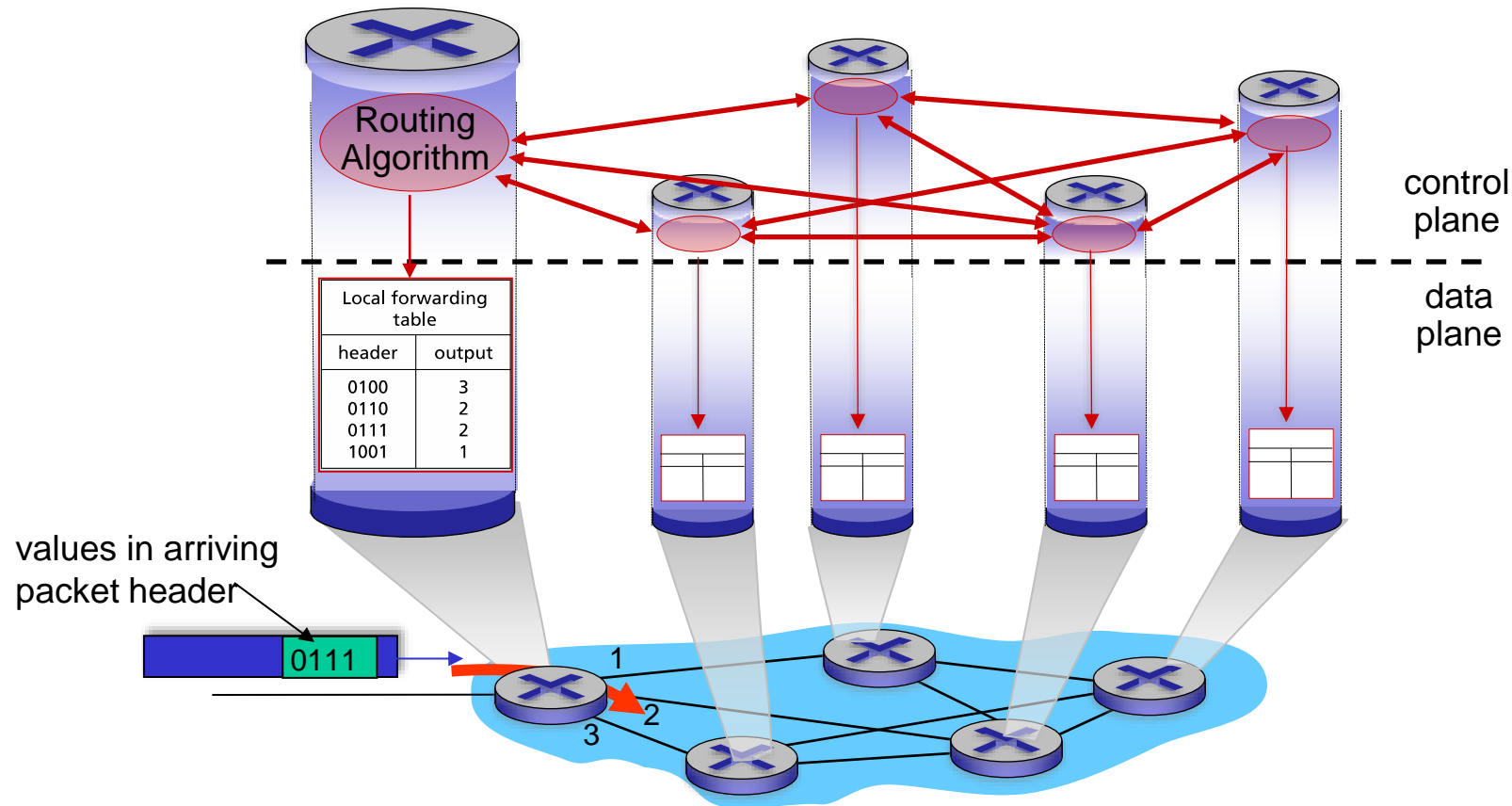
- network management, configuration
 - SNMP
 - NETCONF/YANG

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

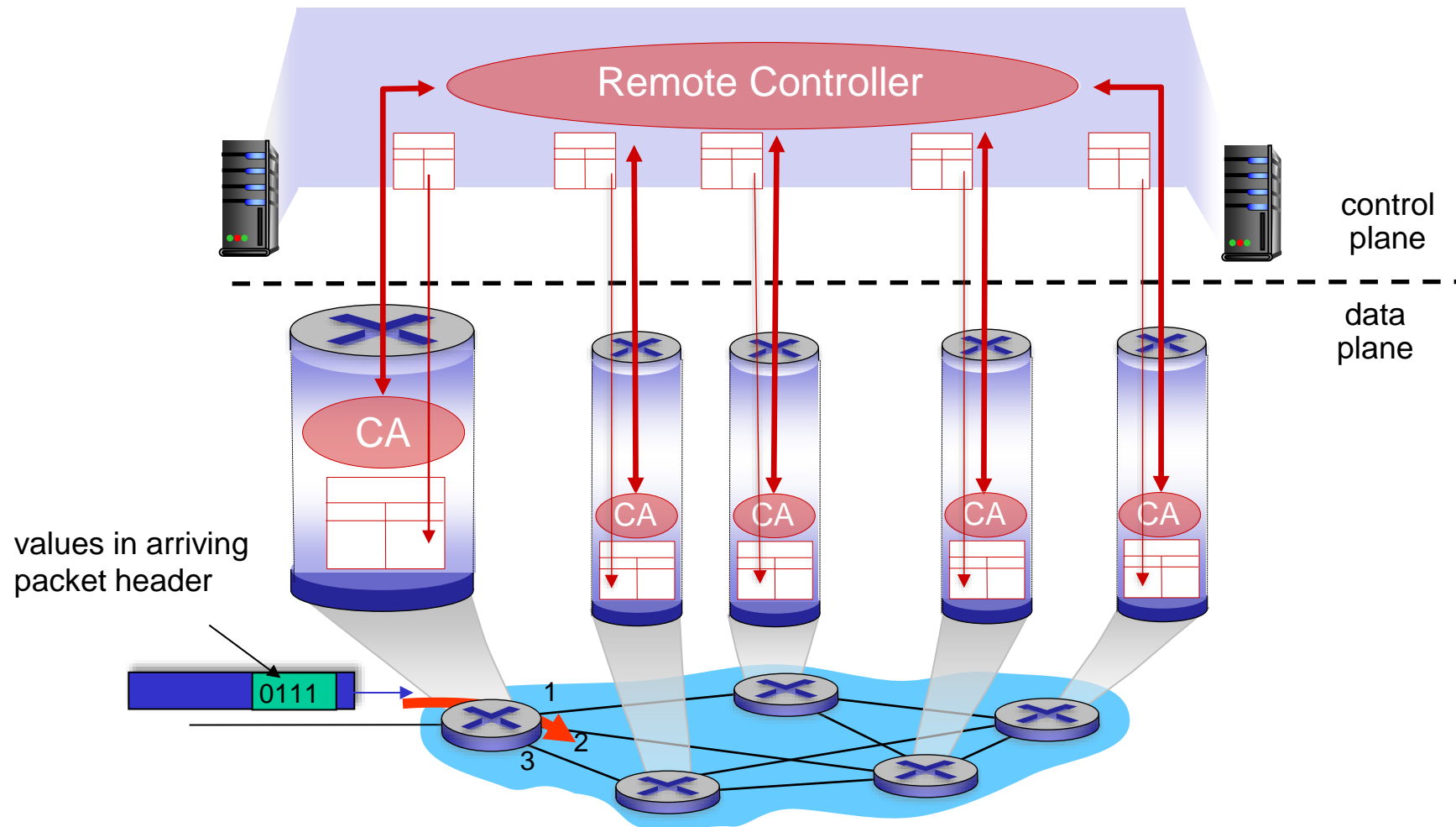
Per-router control plane

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



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

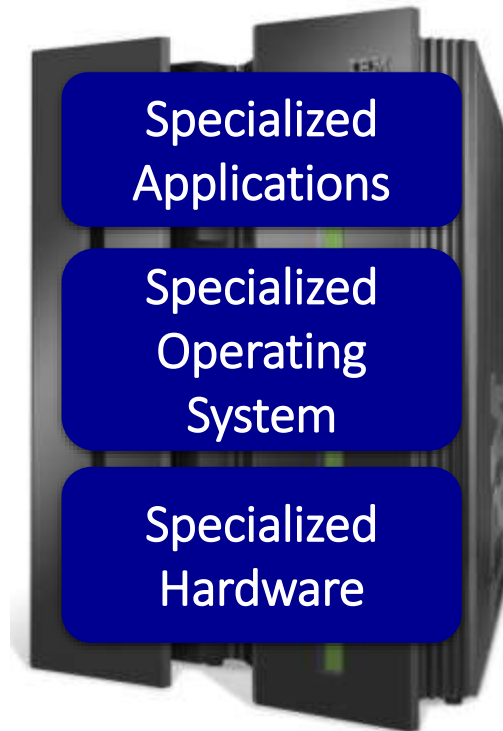


Software defined networking (SDN)

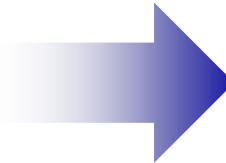
Why a *logically centralized* control plane?

- 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

SDN analogy: mainframe to PC revolution



Vertically integrated
Closed, proprietary
Slow innovation
Small industry



— Open Interface —



Windows



Linux



MAC OS

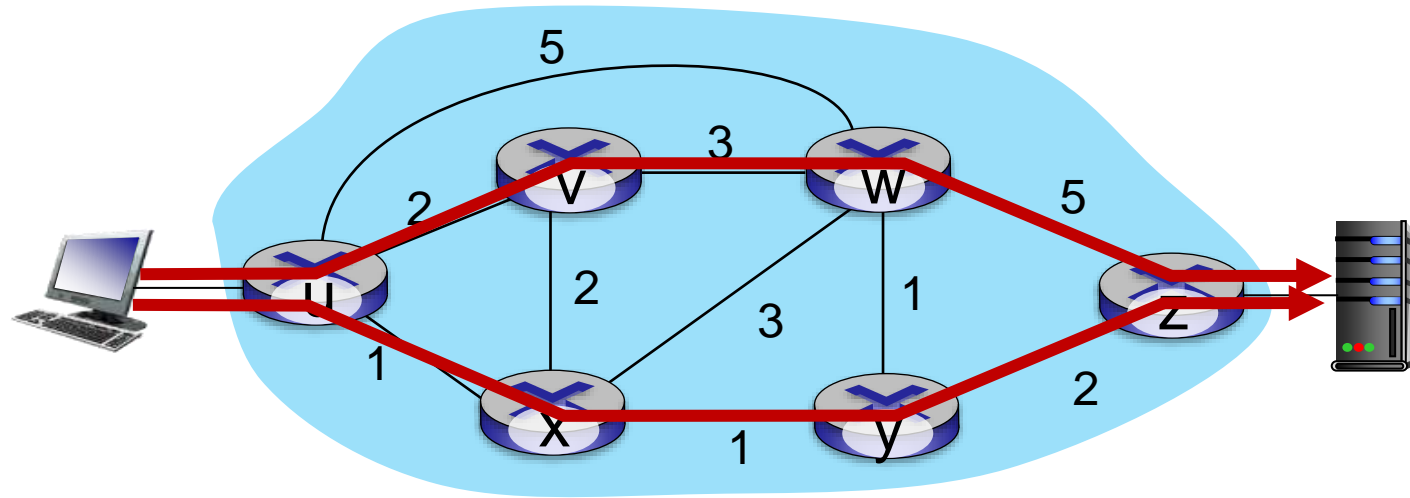
— Open Interface —



Microprocessor

Horizontal
Open interfaces
Rapid innovation
Huge industry

Traffic engineering: difficult with traditional routing

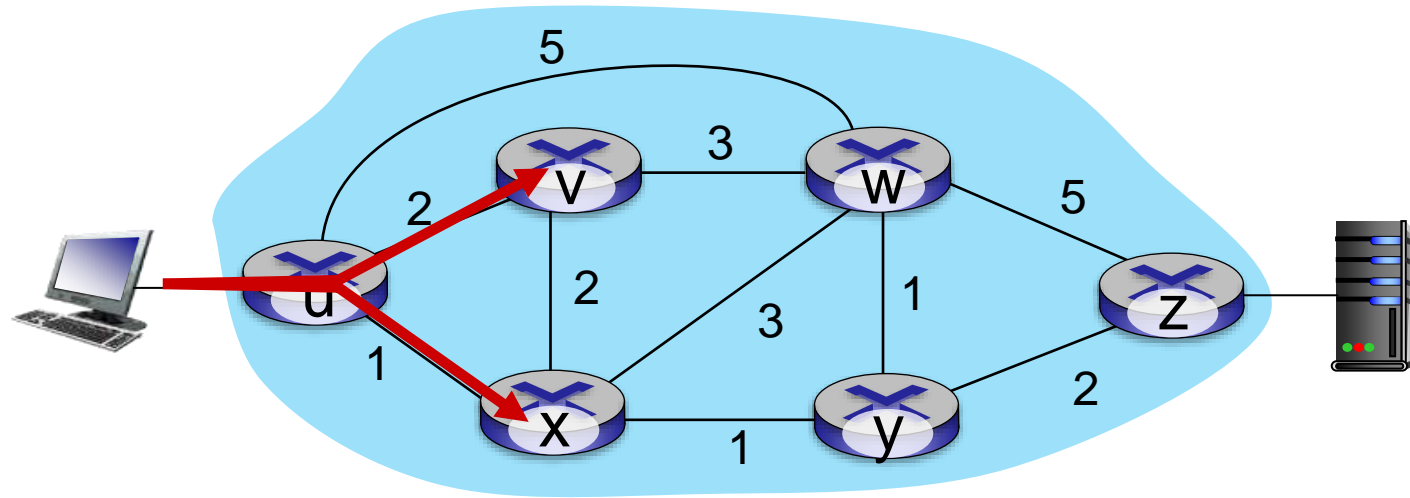


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

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!

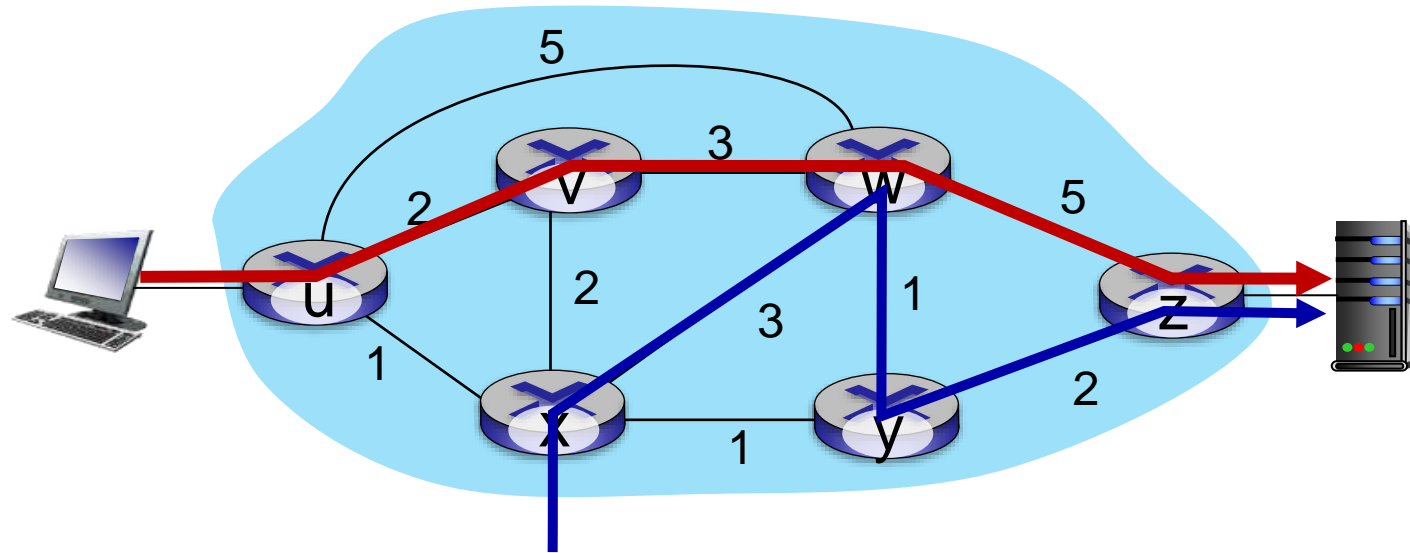
Traffic engineering: difficult with traditional routing



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)

Traffic engineering: difficult with traditional routing



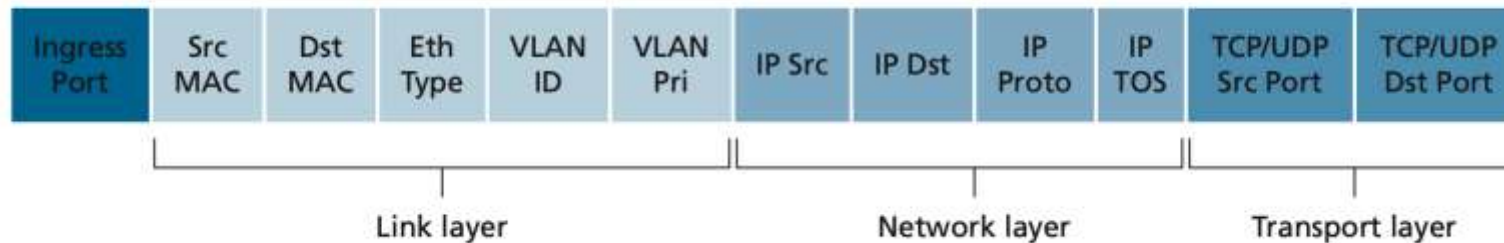
Q: what if we want to route blue and red traffic differently from w to z?

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

SDN OpenFlow - Functionality

- SDN packet filtering functions at **3 different layers**, unlike the other network devices
- Capable of "reading" 11 –packet-header fields and the incoming port ID



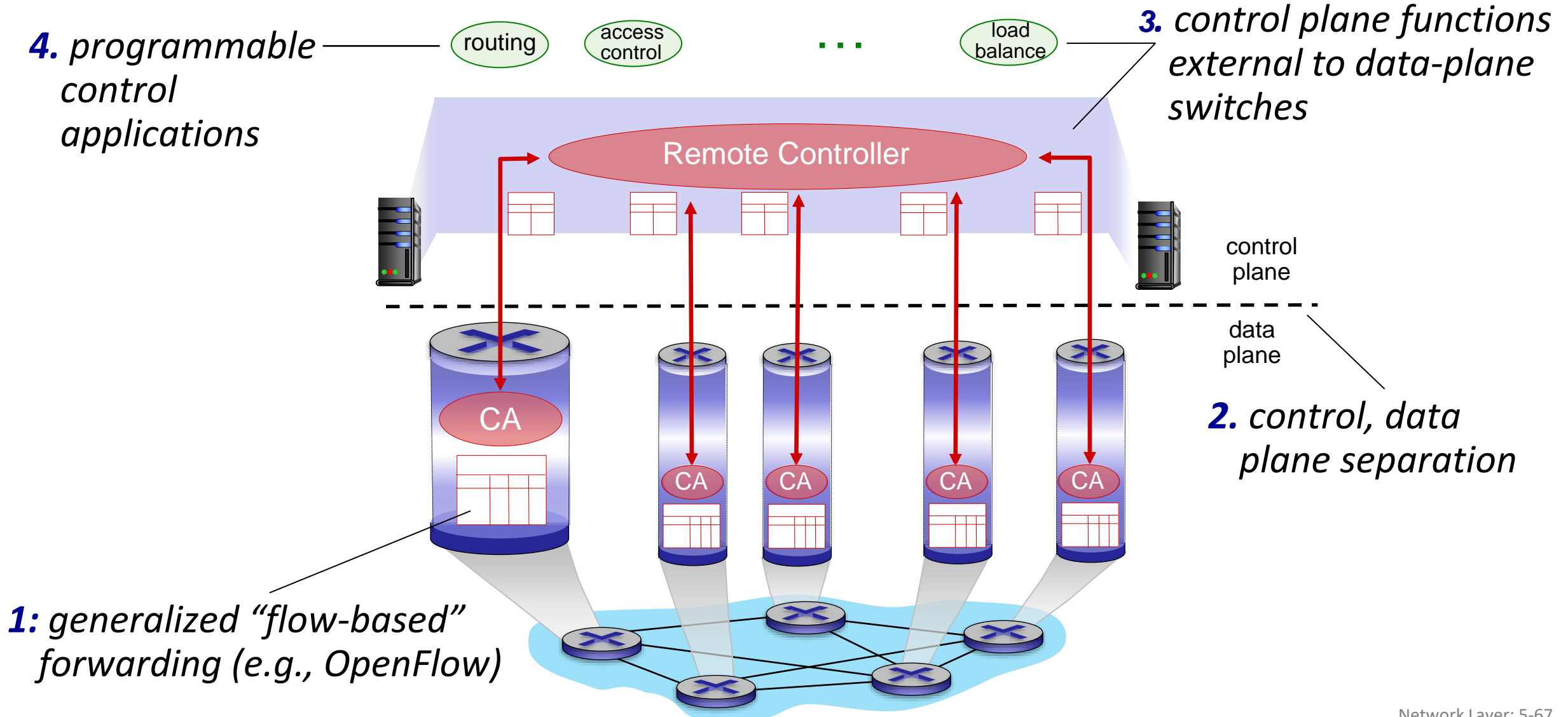
4 Key Characteristics of SDN Architecture

- Flow-based forwarding: packet forwarding by SDN-controlled switches can be based on any number of header field values in the transport-layer, network-layer, link-layer header.
 - In contrast to traditional router-based forwarding where datagrams forwarding was based on datagram's destination IP address.
- Separation of data plane and control plane. Data plane consists of multiple network switches; control plane consists of servers and software that determine and manage the switches flow tables

4 Key Characteristics of SDN Architecture

- Network Control Functions: external to data-plane switches. SDN control plane is implemented in software. Controller is logically centralized; it is typically implemented on several servers that provide coordinated, scalable performance and high availability.
- Programmable network: network is programmable through the network-control applications & APIs running in the control plane.

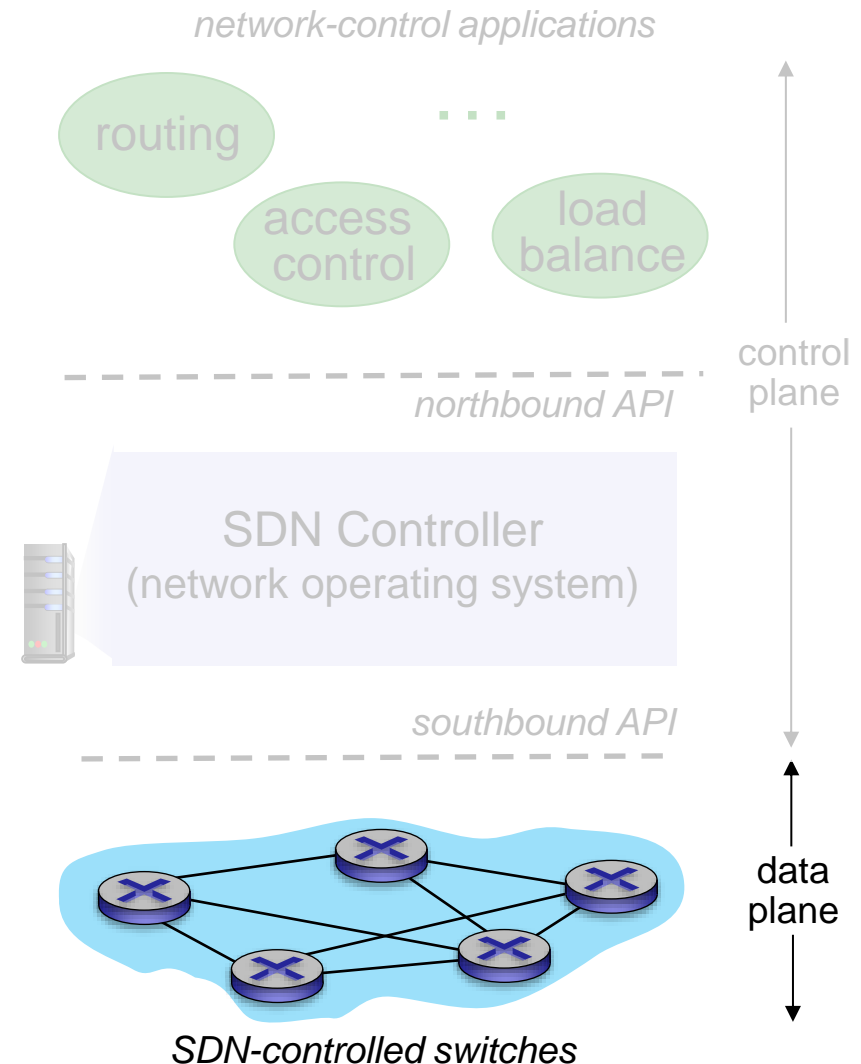
Software defined networking (SDN)



Software defined networking (SDN)

Data-plane switches:

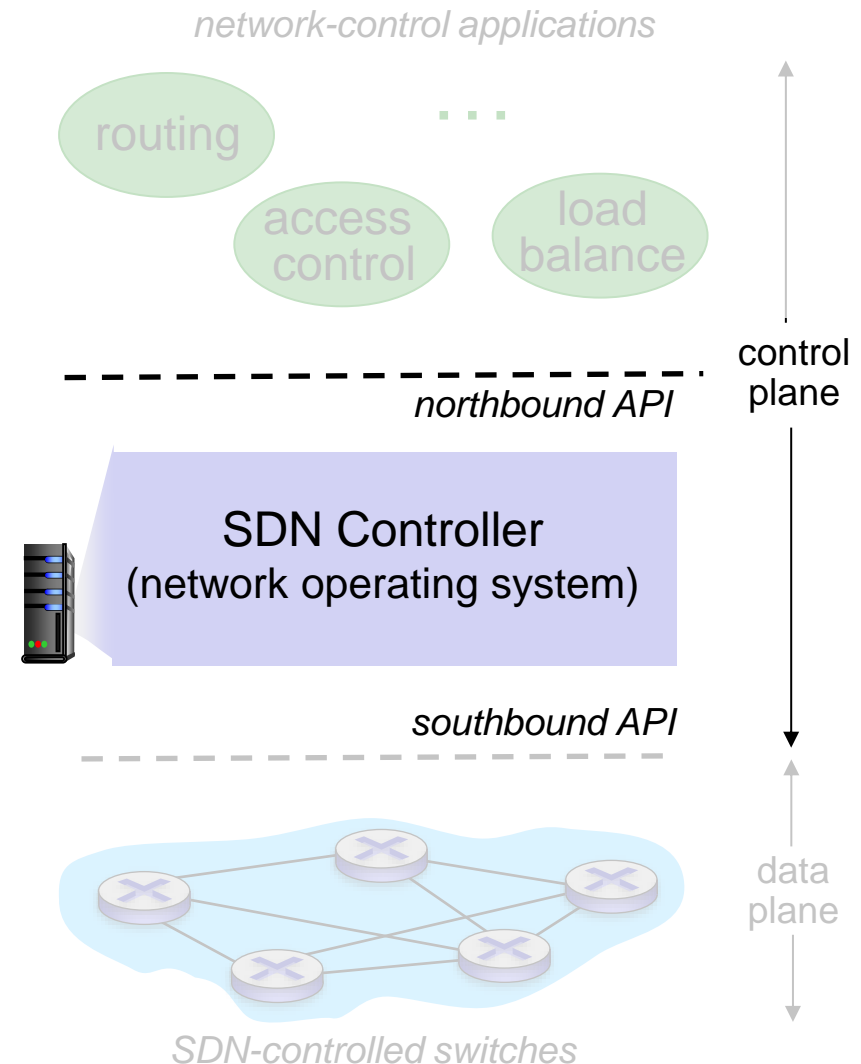
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable, what is not
- protocol for communicating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



Software defined networking (SDN)

SDN controller (network OS):

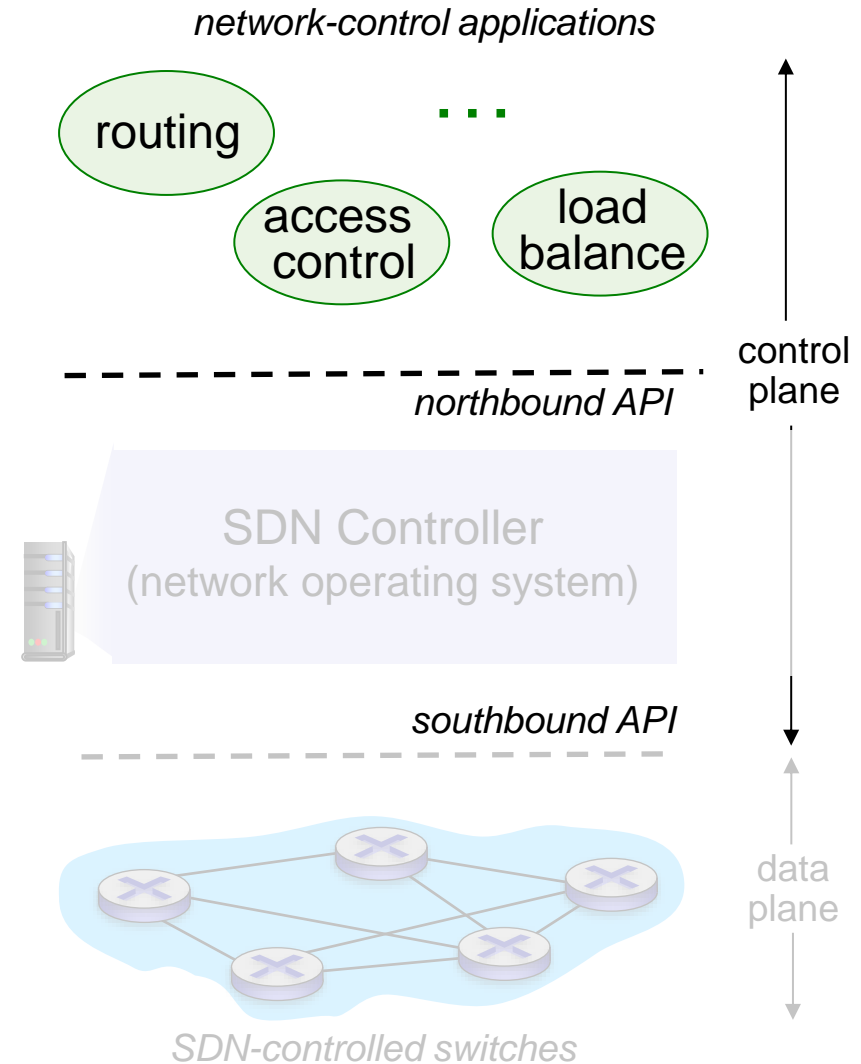
- maintain network state information
- interacts with network control applications “above” via API
- interacts with network switches “below” via API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



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



SDN Controller and SDN Network –Control Applications

- SDN control plane divides broadly into 2 components – the SDN controller and the SDN network-control applications.
- SDN Controller: 3 layers:
 - **Communication Layer:** communicating between the SDN controller and controlled network devices. **OpenFlow** is used to transfer information between the controller and switch, or a host or any other device. Additional information such as link's availability needs to be communicated, new devices added to the network, heartbeat.

SDN Controller and SDN Network –Control Applications

- **Network-wide state-management layer:** ultimate control decisions made by the SDN control plane, such as configuring flow tables in all switches, to achieve the desired end-to-end forwarding, load balancing , firewall capability. In order to execute, it requires the controller to have up-to-date information about the state of the networks's hosts, links, switches, and other devices.

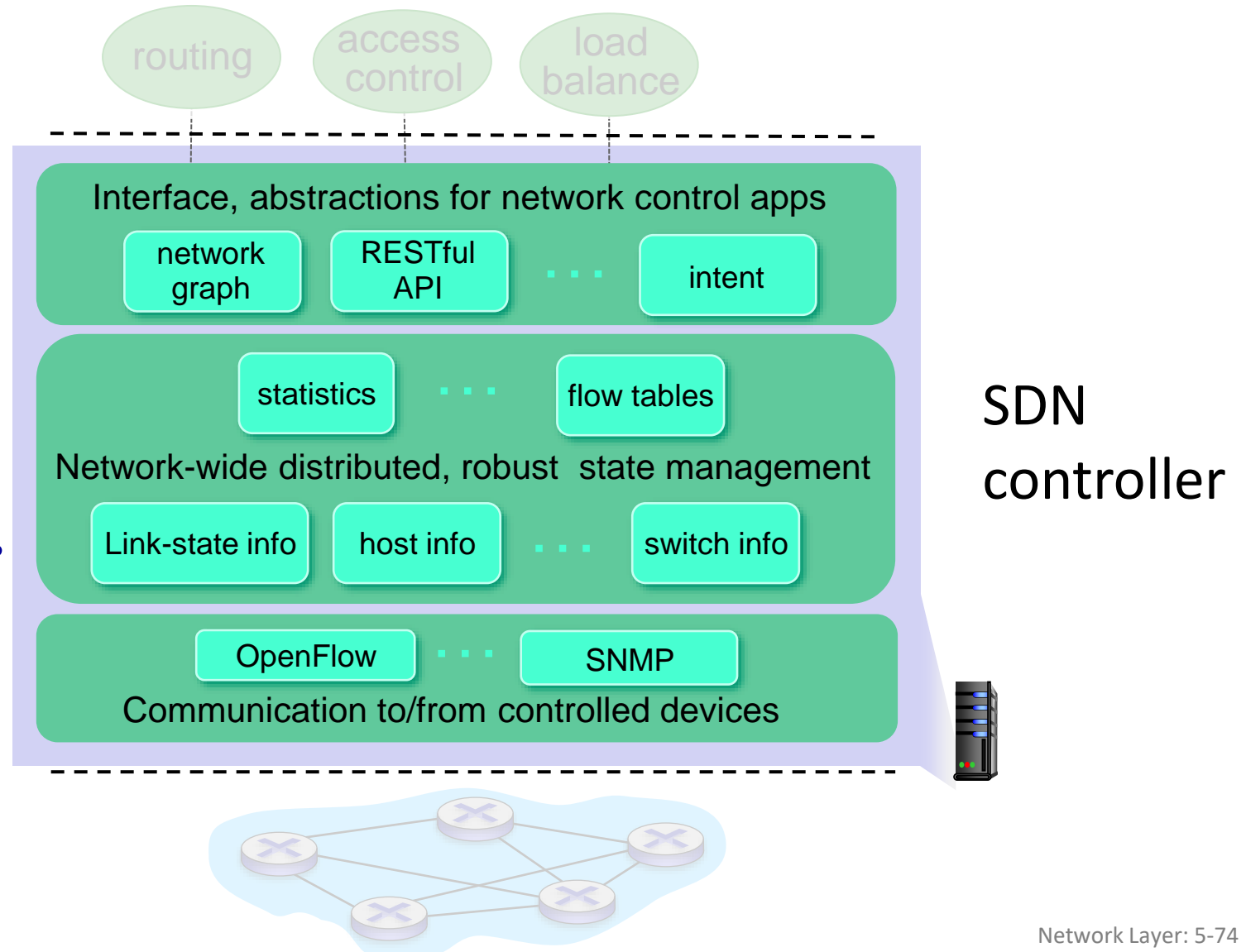
- **The interface to the network-control application layer:** the controller interacts with network-control applications through its interface. This API allows network-control applications to read/write network state and flow tables within the state-management layer. Different types of APIs may be provided. **REST** – request-response interface. **RESTful API** is an interface that two computer systems use to exchange information securely over the internet

Components of SDN controller

interface layer to network control apps: abstractions API

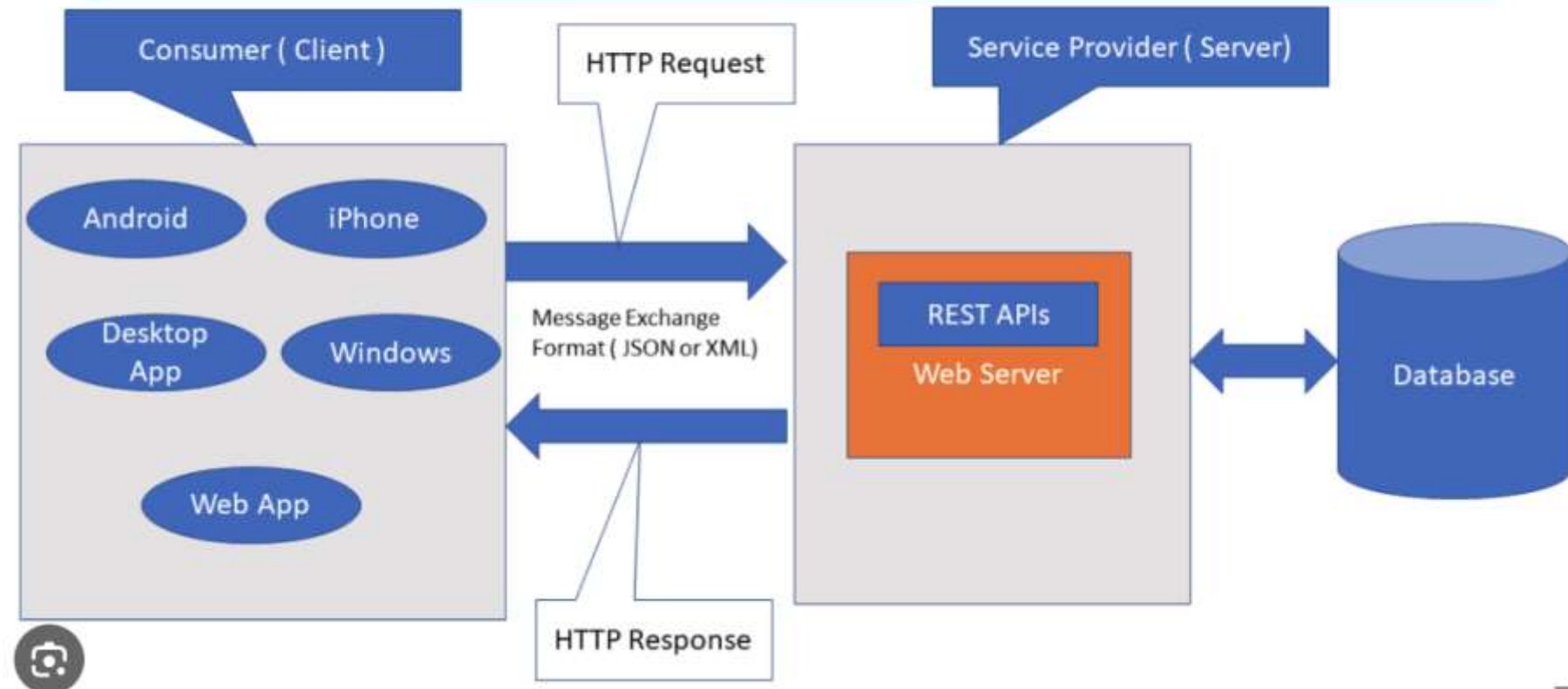
network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches



REST Architecture

REST – Architecture



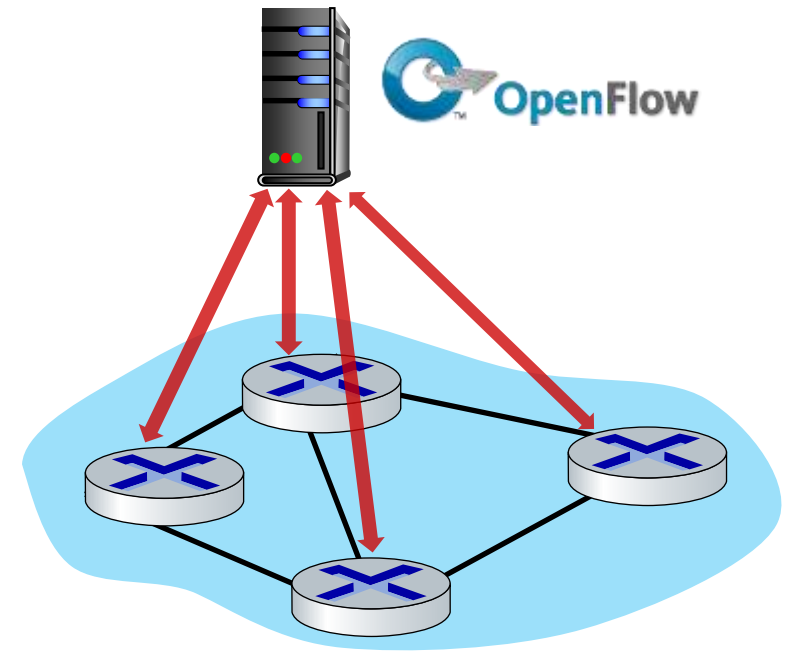
OpenFlow protocol

- operates between SDN controller, and SDN-controlled switch or other device implementing the Open-Flow API.
- **TCP** used to exchange messages; default port 6653

Important messages flowing from the controller to the controlled switch:

- **Configuration:** this message allows the controller to query and set a switch's configuration parameters
- **Modify-State:** this message is used by a controller to add/delete or modify entries in the switch's flow table, and to set switch port properties

OpenFlow Controller

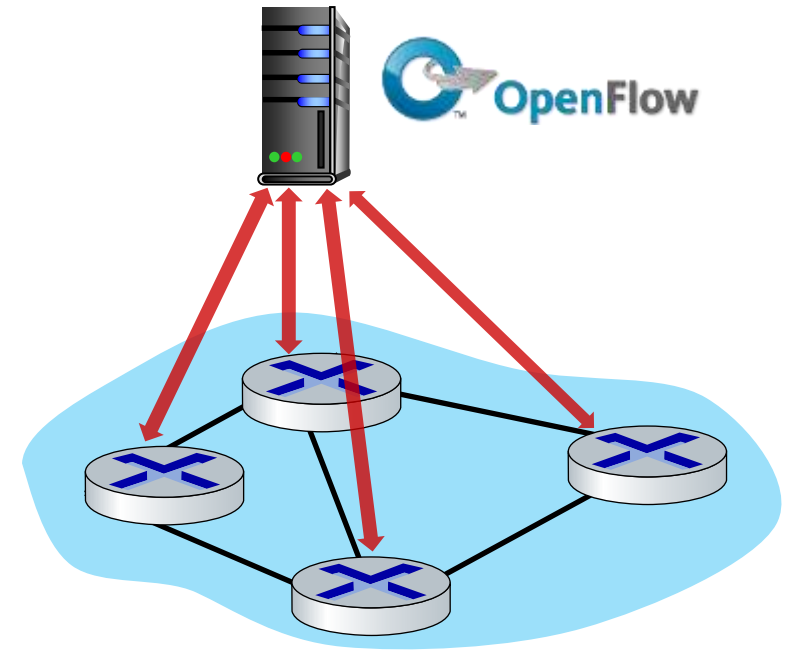


OpenFlow protocol

Important messages flowing from the controller to the controlled switch:

- **Read-State:** this message is used by a controller to collect statistics and counter values from the switch's flow table and ports
- **Send-Packet:** this message is used by the controller to send specific packet out of a specified port at the controlled switch. The message itself contains the packet to be sent in its payload.

OpenFlow Controller

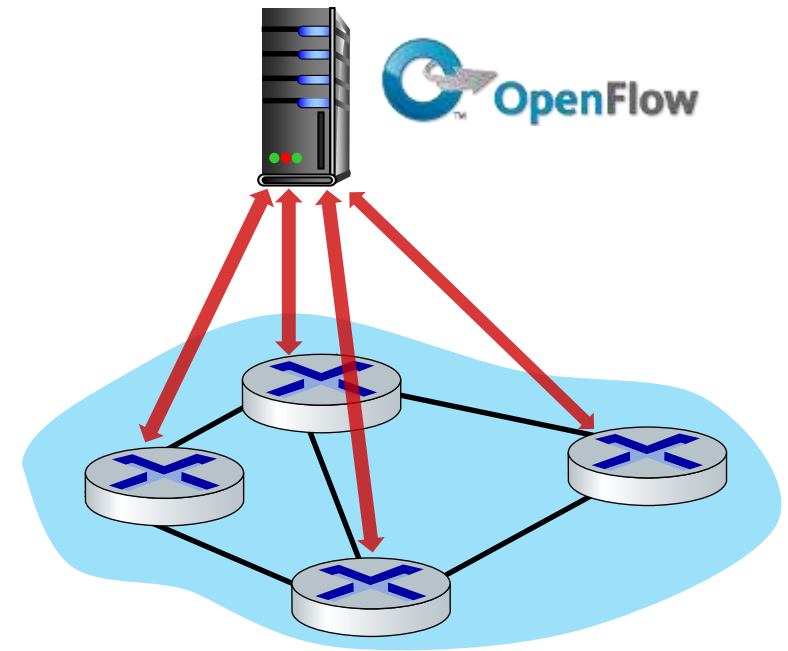


OpenFlow protocol

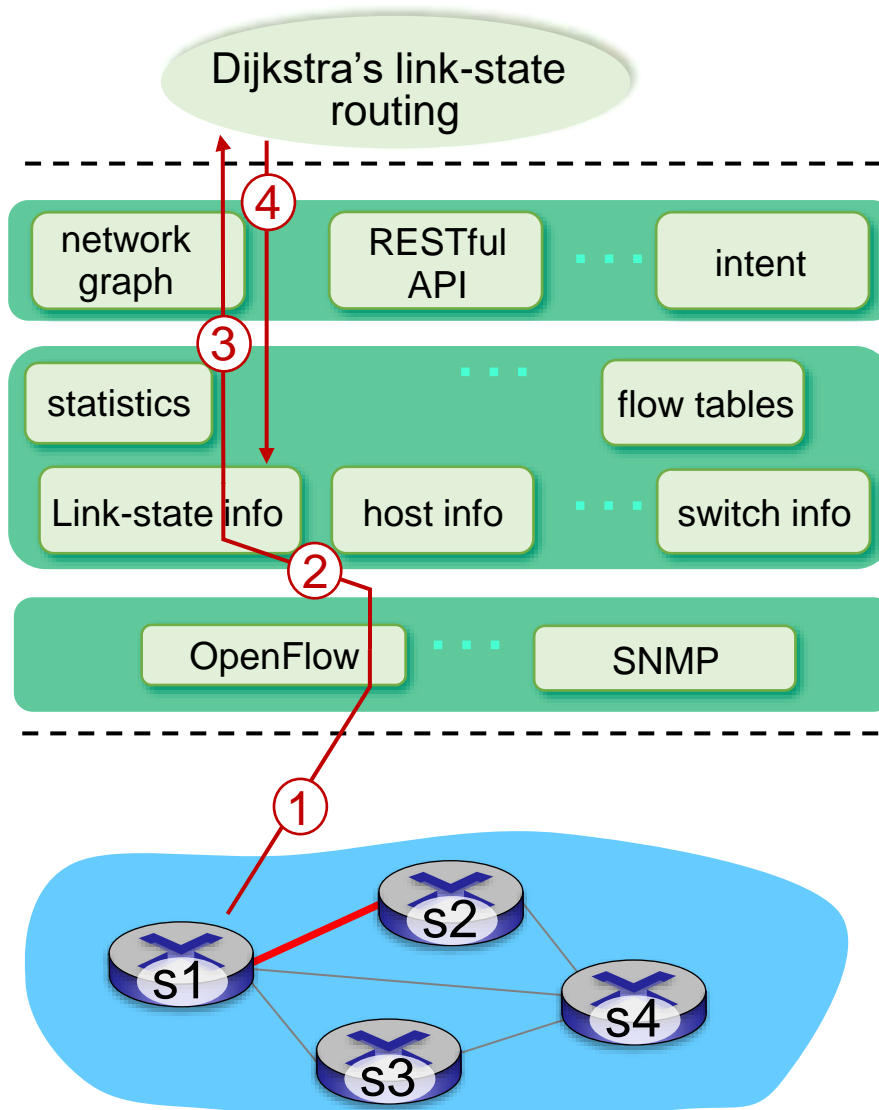
Among the messages flowing from the SDN-controlled switch to the controller are the following:

- **Flow-removed:** this message informs the controller that a flow table entry has been removed
- **Port-status:** this message is used by a switch to inform the controller of a change in port status
- **Packet-in:** the packet-in message is used to send packets to the controller.

OpenFlow Controller

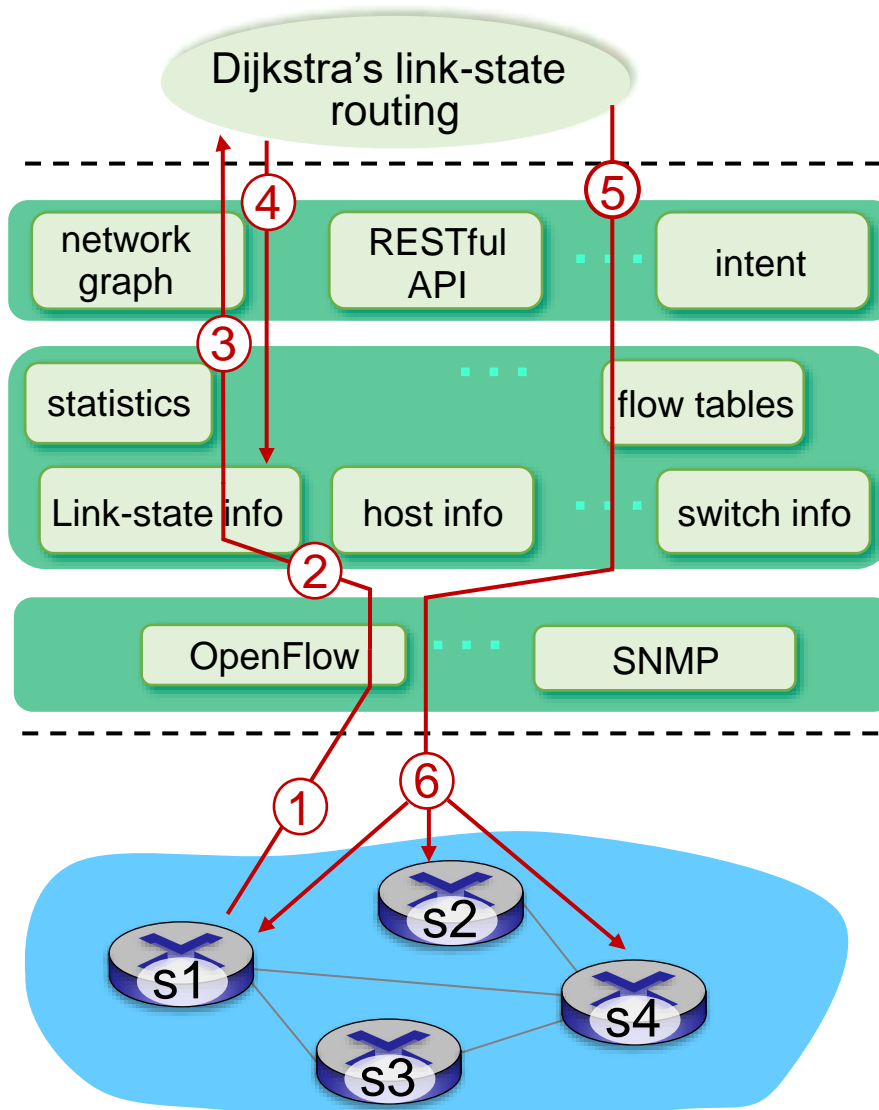


SDN: control/data plane interaction example



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

SDN: control/data plane interaction example

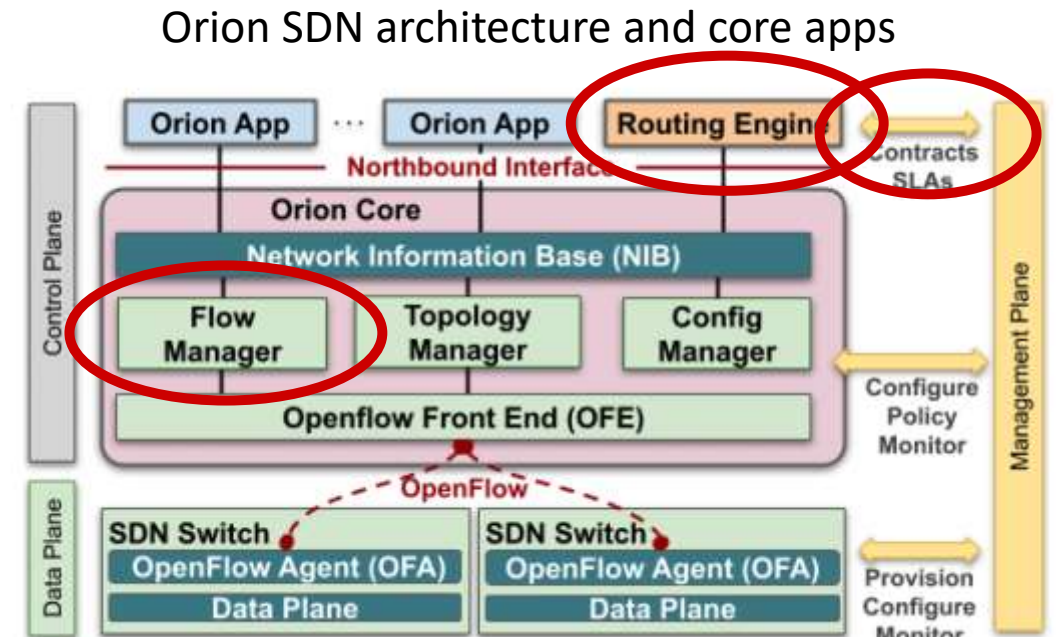


- ⑤ link state routing app interacts with flow-table manager in SDN controller, which determines the flow tables to be updated
- ⑥ The flow table manager then uses OpenFlow protocol to update flow table entries at affected switches – s1(now routes to s2 via s4).

Google ORION SDN control plane

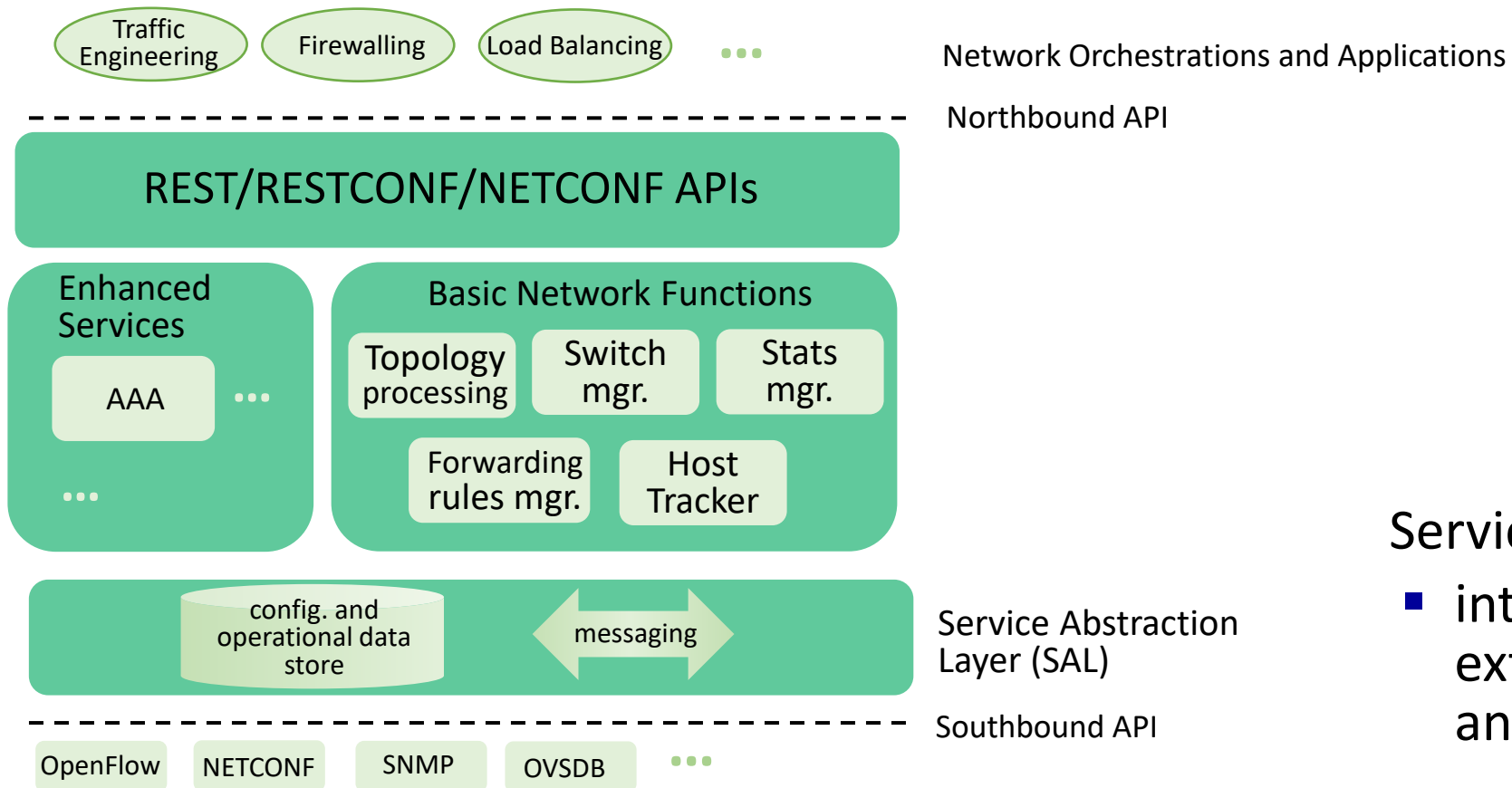
ORION: Google's SDN control plane (*NSDI'21*): control plane for Google's datacenter (Jupiter) and wide area (B4) networks

- **routing** (intradomain, iBGP), traffic engineering: implemented in *applications* on top of ORION core
- **edge-edge flow-based** controls (e.g., CoFlow scheduling) to meet contract SLAs
- **management**: pub-sub distributed microservices in Orion core, OpenFlow for switch signaling/monitoring



Note: ORION provides *intradomain* services within Google's network

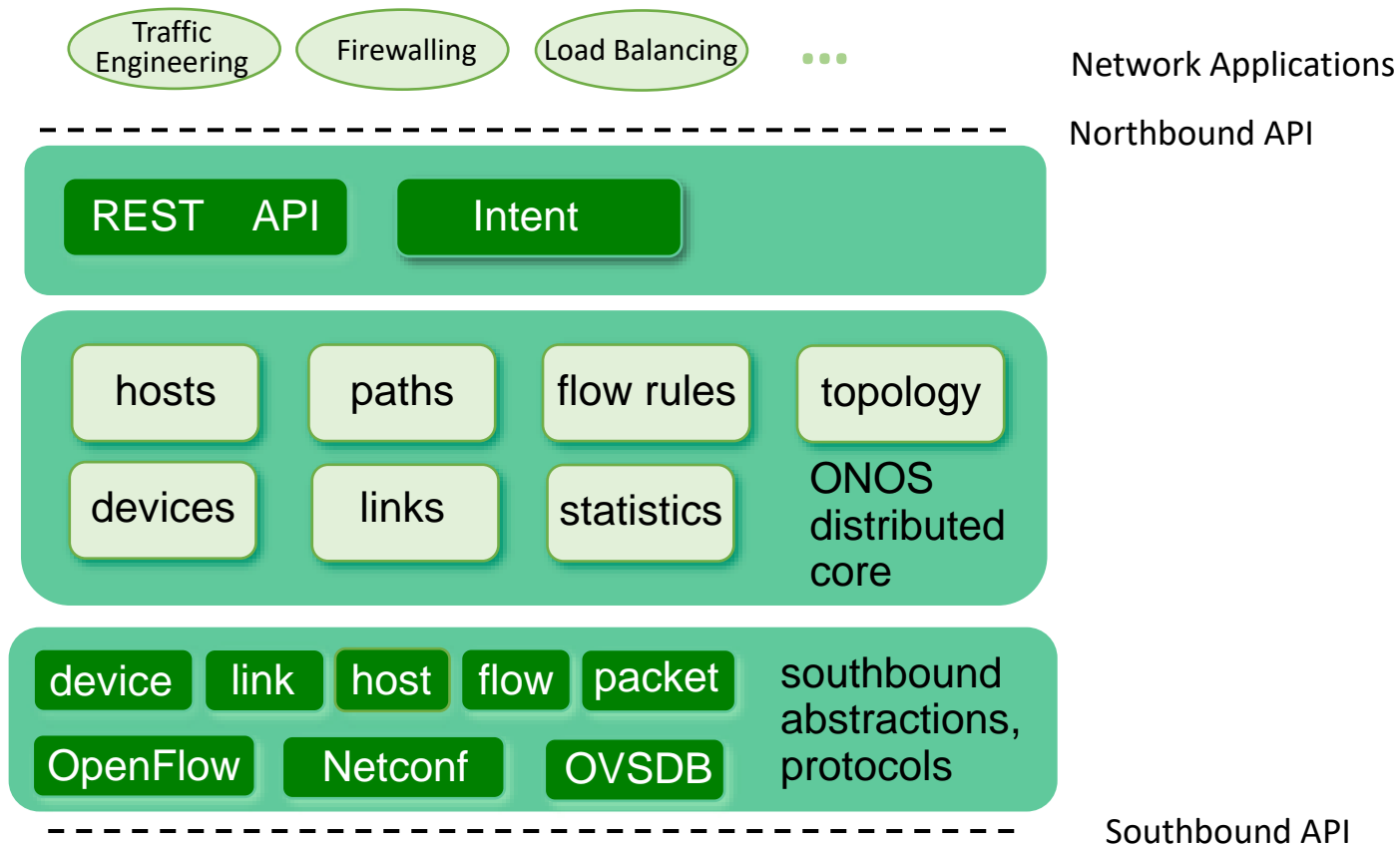
OpenDaylight (ODL) controller



Service Abstraction Layer:

- interconnects internal, external applications and services

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

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

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

Network layer: “control plane” roadmap

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



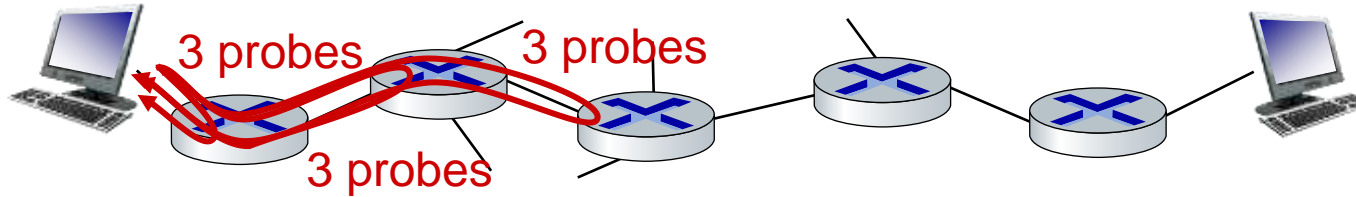
- network management, configuration
 - SNMP
 - NETCONF/YANG

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(side) IP datagrams
 - ICMP messages are carried as IP payload
- *ICMP message*: type, code plus first 8 bytes of IP datagram causing error
- Traceroute uses ICMP datagrams in conjunction with UDP segments to determine the names and address of the routers and switches between source and destination.

<u>Type</u>	<u>Code</u>	<u>description</u>
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

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 n th 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: “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

What is network management?

- autonomous systems (aka “network”): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, configuration, control



"**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."

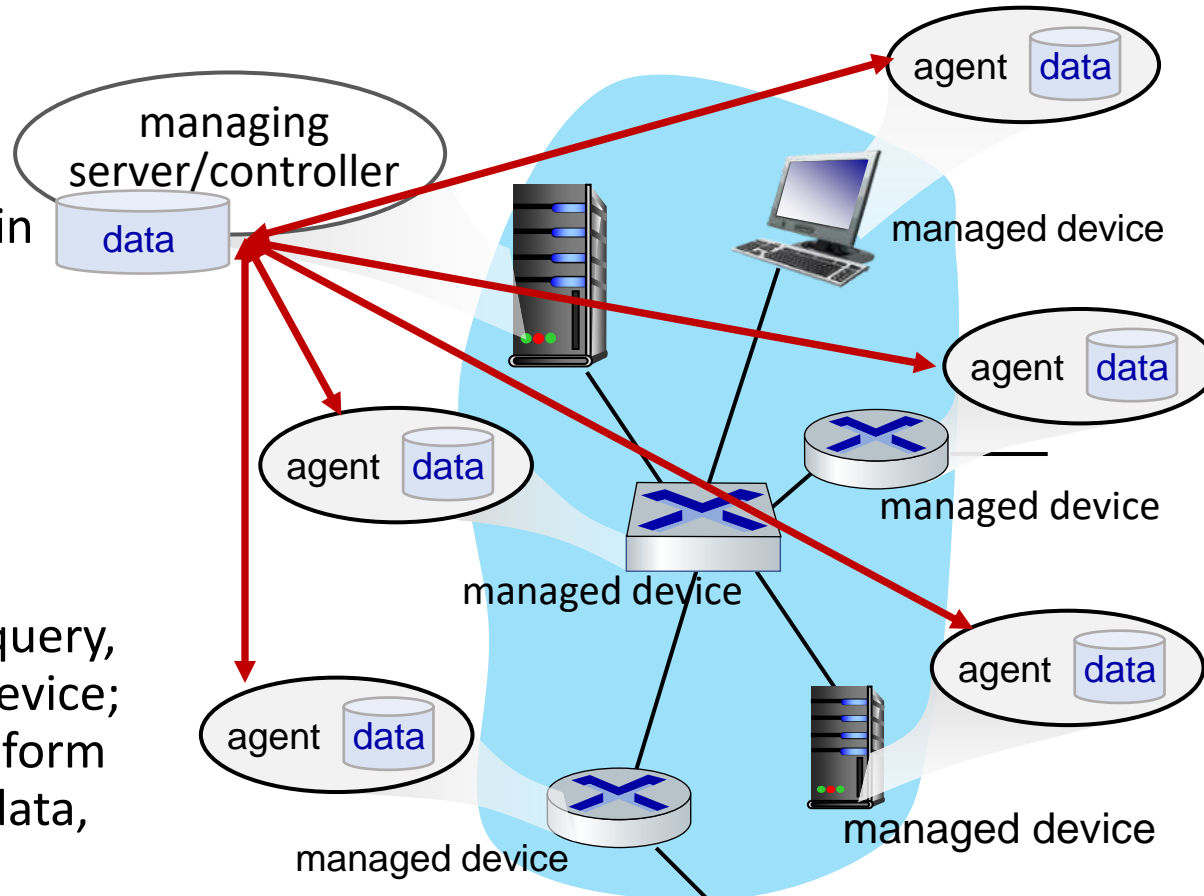
Components of network management

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 operator approaches to management

CLI (Command Line Interface)

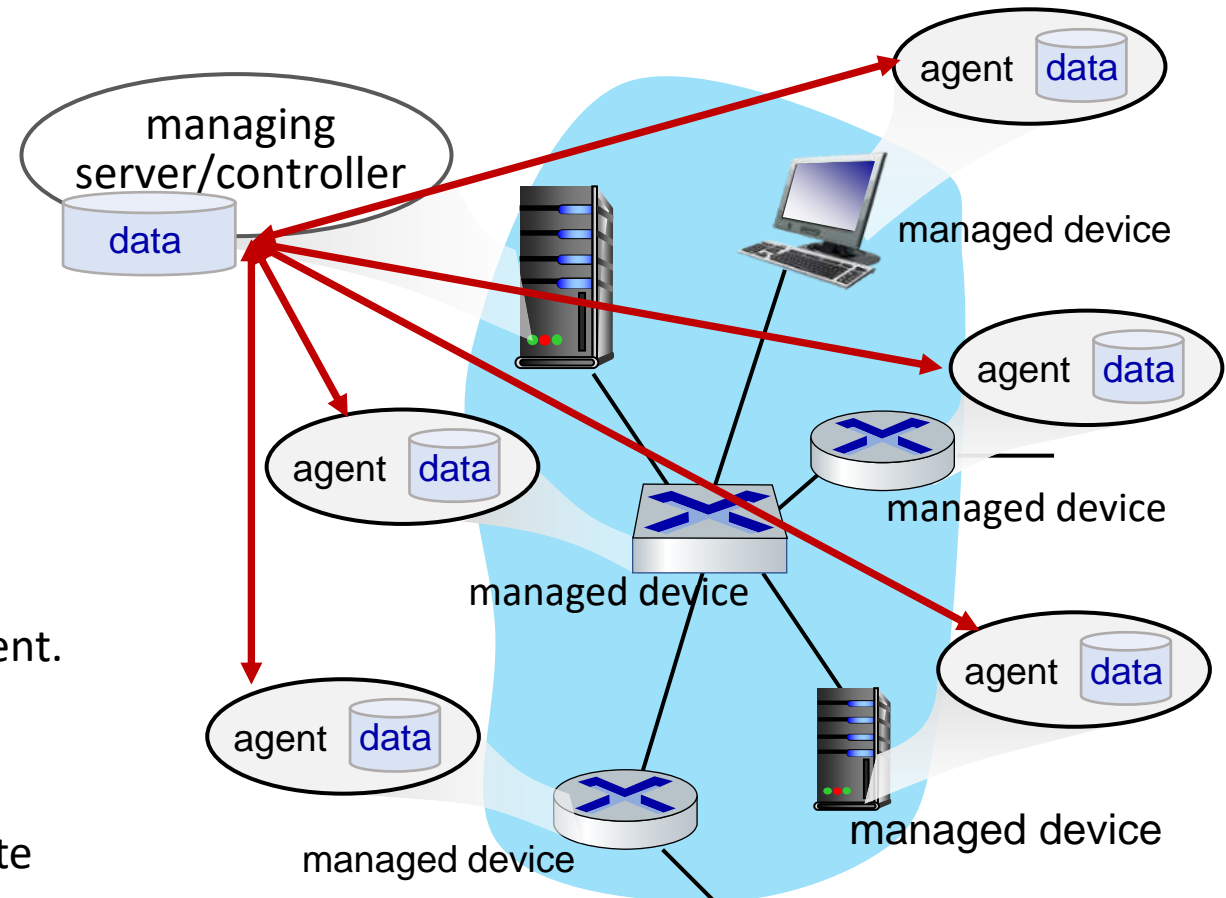
- operator issues (types, scripts) direct to individual devices (e.g., via ssh)
- Not scalable across multiple devices

SNMP/MIB

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

NETCONF/YANG

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



SNMP

- **SNMP v3**

Application layer protocol used to convey network-management control and information messages between a managing server and an agent executing on behalf of that managing server.

- Typically, a request will be used to query (retrieve) or modify (set) MIB object values associated with a managed device.
- A second common usage of SNMP is for an agent to send an unsolicited message, known as a **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.

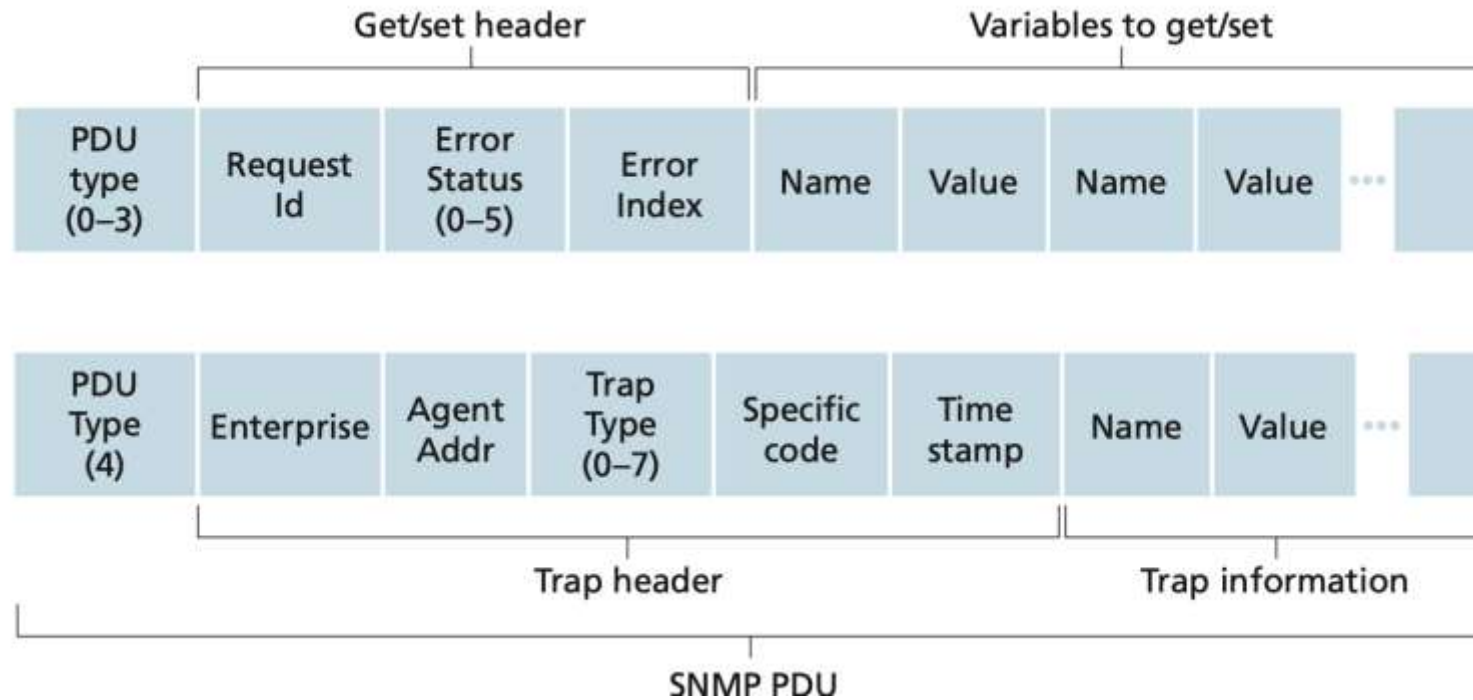
- SNMP v3 defines seven types of messages. Known as protocol data units (PDUs).

SNMP protocol: message types

SNMPv3 PDU Type	Sender-receiver	Description
GetRequest	manager-to-agent	get value of one or more MIB object instances
GetNextRequest	manager-to-agent	get value of next MIB object instance in list or table
GetBulkRequest	manager-to-agent	get values in large block of data, for example, values in a large table
InformRequest	manager-to-manager	inform remote managing entity of MIB values remote to its access
SetRequest	manager-to-agent	set value of one or more MIB object instances
Response	agent-to-manager or manager-to-manager	generated in response to GetRequest, GetNextRequest, GetBulkRequest, SetRequest PDU, or InformRequest
SNMPv2-Trap	agent-to-manager	inform manager of an exceptional event #

Table 5.2 ♦ SNMPv3 PDU types

SNMP protocol: message formats



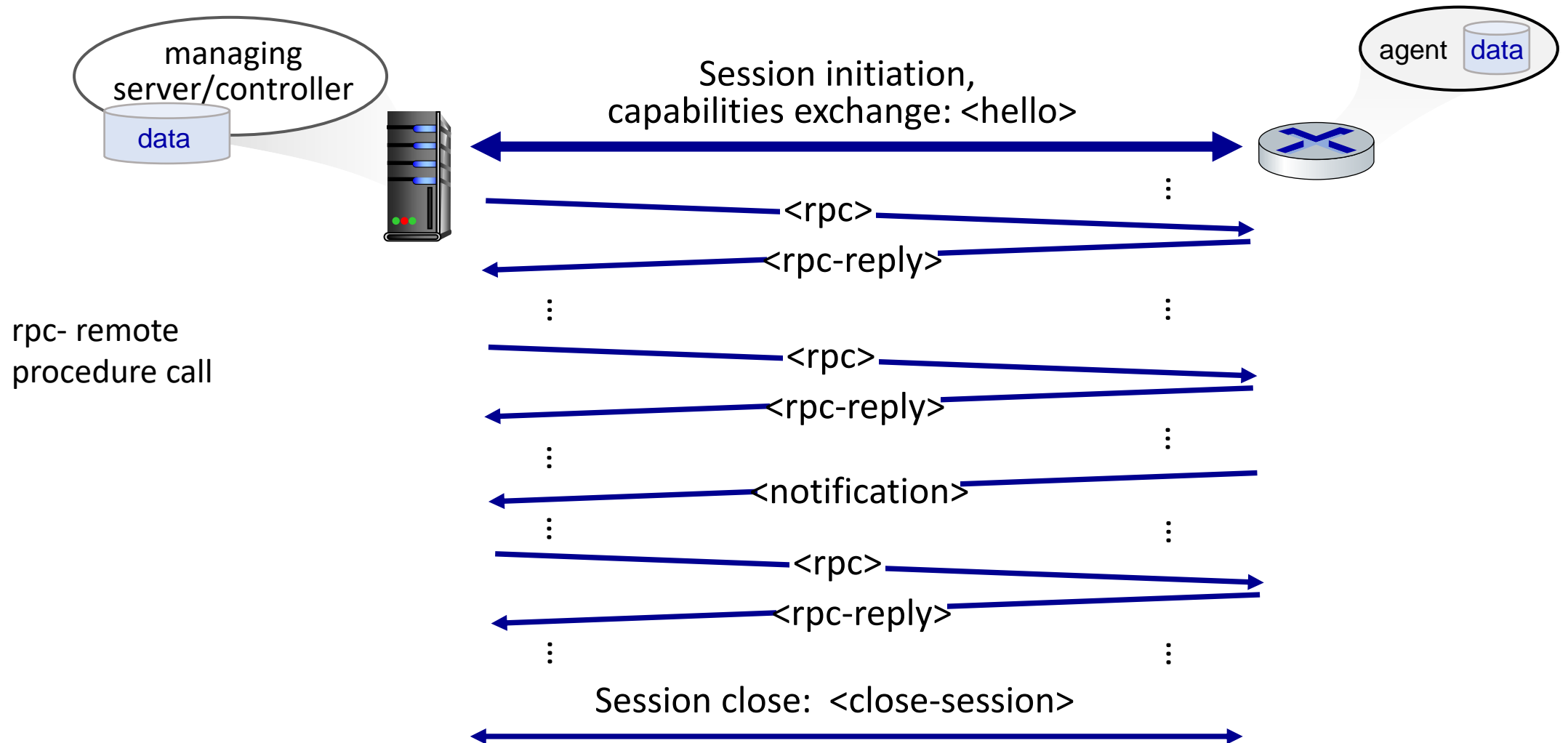
SNMP: Management Information Base (MIB)

- An MIB object might be a counter, such as the number of IP datagrams discarded at a router due to errors in an IP datagram header; or the number of carrier sense errors in an Ethernet interface card; descriptive information such as the version of the software running on a DNS server; status information such as whether a particular device is functioning correctly; or protocol-specific information such as a routing path to a destination.
- There are over 400 MIB modules defined in various IETC RFC's

NETCONF overview

- **goal:** actively manage/**configure** devices network-wide
- operates between managing server and managed network devices
 - actions: retrieve, set, modify, activate configurations
 - 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

NETCONF initialization, exchange, close

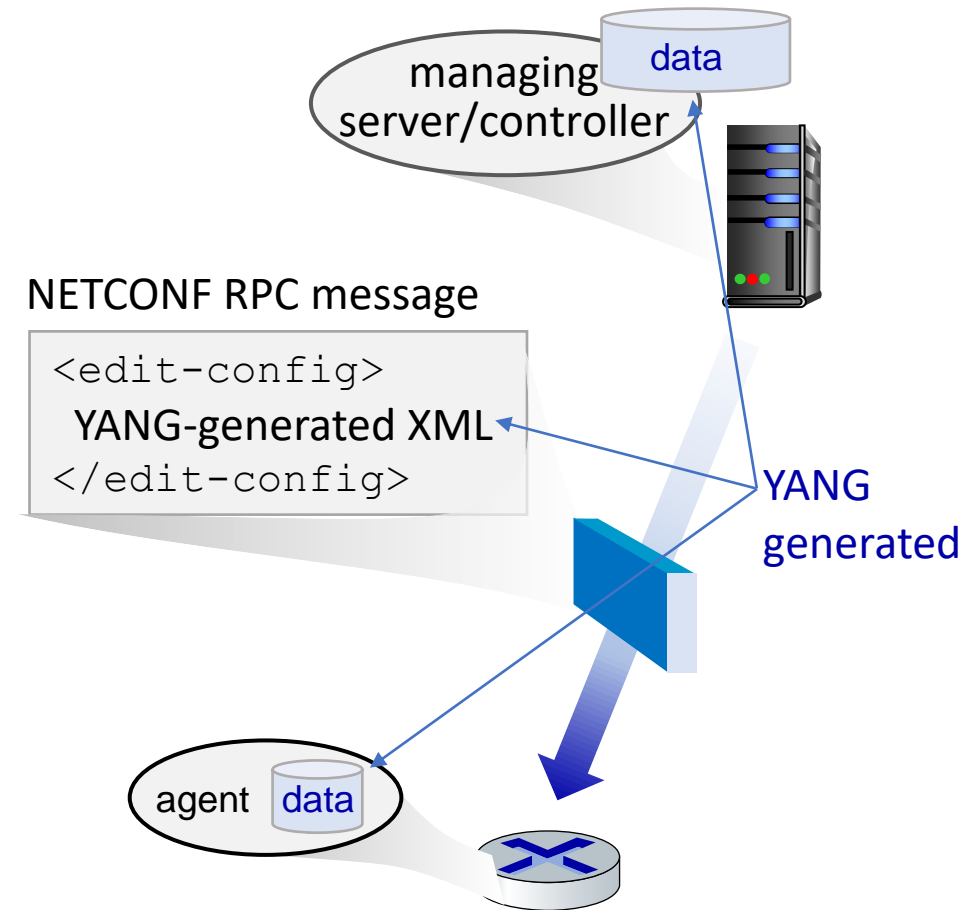


Selected NETCONF Operations

NETCONF Operation	Description
<get-config>	Retrieve all or part of a given configuration. A device may have multiple configurations. There is always a <code>running/</code> configuration that describes the 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 the managed device. If the <code>running/configuration</code> is specified, then the device's current (running) configuration will be changed. If the managed device was able to satisfy the request, an <code><rpc-reply></code> is sent containing an <code><ok></code> element; otherwise <code><rpc-error></code> response is returned. On error, the device's configuration state can be roll-ed-back to its previous state.
<lock>, <unlock>	The <code><lock></code> (<code><unlock></code>) operation allows the managing server to lock (unlock) the entire configuration datastore system of a managed device. Locks are intended to be short-lived and allow a client 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 <code><notification></code> for specified events of interest from the managed device to the managing server, until the subscription is terminated.

YANG

- YANG is the data modeling language used to precisely specify the structure, syntax, and semantics of network management data used by NETCONF,
- All YANG definitions are contained in modules, and an 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 and also allows data modelers to express constraints that must be satisfied by a valid NETCONF configuration—a powerful aid in helping ensure that NETCONF configurations satisfy specified correctness and consistency constraints.
- YANG is also used to specify NETCONF notifications.



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!

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

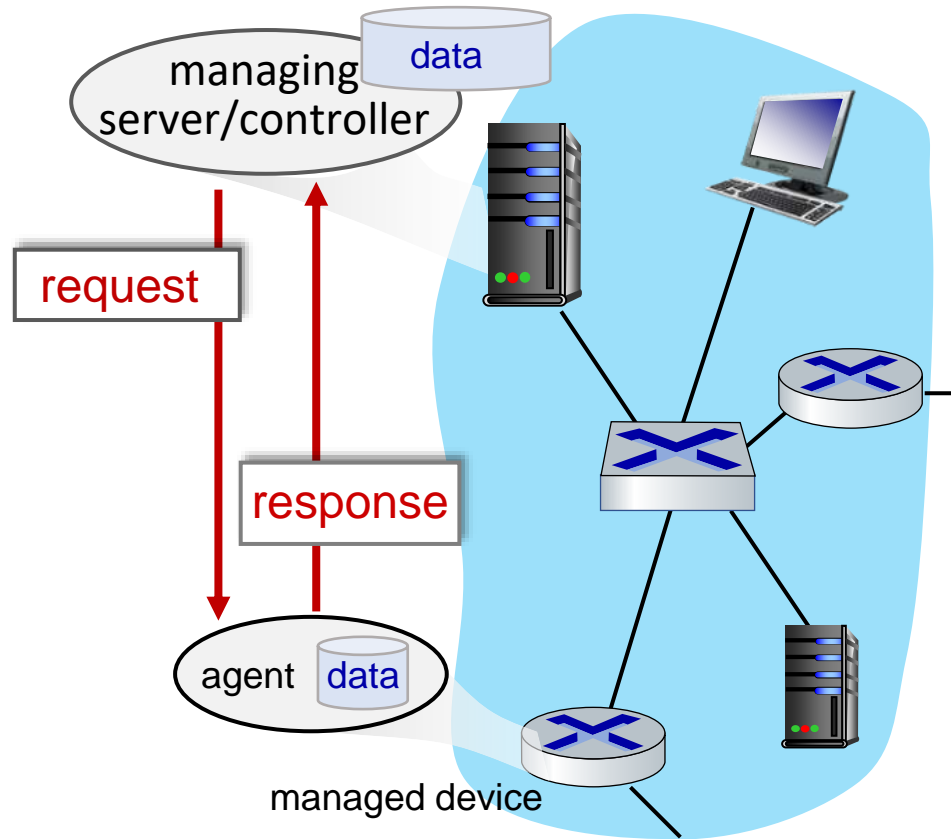


- network management, configuration
 - SNMP
 - NETCONF/YANG

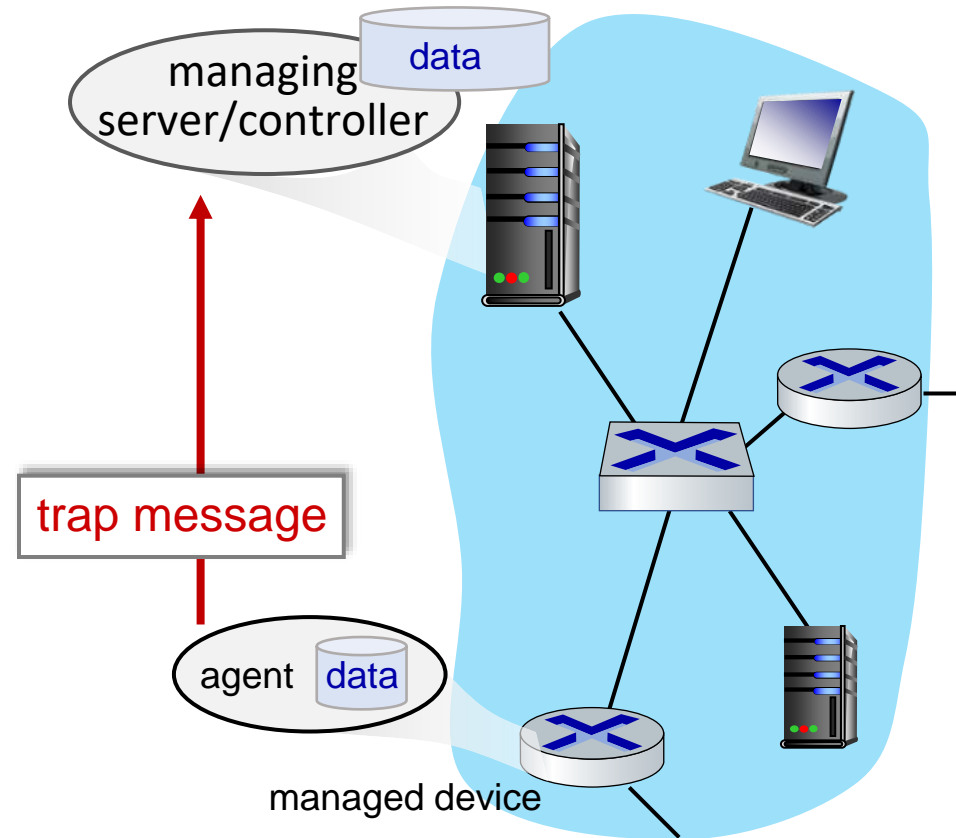
Additional Chapter 5 slides

SNMP protocol

Two ways to convey MIB info, commands:



request/response mode

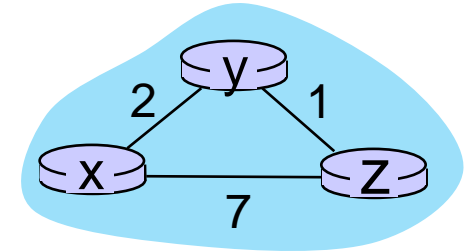
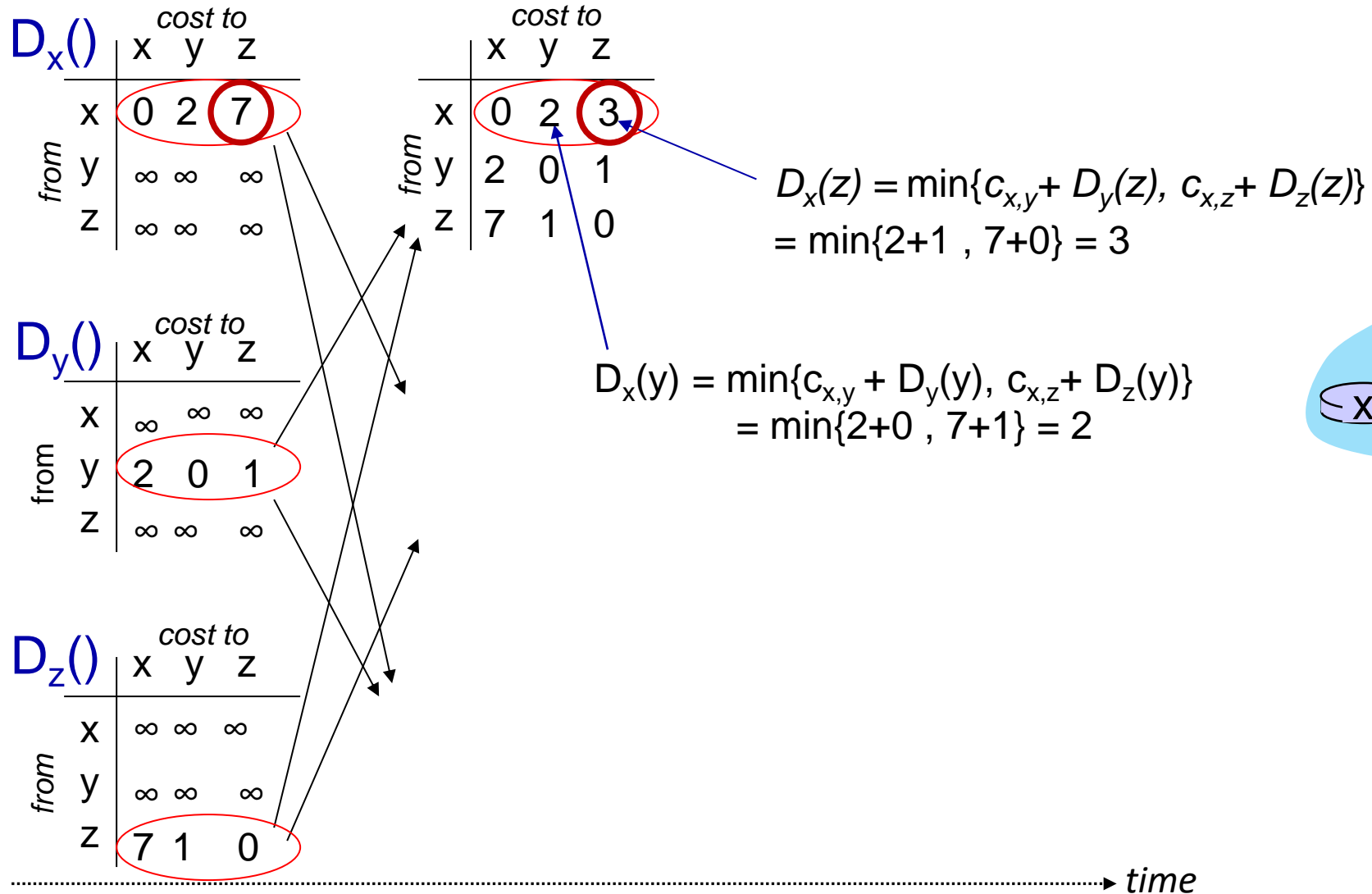


trap mode

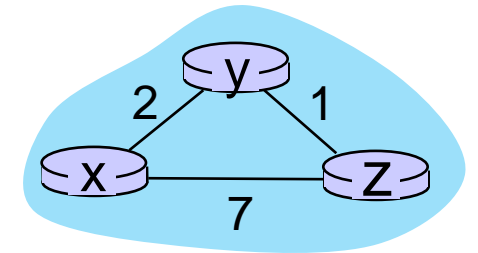
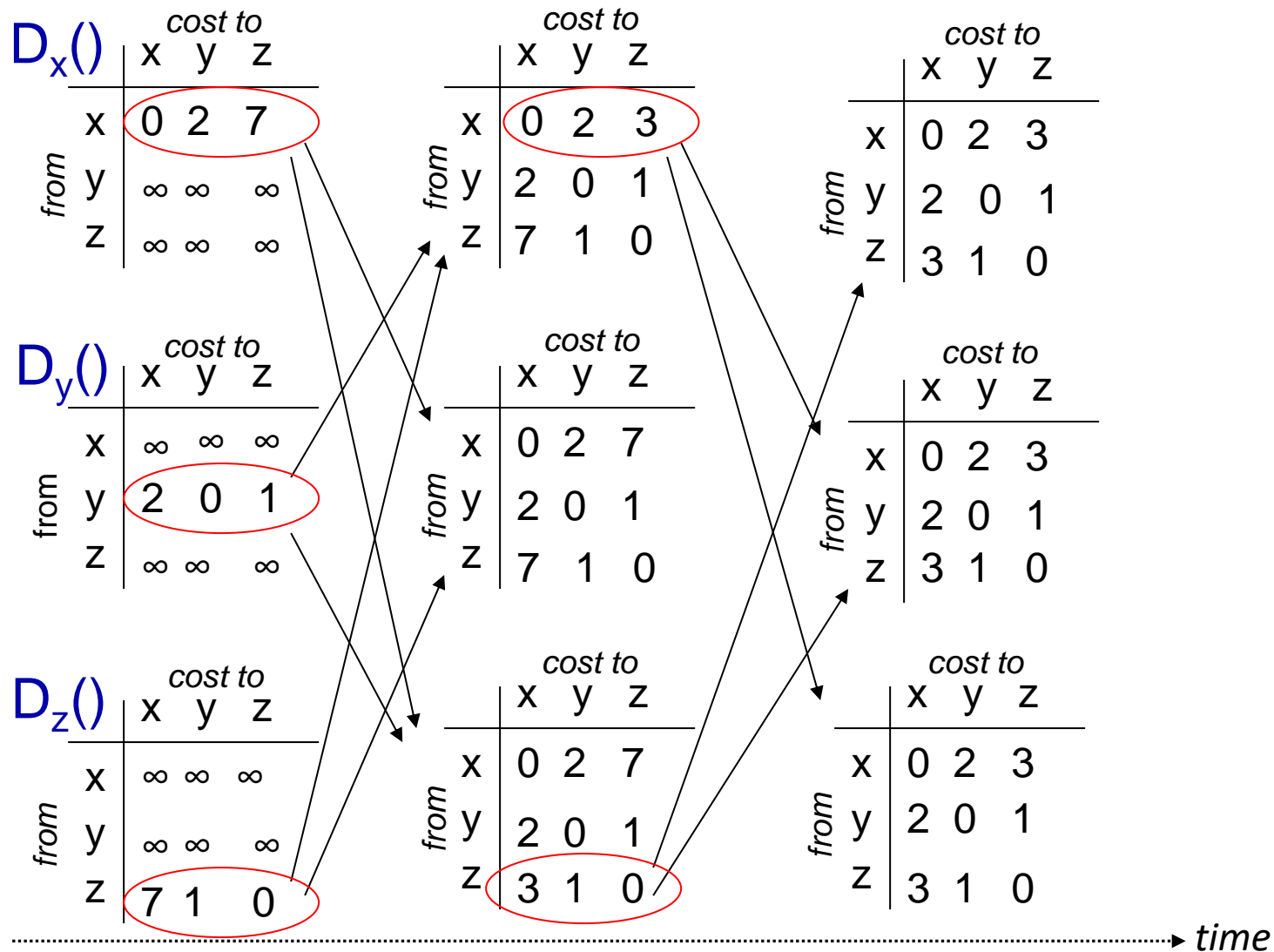
Sample NETCONF RPC message

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

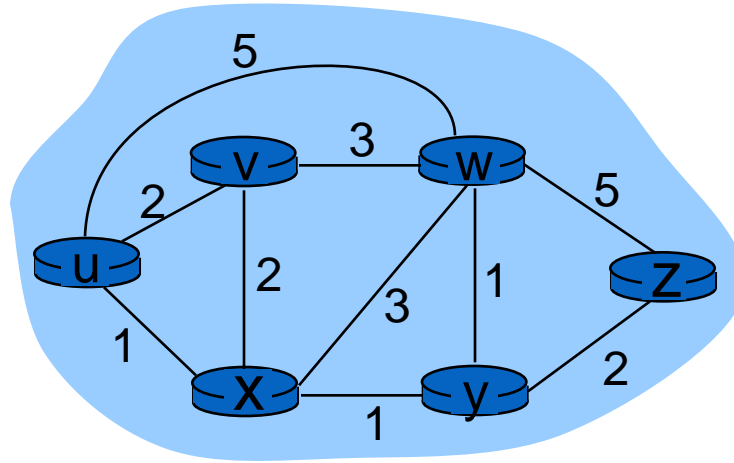
Distance vector: another example



Distance vector: another example



Graph abstraction

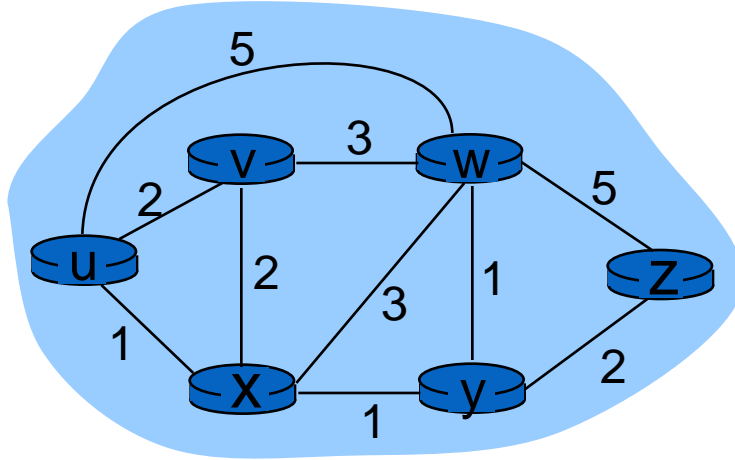


graph: $G = (N, E)$

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

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

Graph abstraction: costs



$c(x, x') = \text{cost of link } (x, x')$
e.g., $c(w, z) = 5$

cost could always be 1, or
inversely related to
bandwidth, or
related to congestion

cost of path $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Global vs Decentralized Routing Algorithm

Global algorithm

- Has complete information about connectivity and link costs.
- Often referred to as link-state (LS) algorithms.

Decentralized algorithm

- The calculation of the least-cost path is carried out in an iterative, distributed manner.
- No node has complete information about the costs of all network links.
- Each node begins with only the knowledge of the costs of its own directly attached links.
- Distance-vector (DV) algorithm.

Routing algorithm classification

Static vs Dynamic Routing Algorithm

Static algorithm

- Routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a router's forwarding table)

Dynamic algorithm

- Changes the routing paths as the network traffic loads or topology change.

Routing algorithm classification

Load-sensitive vs Load-insensitive Routing Algorithm

Load-sensitive algorithm

- Link costs vary dynamically to reflect the current level of congestion in the underlying link.
- If currently link is congested, a routing algorithm will tend to choose other routes.

Load-insensitive algorithm

- Algorithm is not impacted from link congestion.

Global Routing: Link-State Routing Algorithm

A Link-State Routing Algorithm

Dijkstra's algorithm

- net 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 dest.'s

notation:

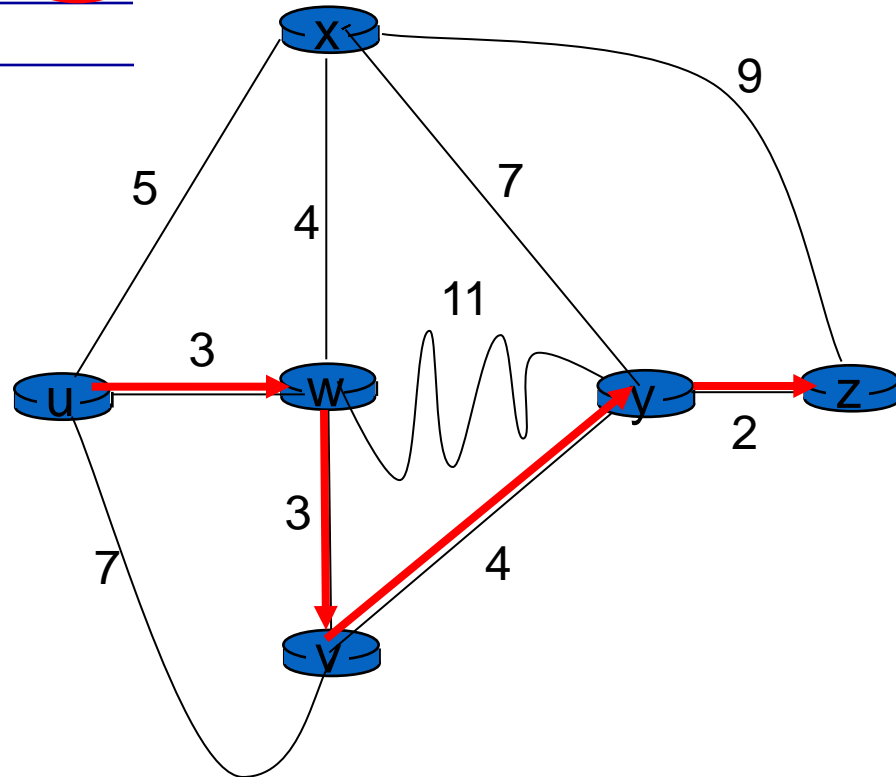
- $c(x,y)$: link cost from node x to y ;
 $= \infty$ if not direct neighbors
- $D(v)$: current value of cost of path from source to dest. v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least cost path definitively known

Dijkstra's algorithm: 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	14,w	∞
2	uwx	6,w			12,x	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist
 - ❖ can be broken arbitrarily



Dijkstra's Algorithm

1 **Initialization:**

2 $N' = \{u\}$

3 for all nodes v

4 if v adjacent to u

5 then $D(v) = c(u,v)$

6 else $D(v) = \infty$

7

8 **Loop**

9 find w not in N' such that $D(w)$ is a minimum

10 add w to N'

11 update $D(v)$ for all v adjacent to w and not in N' :

12 **$D(v) = \min(D(v), D(w) + c(w,v))$**

13 /* new cost to v is either old cost to v or known

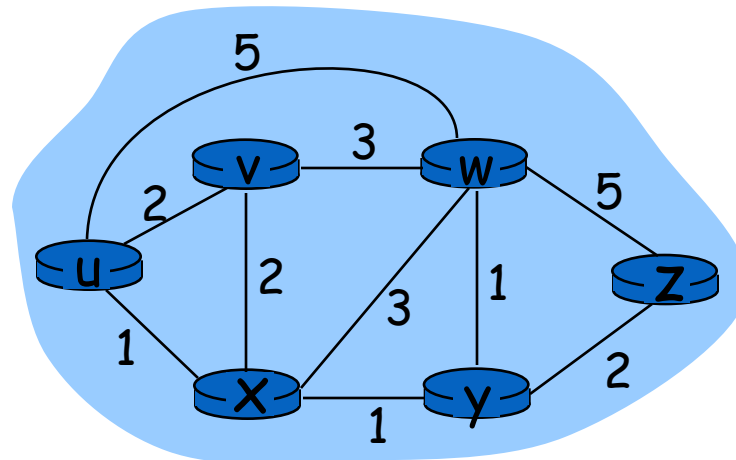
14 shortest path cost to w plus cost from w to v */

15 **until all nodes in N'**



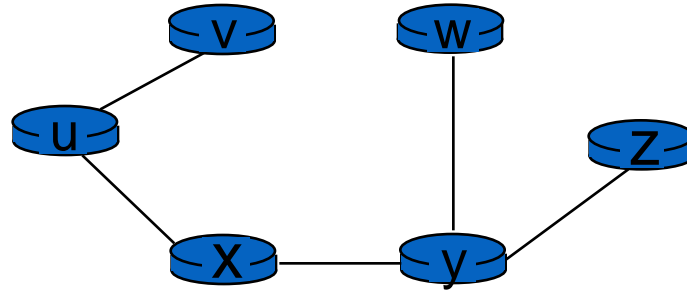
Dijkstra's algorithm: example

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					



Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



resulting forwarding table in u:

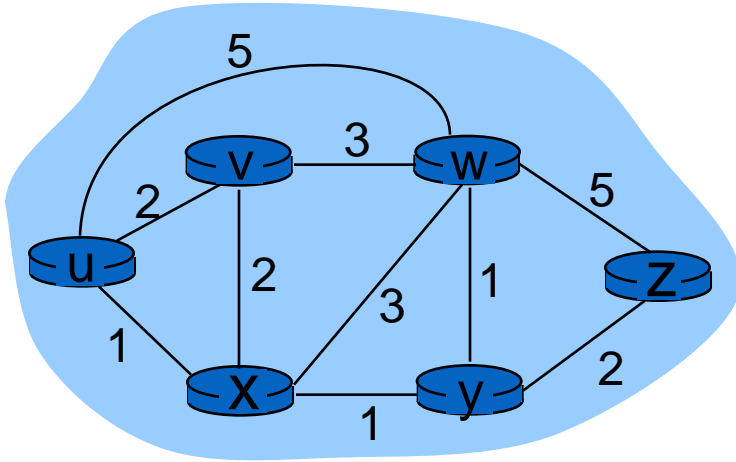
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Decentralized Routing: Distance Vector Routing Algorithm

Distance vector algorithm

- The distance-vector (DV) algorithm is iterative, asynchronous, and distributed.
- **Distributed:** Each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results back to its neighbors.
- **Iterative:** the process continues on until no more information is exchanged between neighbors.
- **Asynchronous:** does not require all of the nodes to operate in lockstep with each other.

Bellman-Ford example



$$d_v(z) = 5, d_x(z) = 3, d_w(z) = 3$$

B-F equation says:

$$\begin{aligned} d_u(z) &= \min \{ c(u,v) + d_v(z), \\ &\quad c(u,x) + d_x(z), \\ &\quad c(u,w) + d_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

**node x
table**

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

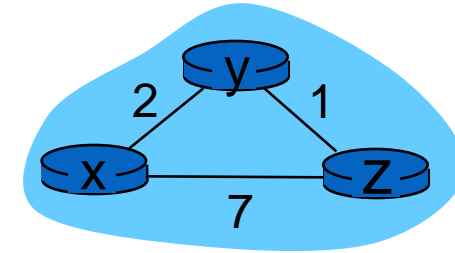
**node y
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

**node z
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

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



time

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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

node x
table

	cost to		
	x	y	z
from x	0	2	7
from y	∞	∞	∞
from z	∞	∞	∞

node y
table

	cost to		
	x	y	z
from x	∞	∞	∞
from y	2	0	1
from z	∞	∞	∞

node z
table

	cost to		
	x	y	z
from x	∞	∞	∞
from y	∞	∞	∞
from z	7	1	0

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

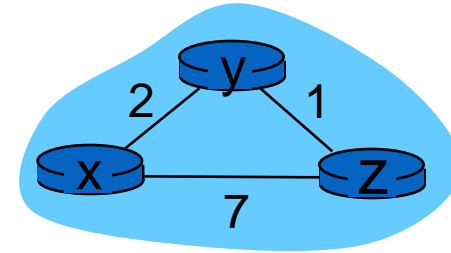
	cost to		
	x	y	z
from x	0	2	7
from y	2	0	1
from z	7	1	0

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

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

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

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



time

Distance vector algorithm

iterative, asynchronous:

- each local iteration caused by:
 - ❖ local link cost change
 - ❖ DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary

each node:

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

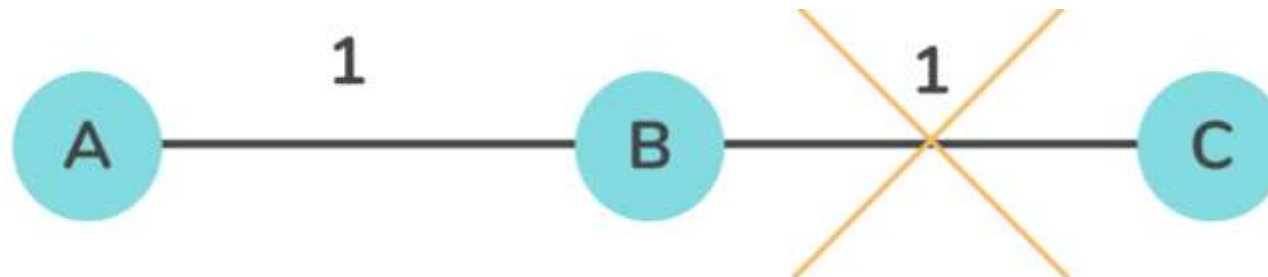
recompute estimates

if DV to any dest has changed, *notify* neighbors

Distance vector algorithm

Count to Infinity problem

- The connection between B and C gets disconnected.
- B will know that it cannot get to C at a cost of 1 anymore and update its table accordingly. However, it can be possible that A sends some information to B that it is possible to reach C from A at a cost of 2.
- Then, since B can reach A at a cost of 1, B will erroneously update its table that it can reach C via A at a cost of $1 + 2 = 3$ units.
- A will then receive updates from B and update its costs to 4, and so on. Thus, the process enters into a loop of bad feedback and the cost shoots towards infinity



A Comparison of LS and DV Routing Algorithms

Message complexity

- LS requires each node to know the cost of each link in the network. This requires $O(|N| |E|)$ messages to be sent.
- Also, whenever a link cost changes, the new link cost must be sent to all nodes.
- The DV algorithm requires message exchanges between directly connected neighbors at each iteration.

A Comparison of LS and DV Routing Algorithms

Speed of convergence

- LS is an $O(|N|^2)$ algorithm requiring $O(|N| \cdot |E|)$ messages. The DV algorithm can converge slowly and also suffers from the count-to-infinity problem.

Robustness

- In case of router fails or misbehaves.
- Under 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. But an LS node is computing only its **own forwarding tables**; other nodes are performing similar calculations for themselves, which provides a degree of **robustness**.
- Under DV, a node can advertise incorrect least-cost paths to any or all destinations, which can be diffused through the entire network under DV.

Hierarchical routing

Homogenous set of routers all executing the same routing algorithm is a bit **simplistic**.

Scale: As the number of routers becomes large, the overhead involved in computing, storing, and communicating routing information (for example, LS updates or least-cost path changes) becomes expensive.

Administrative autonomy: Ideally, an organization should be able to run and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of the problems can be solved by organizing routers into **autonomous systems** (ASs),

Internet approach to scalable routing

- Each AS consisting of a group of routers that are typically under the same administrative control (e.g., operated by the same ISP or belonging to the same company network).
- Routers within the same AS all run the same routing algorithm and have information about each other
- The routing algorithm running within an autonomous system is called an intra-autonomous system routing protocol.
- Routers connecting AS to other ASes are called gateway routers.

Internet approach to scalable routing

aggregate routers into regions known as “**autonomous systems**” (AS) (a.k.a. “domains”)

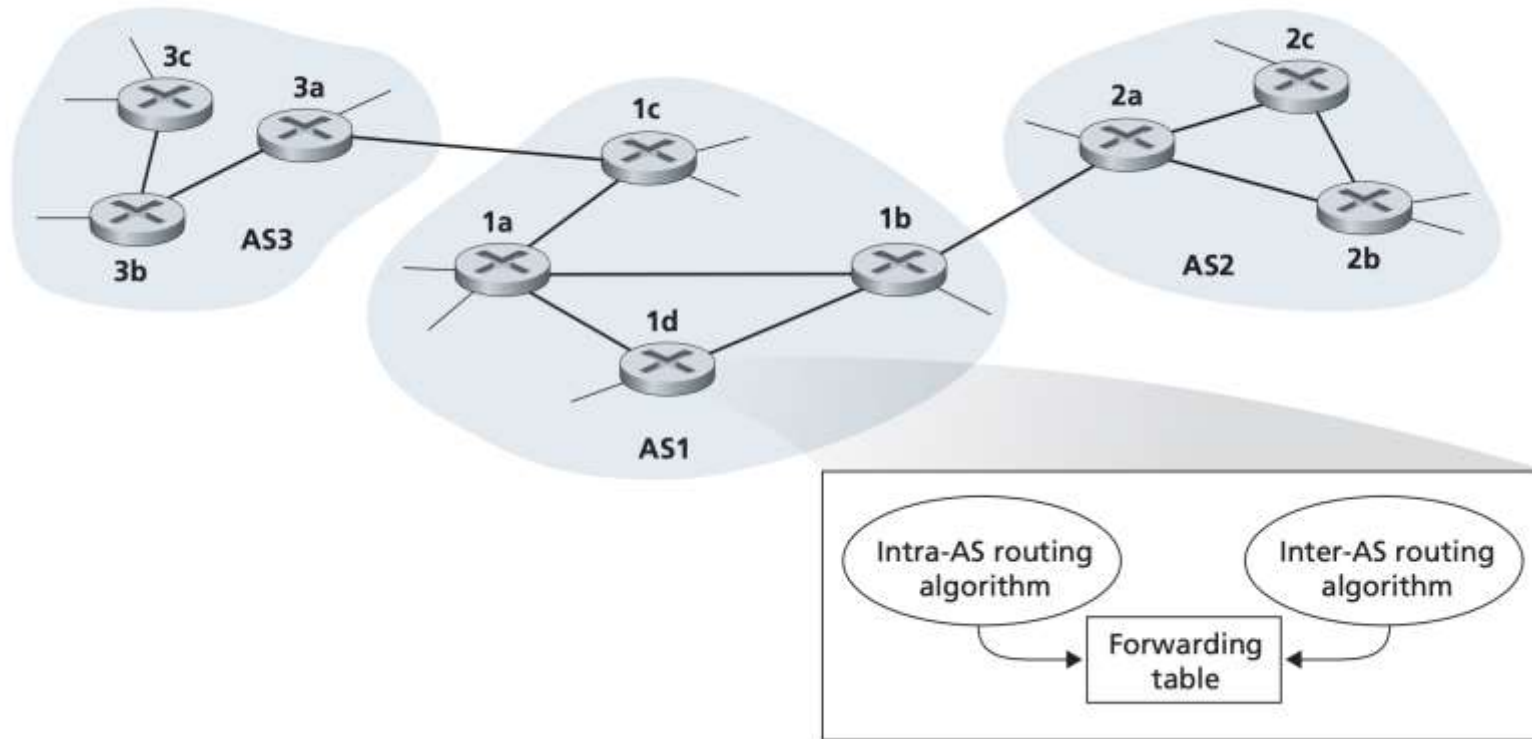
intra-AS routing

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

inter-AS routing

- routing among AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

Interconnected ASes

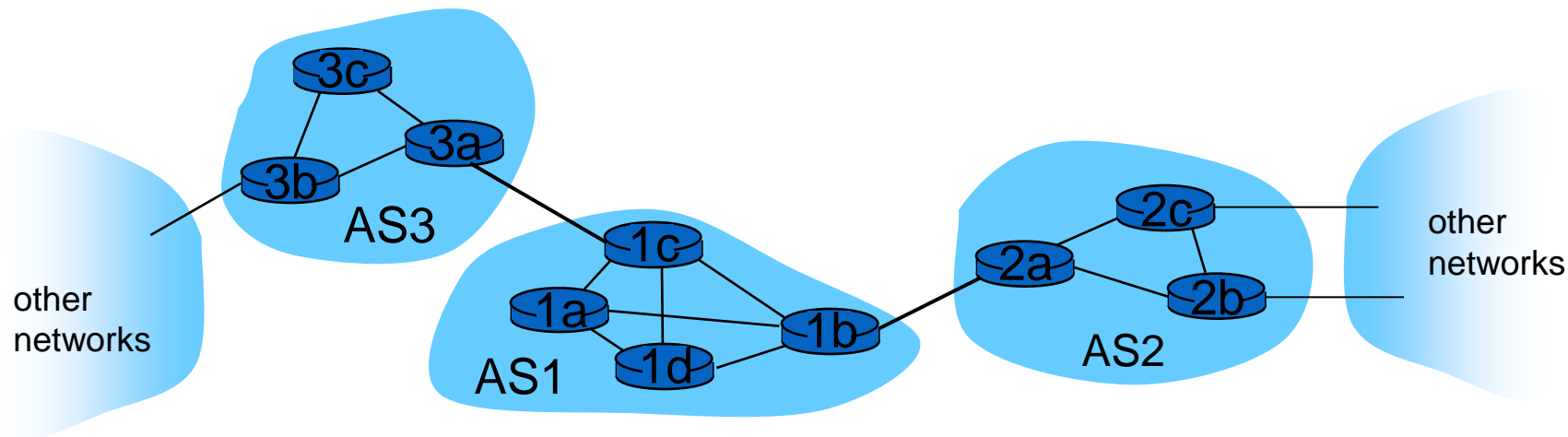


Forwarding table configured by both intra- and inter-AS routing algorithm

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

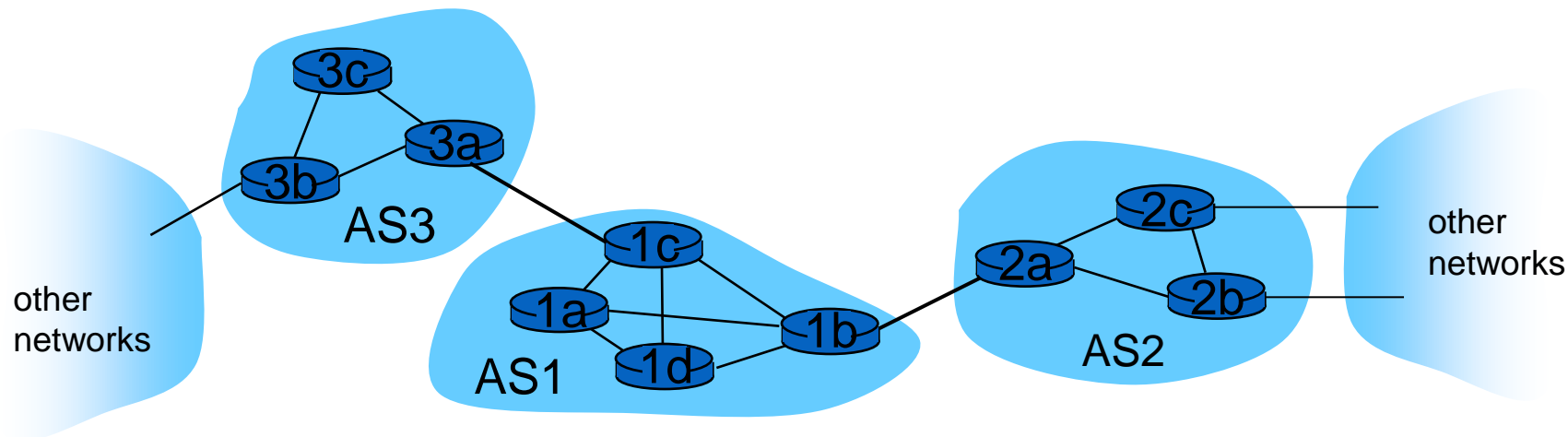
Inter-AS tasks

- Consider a router in AS1 and suppose it receives a packet whose destination is outside the AS.
- The router should forward the packet to one of its two gateway routers, 1b or 1c, but which one?



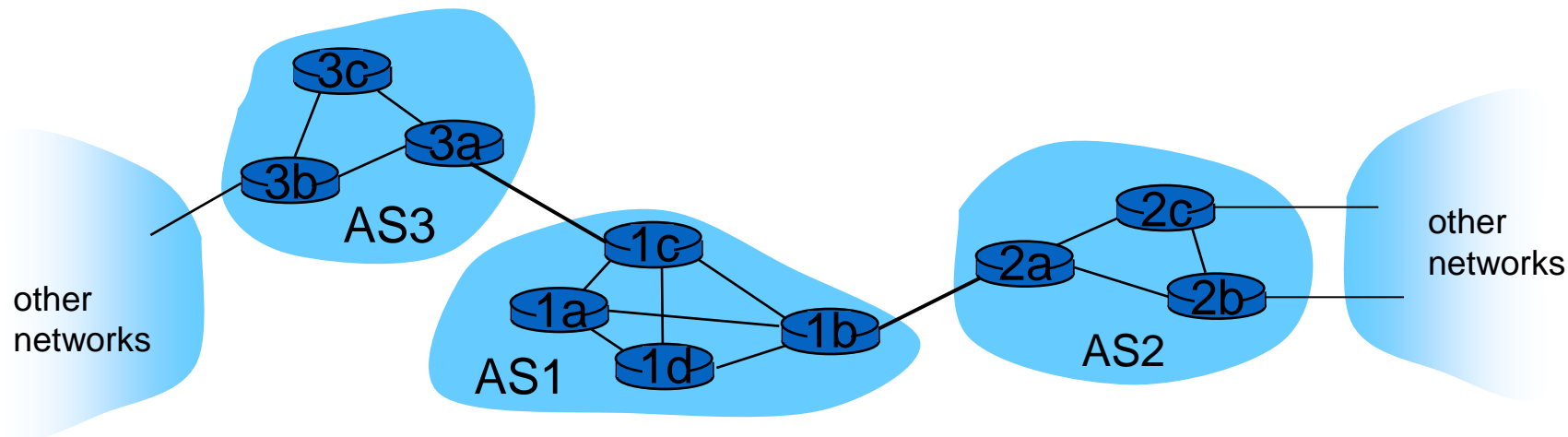
Inter-AS tasks

- To solve this problem, AS1 needs:
 - To learn which destinations are reachable via AS2 and which destinations are reachable via AS3
 - To propagate this reachability information to all the routers within AS1, so that each router can configure its forwarding table to handle external-AS destinations.



Inter-AS tasks

- Obtaining reachability information from neighboring ASs and propagating the reachability information to all routers internal to the AS are handled by the inter-AS routing protocol.
- Inter-AS routing protocol involves communication between two ASs, the two communicating ASs must run the same inter-AS routing protocol.
- In the Internet all ASs run the same inter-AS routing protocol, called **BGP4**



Intra-AS Routing

Intra-AS Routing

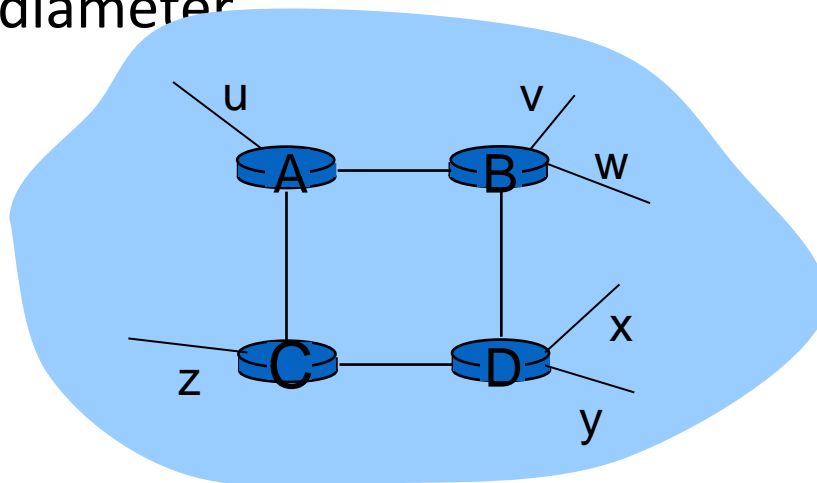
- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
 - RIP: Routing Information Protocol (Based on distance vector mechanism)
 - OSPF: Open Shortest Path First (Based on link state mechanism)

RIP (Routing Information Protocol)

- RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol.
- RIP uses hop count as a cost metric; that is, each link has a cost of 1, costs are from source router to a destination subnet.
- In RIP, routing updates are exchanged between neighbors approximately every 30 seconds using a RIP response message.
- The response message sent by a router or host contains a list of up to 25 destination subnets within the AS, as well as the sender's distance to each of those subnets. Response messages are also known as RIP advertisements.

RIP (Routing Information Protocol)

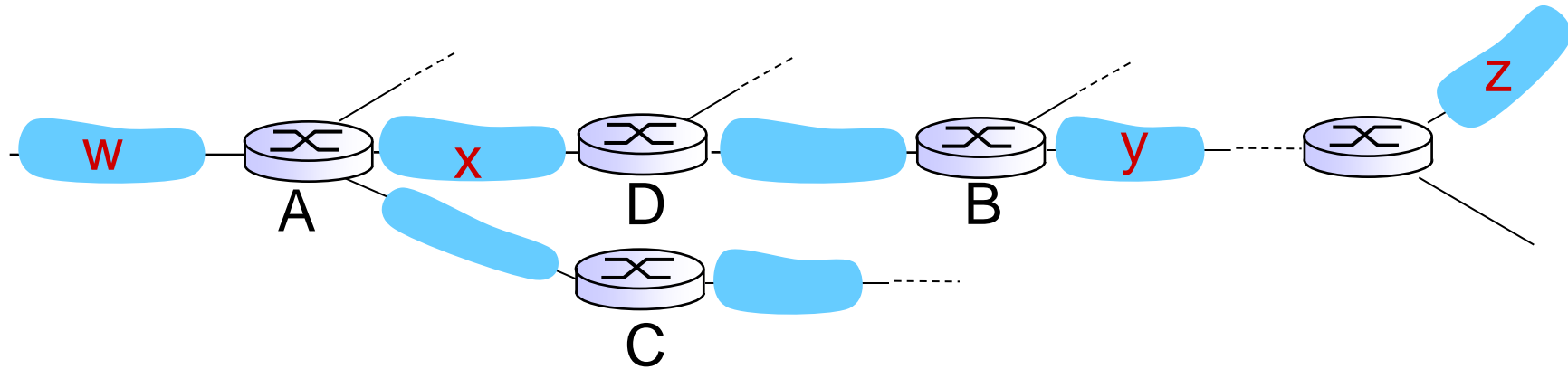
- RIP uses the term hop, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet.
- The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter



from router A to destination *subnets*:

<u>subnet</u>	<u>hops</u>
u	1
v	2
w	2
x	3
y	3
z	2

RIP: example



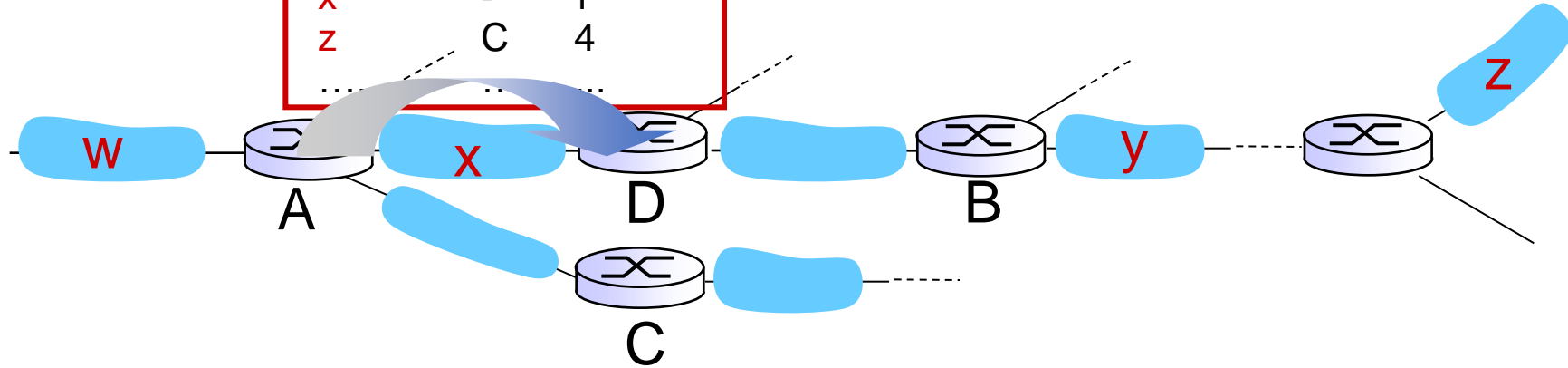
routing table in router D

destination subnet	next router	# hops to dest
W	A	2
y	B	2
Z	B	7
X	--	1
....

RIP: example

A-to-D advertisement

dest	next	hops
W	-	1
X	-	1
Z	C	4
...



routing table in router D

destination subnet	next router	# hops to dest
W	A	2
Y	B	2
Z	B → A	7 → 5
X	--	1
....

RIP: link failure, recovery

- RIP routers exchange advertisements approximately every 30 seconds.
- If a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable.
- When this happens, RIP modifies the local routing table and then propagates this information by sending advertisements to its neighboring routers (the ones that are still reachable).
- Router can also request information about its neighbor's cost to a given destination using RIP's request message. Routers send RIP request and response messages to each other over UDP using port number 520.

OSPF (Open Shortest Path First)

- A router constructs a complete topological map (that is, a graph) of the entire AS.
- The router then locally runs **Dijkstra's shortest-path** algorithm to determine a shortest-path tree to all subnets, with itself as the root node.
- Individual link costs are configured by the network administrator
 - Set all link costs to 1, thus achieve minimum-hop routing
 - Set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links.

OSPF (Open Shortest Path First)

- A router broadcasts routing information to all other routers in the autonomous system.
- A router broadcasts link-state information whenever there is a change in a link's state.
- It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed.

OSPF (Open Shortest Path First)

- **Security:** Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders from injecting incorrect information into router tables.
- Simple: the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext.
- MD5: is based on shared secret keys that are configured in all the routers. Then the router includes the resulting hash value in the OSPF packet.

OSPF (Open Shortest Path First)

- **Multiple same-cost paths:** When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- **Integrated support for unicast and multicast routing**
- **Support for hierarchy within a single routing domain:** has an ability to structure an AS hierarchically.

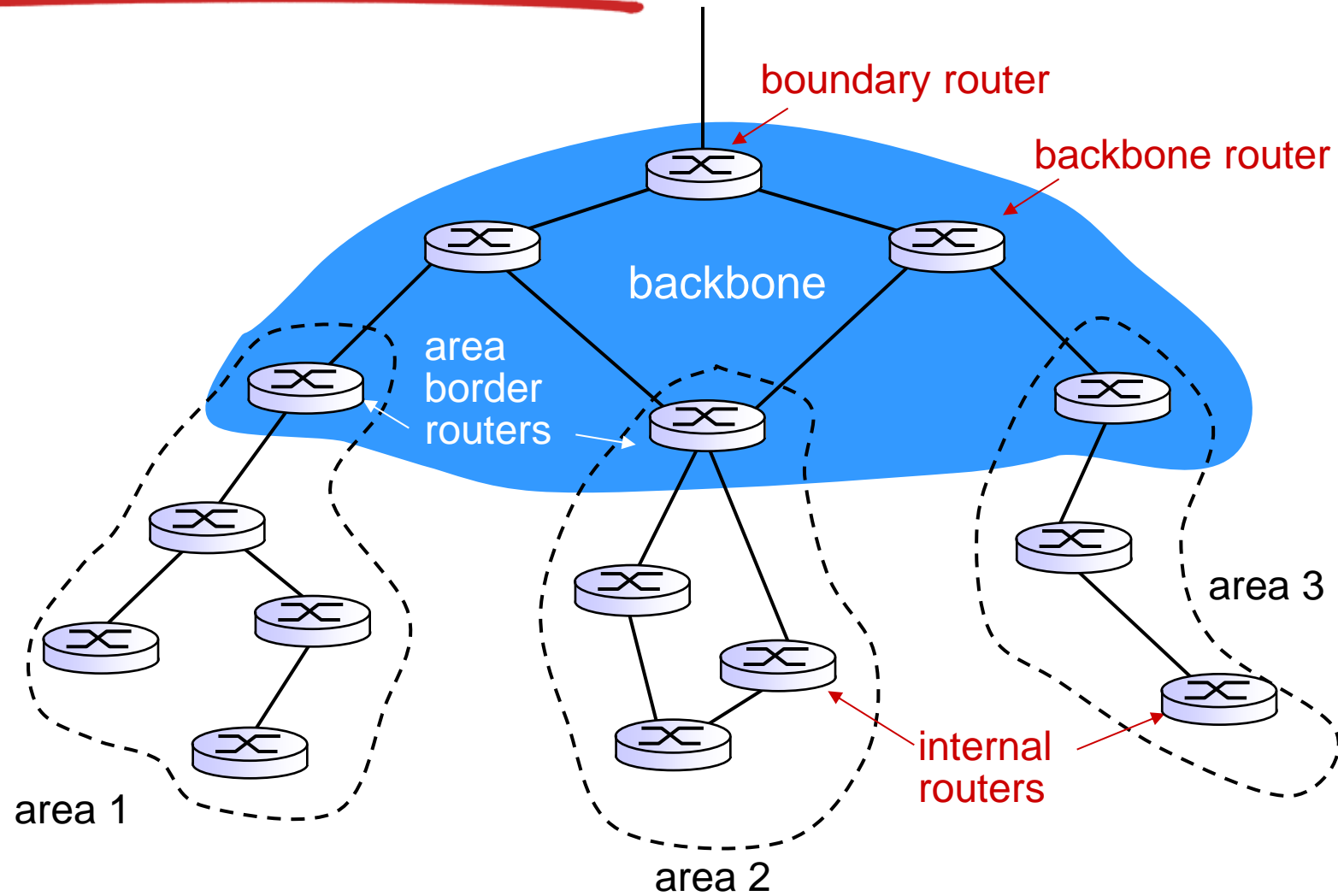
Hierarchical OSPF

- ❖ In OSPF, AS can be configured hierarchically into areas.
- ❖ Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area.
- ❖ Within each area, one or more area border routers are responsible for routing packets outside the area.
- ❖ One OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS.

Hierarchical OSPF

- ❖ The backbone always contains all area border routers in the AS and may contain non-border routers as well.
- ❖ Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

Hierarchical OSPF



IP address: Subnetting

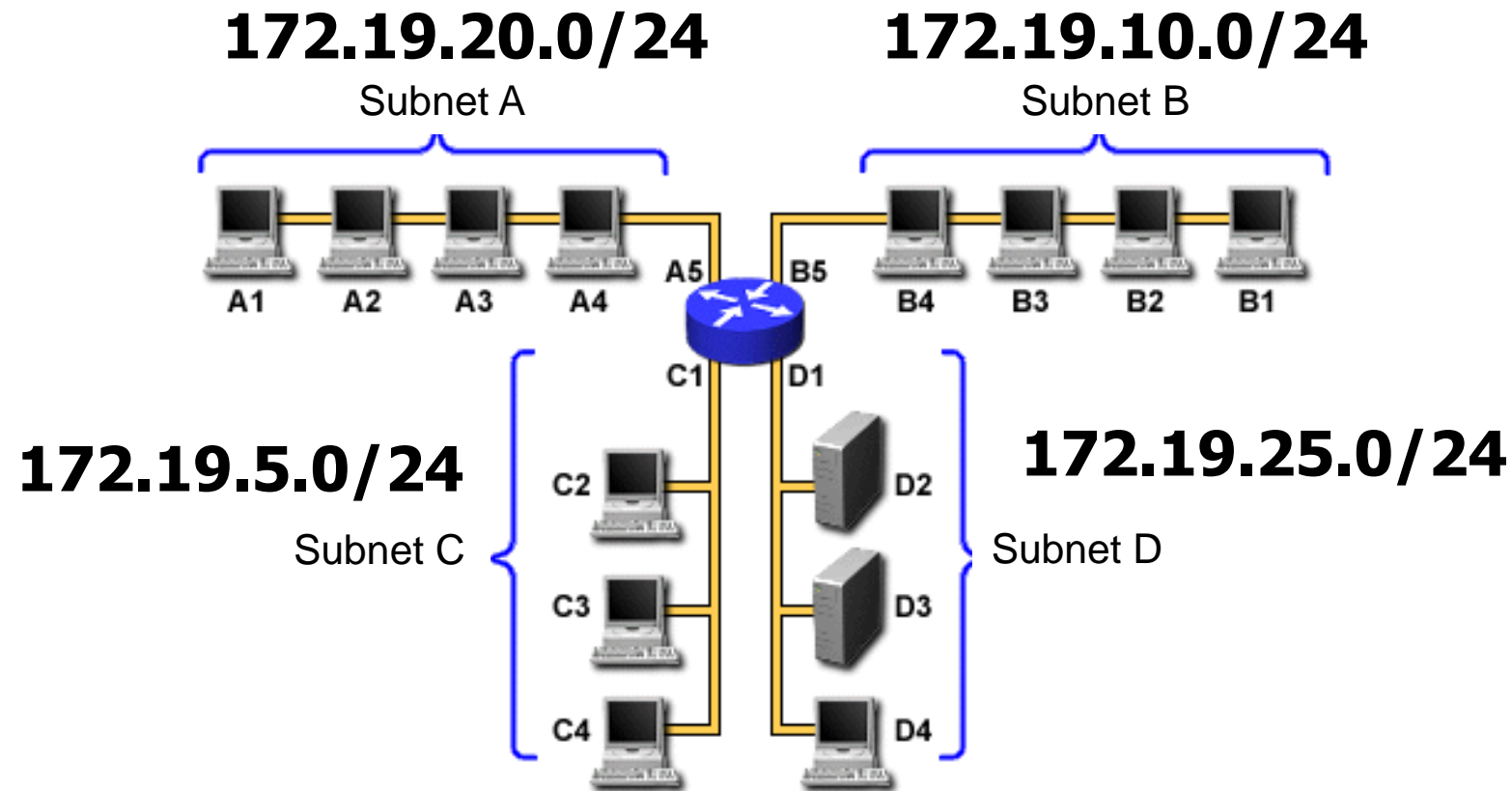
IP addressing: Subnetting

- *IP address*: 32-bit identifier for host, router *interface*
- *interface*: connection between host/router and physical link
 - router's typically have multiple interfaces
 - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP addresses associated with each interface*

Subnet Example

Network address **172.19.0.0** with **/16** network mask

Using Subnets: subnet mask 255.255.255.0 or /24



Subnet Example

Network address **172.19.0.0** with /16 network mask

Network	Network	Host	Host
172	19	0	0

- **Why would you want to subnet?**
 - Divide larger network into smaller networks
 - Better management of traffic.

Subnet Example

Network address **172.19.0.0** with /16 network mask

Network	Network	Host	Host
172	19	0	0

Using Subnets: subnet mask **255.255.255.0** or /24

Network	Network	Subnet	Host
---------	---------	--------	------

Network Mask:
255.255.0.0 or /16

11111111	11111111	00000000	00000000
----------	----------	----------	----------

Subnet Mask:
255.255.255.0 or /24

11111111	11111111	11111111	00000000
----------	----------	----------	----------



- Applying a mask which is larger than the default network mask, will divide your network into subnets.
- Subnet mask used here is 255.255.255.0 or /24

Subnet Example

Network address **172.19.0.0** with **/16** network mask

Using Subnets: **subnet mask** 255.255.255.0 or /24

Network	Network	Subnet	Host
172	19	0	Host
172	19	1	Host
172	19	2	Host
172	19	3	Host
172	19	etc.	Host
172	19	254	Host
172	19	255	Host

Subnets

**255
Subnets**

$2^8 - 1$

**Cannot use last
subnet as it
contains broadcast
address**

Subnet Example

Using Subnets: **subnet mask** 255.255.255.0 or /24

Network		Subnet	Hosts	Hosts Addresses	
172	19	0	1	→	254
172	19	1	1	→	254
172	19	2	1	→	254
172	19	3	1	→	254
172	19	etc.	1	→	254
172	19	254	1	→	254
172	19	255	Host	Each subnet has 254 hosts, $2^8 - 2$	

Important things to remember about Subnetting

- You can only subnet the host portion, you do not have control of the network portion.
- Subnetting does not give you more hosts, it only allows you to divide your larger network into smaller networks.
- When subnetting, you will actually lose host addresses

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Host IP Address

138.	101.	114.	250
10001010	01100101	01110010	11111010

Network Mask

255.	255.	0.	0
11111111	11111111	00000000	00000000

Subnet Mask

255.	255.	255.	192
11111111	11111111	11111111	11000000

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

Host IP Address

138.	101.	114.	250
10001010	01100101	01110010	11111010

Network Mask

255.	255.	0.	0
11111111	11111111	00000000	00000000

Major Network Address

	10001010	01100101	01110010	11111010
	11111111	11111111	00000000	
Bitwise	10001010	01100101	00000000	00000000
AND	138.	101.	0.	0

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

Major Network Address

10001010	01100101	00000000	00000000
138.	101.	0.	0

Major Network Broadcast Address

10001010	01100101	00000000	00000000
138.	101.	0.	0
10001010	01100101	11111111	11111111
138.	101.	255.	255

Range of Hosts if not subnetted

From	10001010	01100101	00000000	00000001
	138.	101.	0.	1
			
To	10001010	01100101	11111111	11111110
	138.	101.	255.	254

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

Host IP	138.	101.	114.	250
address	10001010 01100101 01110010 11111010			

Network	255.	255.	0.	0
Mask	11111111 11111111 00000000 00000000			

Subnet	255.	255.	255.	192
Mask	11111111 11111111 11111111 11000000			

Subnet Address

Host IP	138.	101.	114.	250
address	10001010 01100101 01110010 11111010			

Subnet	255.	255.	255.	192
Mask	11111111 11111111 11111111 11000000			

138.	101.	114.	192
10001010 01100101 01110010 11000000			

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

Host IP	138.	101.	114.	250
address	10001010	01100101	01110010	11111010

Network	255.	255.	0.	0
Mask	11111111	11111111	00000000	00000000

Subnet	255.	255.	255.	192
Mask	11111111	11111111	11111111	11000000

Subnet	138.	101.	114.	192
Address	10001010	01100101	01110010	11000000

Broadcast Address

138.	101.	114.	192
10001010	01100101	01110010	11000000

138.	101.	114.	255
10001010	01100101	01110010	11111111

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Broadcast Address
 - Range of Host Addresses (first host and last host)
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Subnetting – Example

Host IP address	138.	101.	114.	250
	10001010	01100101	01110010	11111010
Subnet Mask	255.	255.	255.	192
	11111111	11111111	11111111	11000000
Subnet Address	138.	101.	114.	192
	10001010	01100101	01110010	11000000
Broadcast Address	138.	101.	114.	255
	10001010	01100101	01110010	11111111
Range of Hosts				
From	138.	101.	114.	193
	10001010	01100101	01110010	110000001
			
To	10001010	01100101	01110010	11111110
	138.	101.	114.	255

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Host IP	138.	101.	114.	250
Address	10001010	01100101	01110010	11111010

Network	255.	255.	0.	0
Mask	11111111	11111111	00000000	00000000

Subnet	255.	255.	255.	192
Mask	11111111	11111111	11111111	11000000

Total number of subnets

Number of subnet bits 10

$$2^{10} = 1,024$$

1,024 total subnets

Subnetting – Example

- **Host IP Address:** 138.101.114.250
- **Network Mask:** 255.255.0.0 (or /16 Mask bits)
- **Subnet Mask:** 255.255.255.192 (or /26 Mask bits)

Host IP	138.	101.	114.	250
Address	10001010	01100101	01110010	11111010

Network	255.	255.	0.	0
Mask	11111111	11111111	00000000	00000000

Subnet	255.	255.	255.	192
Mask	11111111	11111111	11111111	11000000

Number of hosts per subnet

Number of host bits 6

$2^6 = 64$ (host per subnets)

Subtract one for the subnet address and one for the broadcast address

62 hosts per subnet

Example #1

- 191.54.38.15
 - 10111111. 00110110. 00100110. 00001111
- Network mask: 255.255.0.0
- Subnet mask: 255.255.255.0

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Range of Host Addresses (first host and last host)
 - Broadcast Address
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Example #2

- 191.54.38.207
 - ???
- Network mask: 255.255.0.0
- Subnet mask: 255.255.255.224

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Range of Host Addresses (first host and last host)
 - Broadcast Address
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

Example #3

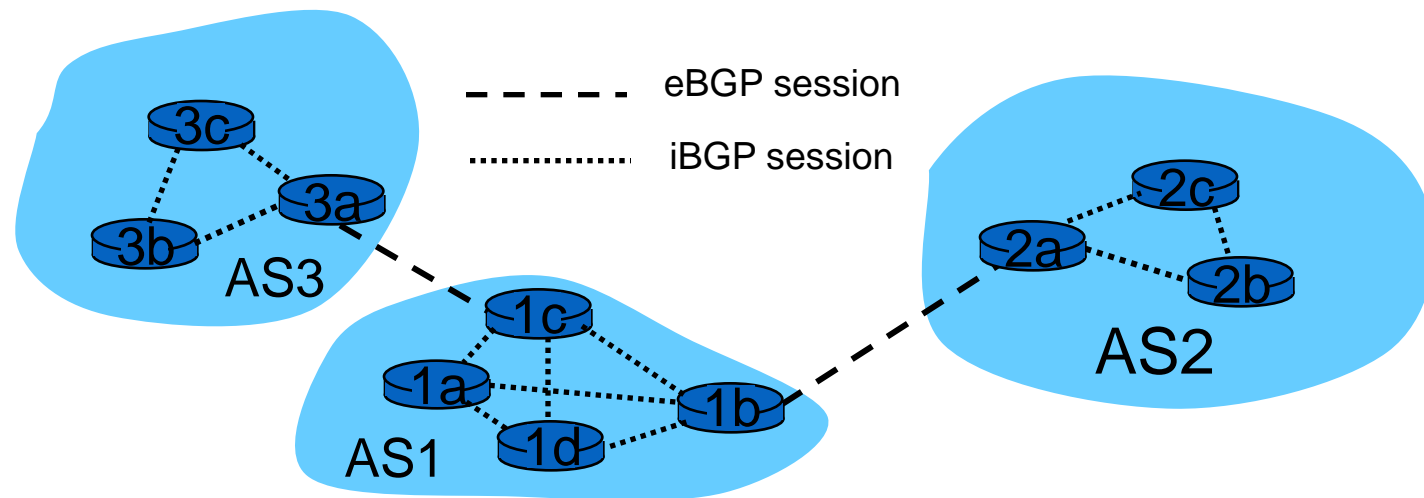
- 191.54.38.207
 - ???
- Network mask: 255.255.255.0
- Subnet mask: 255.255.255.224

Given the following Host IP Address, Network Mask and Subnet mask find the following information:

- Major Network Information
 - Major Network Address
 - Major Network Broadcast Address
 - Range of Hosts if not subnetted
- Subnet Information
 - Subnet Address
 - Range of Host Addresses (first host and last host)
 - Broadcast Address
- Other Subnet Information
 - Total number of subnets
 - Number of hosts per subnet

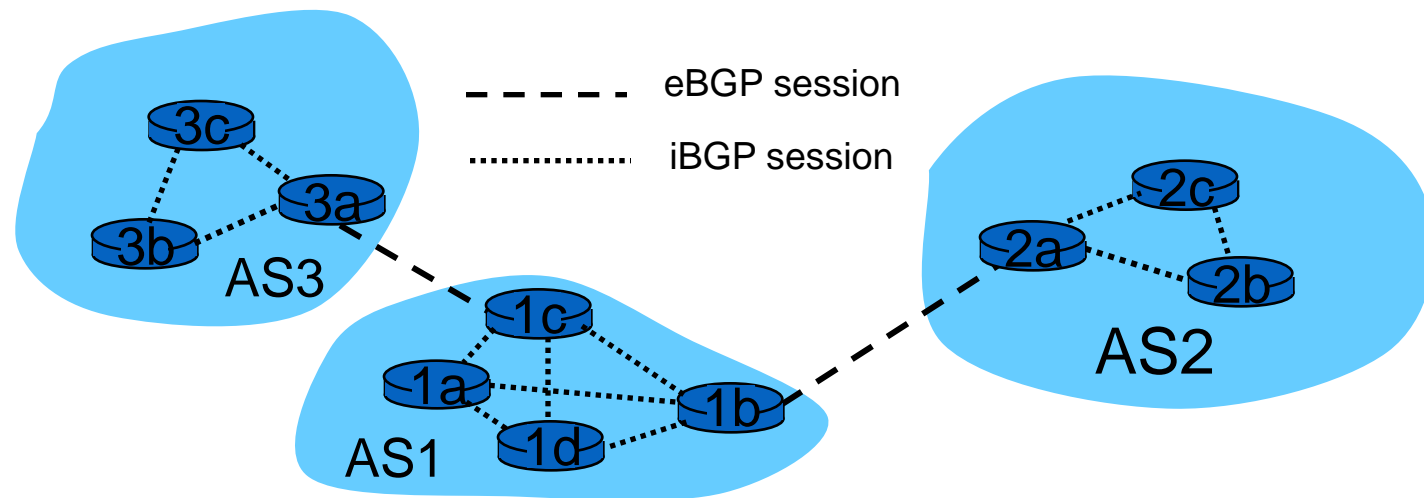
BGP basics

- In BGP, pairs of routers exchange routing information over **TCP connections** using **port 179**.
 - BGP sessions do not need to correspond to physical links.
- In figure, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a.



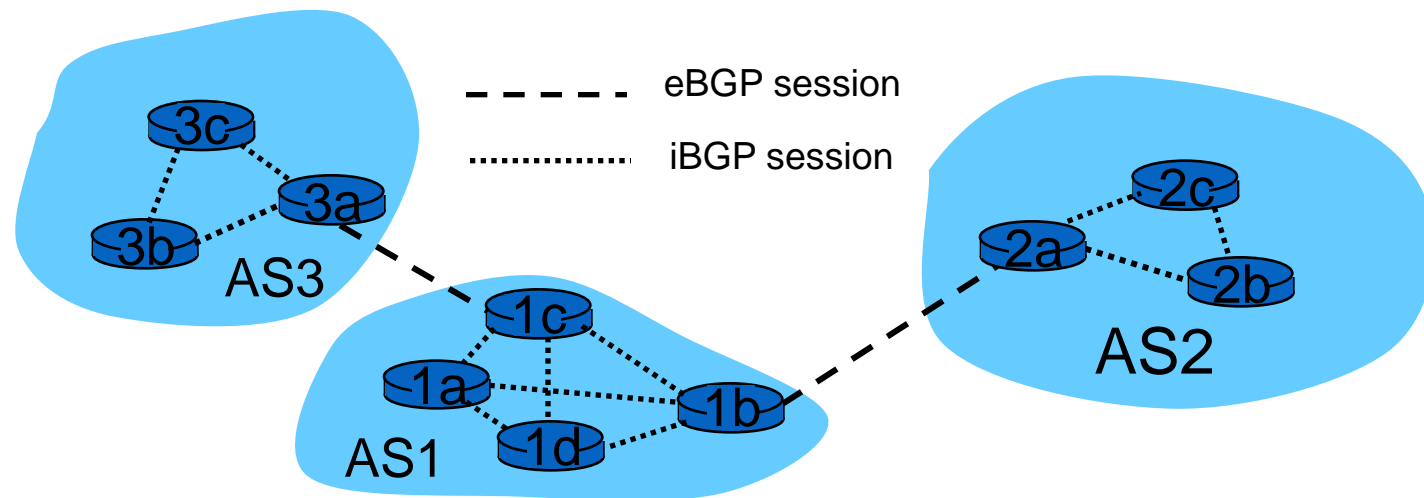
BGP basics

- There are also BGP TCP connections between routers within an AS.
- For each TCP connection, the two routers at the end of the connection are called **BGP peers**, and the TCP connection along with all the BGP messages sent over the connection is called a **BGP session**.



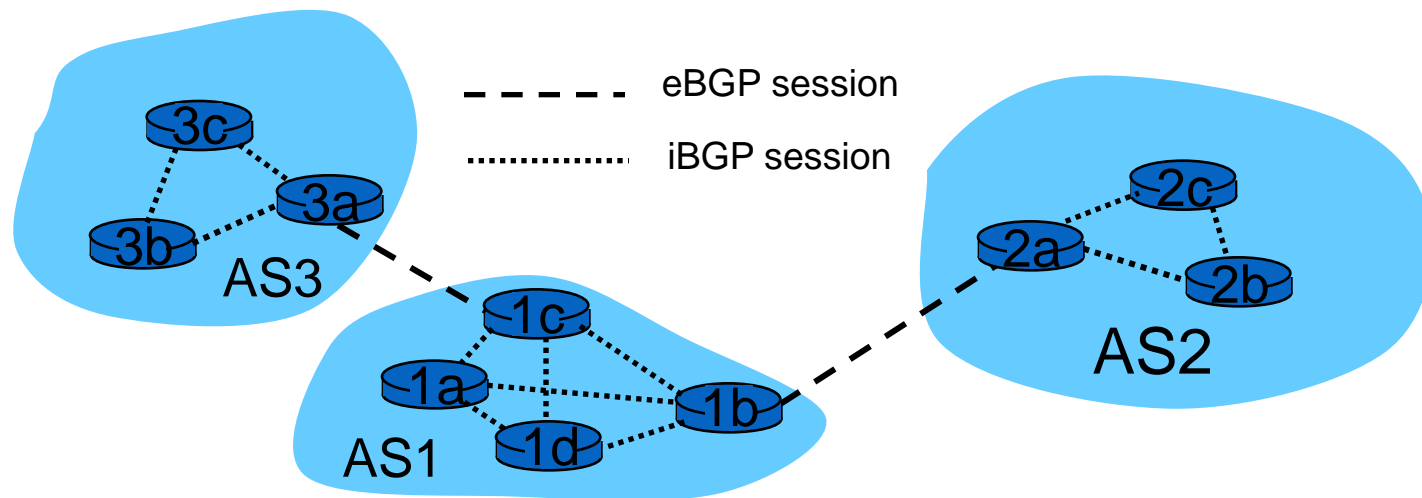
BGP basics

- A BGP session that spans two ASs is called an **external BGP** (eBGP) session, and a BGP session between routers in the same AS is called an **internal BGP** (iBGP) session.



Distributing reachability info

- using eBGP session between 3a and 1c, AS3 sends reachability info to AS1.
 - 1c can then use iBGP to distribute new reachability info to all routers in AS1
 - 1b can then re-advertise new reachability info to AS2 over 1b-to-2a eBGP session
- when router learns of new info, it creates entry in its forwarding table.



Understanding BGP

- BGP allows each AS to learn which destinations are reachable via its neighboring ASs.
- In BGP, destinations are not hosts but instead are CIDRized prefixes, with each prefix representing subnets.
- Suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1.
- Suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3. Then, AS3 could advertise to AS1 the more specific prefix 138.16.67/24 and AS2 could still advertise to AS1 the aggregated prefix 138.16.64/22.

Understanding BGP

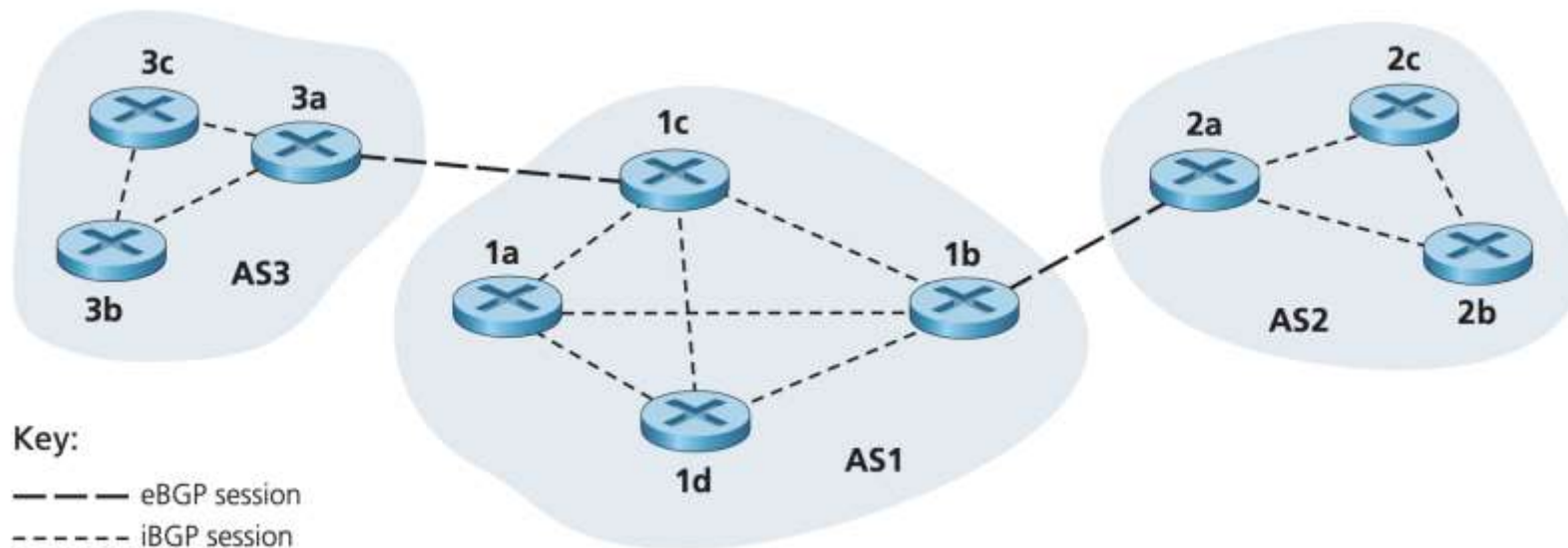
- In BGP, an AS is identified by its globally unique AS number (ASN)
- AS numbers, like IP addresses, are assigned by ICANN regional registries
- When a router advertises a prefix across a BGP session, it includes BGP attributes.
- Two important attributes are **AS-PATH** and **NEXT-HOP**

Understanding BGP

AS-PATH

This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the AS-PATH attribute.

Routers use the AS-PATH attribute to detect and prevent **looping** advertisements; specifically, if a router sees that its AS is contained in the path



AS-PATH and AS number

- AS-PATH is a path vector of AS numbers (ASNs)
 - ICANN serving as the ultimate authority for delegating ASNs
 - Internet Corporation for Assigned Names and Numbers (ICANN)
- ASNs are 16 bit numbers ranging from 1 – 65535
 - 1-64511 are public and used by provider ASes
 - 64512 – 65534 for consumer ASes
 - 0 and 65535 are reserved

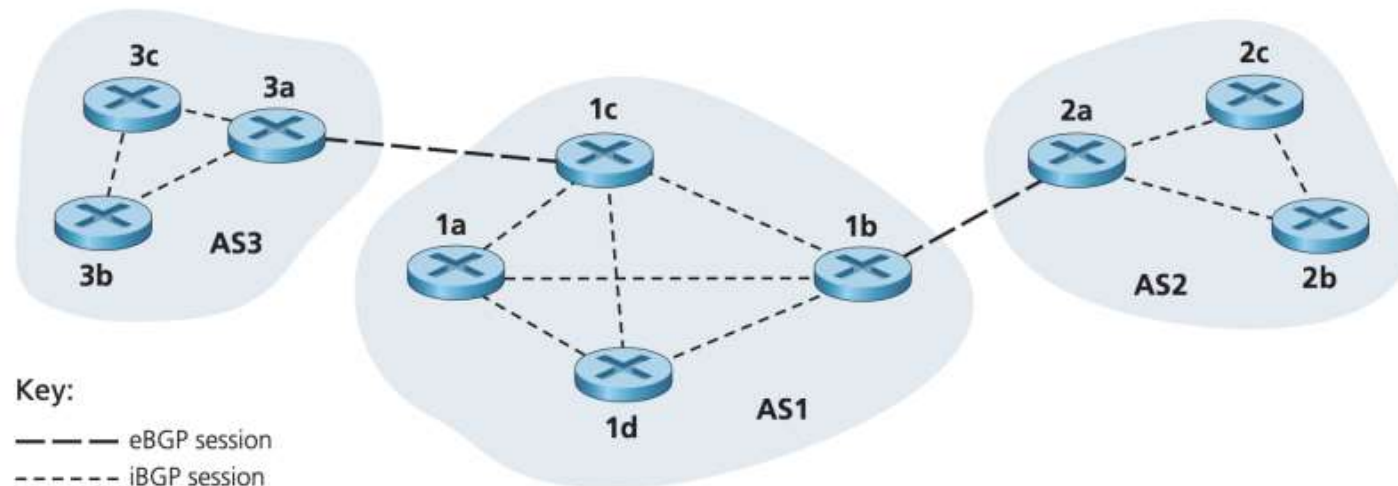
Understanding BGP

NEXT-HOP

When the gateway router 3a in AS3 advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, and an AS-PATH to the prefix.

It also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c.

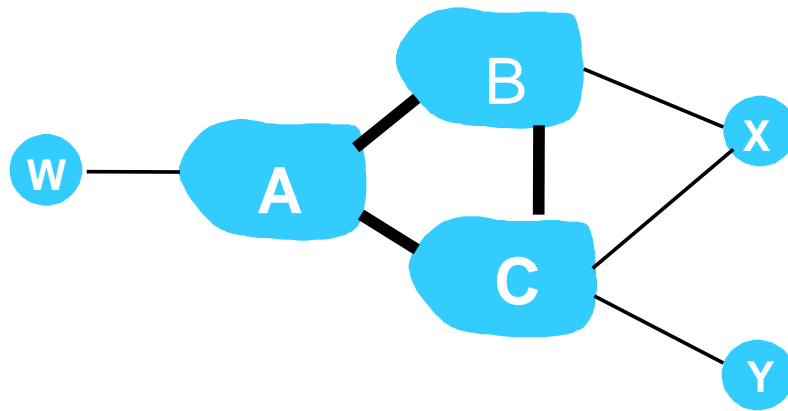
NEXT-HOP attribute is used by routers to properly configure their forwarding tables.





Routing Policy & Route Attributes

- ASes are not just bound by physical relationships but also by business (Provider, consumer relationships)
- “Best” path chosen by a BGP router not necessarily the shortest route in the physical topology
 - There are various policies (import & export)
 - Specified using “route attributes”

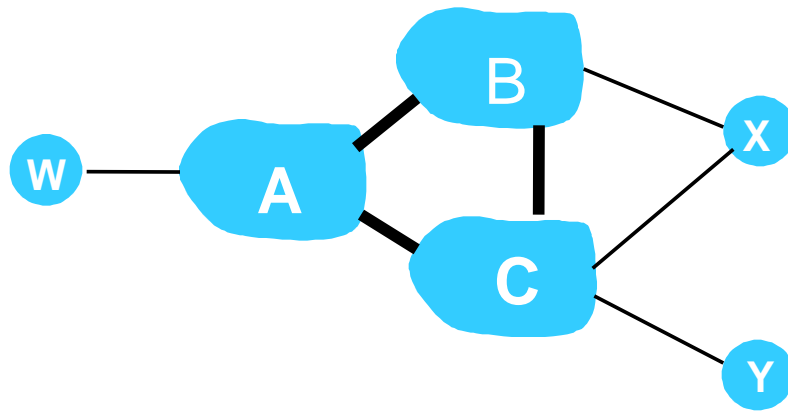
BGP routing policy





legend:  provider network
 customer network:

- ❖ A, B, C, X, Y, W are ASs, and A, B, C are backbone *provider networks*
- ❖ X, W, Y are customer (of provider networks)
- ❖ A, B, and C, all peer with each other, and provide full BGP information to their customer networks
- ❖ X is *dual-homed*: attached to two networks

BGP routing policy

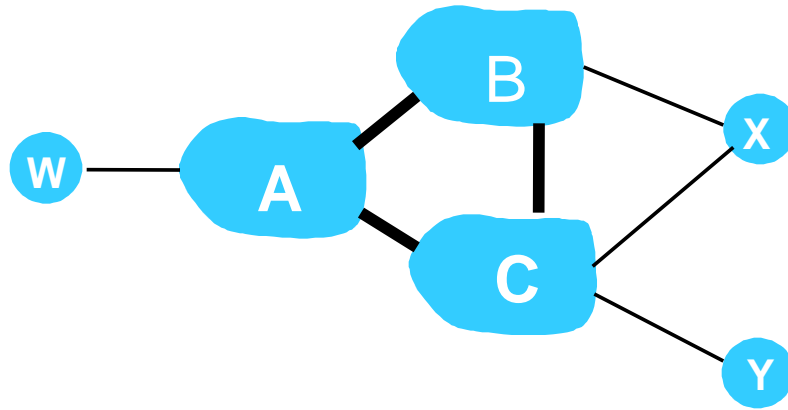




legend:  provider network
 customer network:

Can B sent traffic to Y through X?

- ❖ Even though X may know of a path, say XCY, that reaches network Y, it will not advertise this path to B.
- ❖ Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X.

BGP routing policy (2)

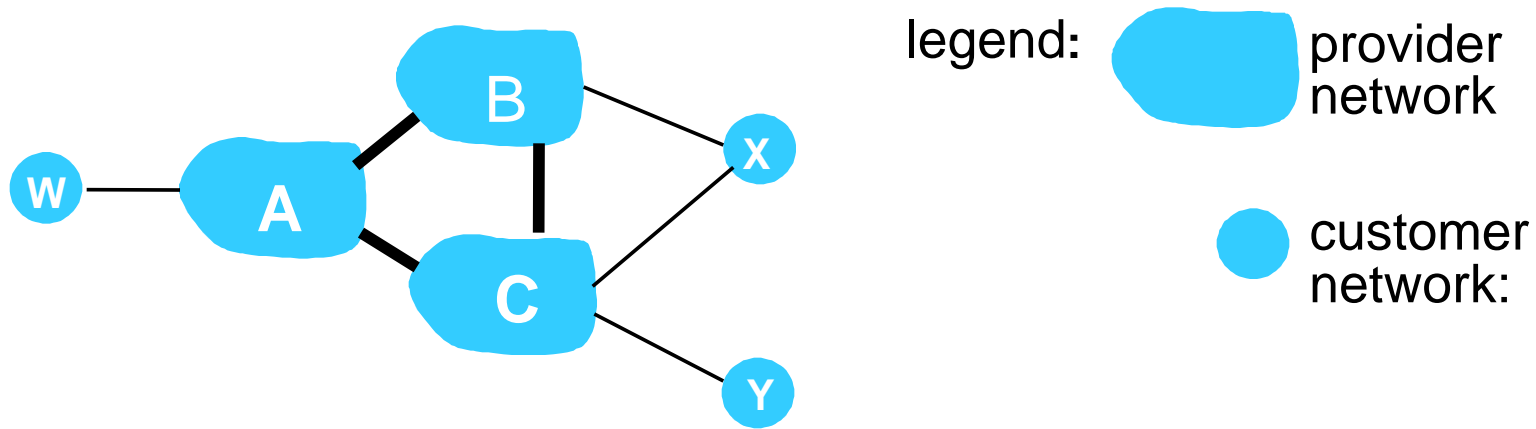


legend:  provider network
 customer network:

- ❖ A advertises path AW to B
- ❖ B advertises path BAW to X
- ❖ **Should B advertise path BAW to C?**

If it does so, then C could route traffic to W via CBAW. If A, B, and C are all backbone providers, then B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C.

BGP routing policy (2)



- ❖ A rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a network that is a customer of that ISP; otherwise the traffic would be getting a free ride on the ISP's network

BGP Elimination Rules

If there are two or more routes to **the same prefix**, then BGP sequentially invokes the following elimination rules until one route remains:

- 1) Routes are assigned a local preference value as one of their attributes. The routes with the highest local preference values are selected.
- 2) The route with the shortest AS-PATH is selected.
- 3) Hot Potato Routing

Hot Potato Routing

The AS gets rid of the packet (the hot potato) as quickly as possible (more precisely, as inexpensively as possible).

This is done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost among all gateways with a path to the destination.

It use information from the intra-AS routing protocol to determine the path costs.

BGP security issues

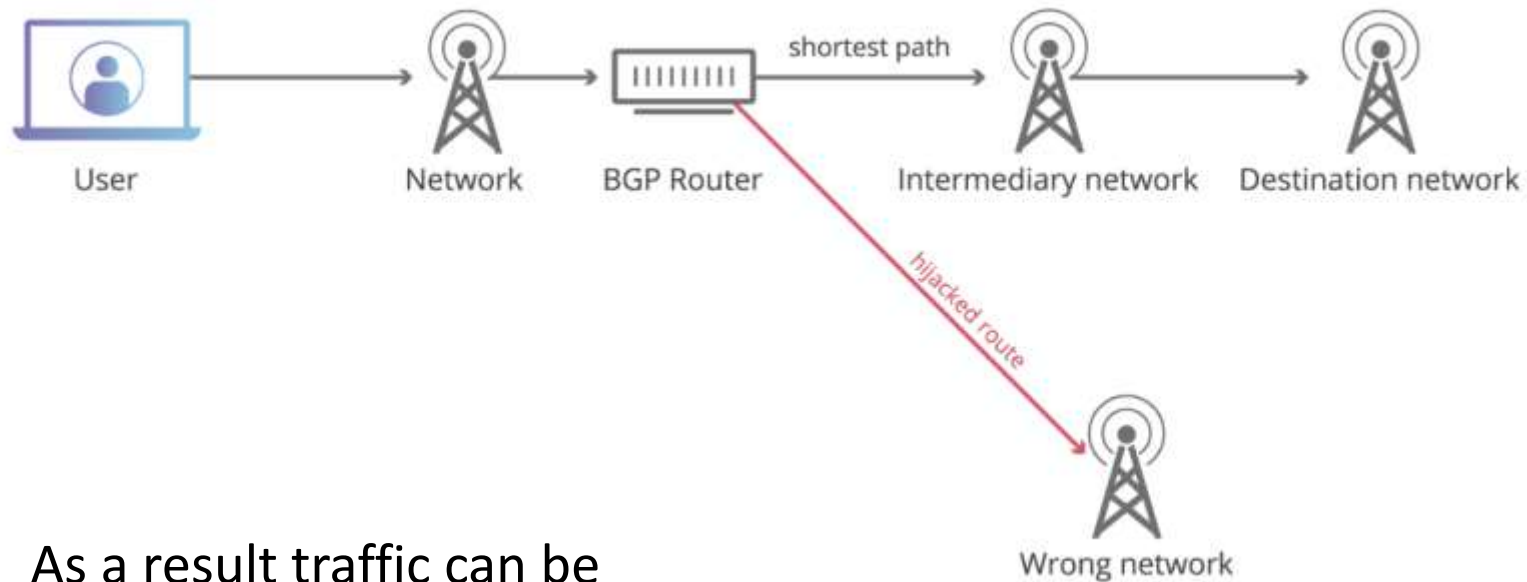
BGP security issues

BGP hijacking

BGP hijacking is when attackers maliciously **reroute** Internet traffic, by falsely announcing ownership of groups of IP addresses, that they do not actually own, control, or route to.

A BGP hijack is much like if someone were to change out all the signs on a stretch of freeway and reroute automobile traffic onto incorrect exits.

BGP security issues - BGP hijacking



As a result traffic can be

- Monitored
- Dropped (black holed)
- Directed to fake websites

BGP security issues - BGP hijacking

- In April 2018, a Russian provider announced a number of IP prefixes that actually belong to Route53 Amazon DNS servers.
 - Via BGP hijacking, the hackers hijacked Amazon DNS queries so that DNS queries for myetherwallet.com went to servers they controlled, returned the wrong IP address, and directed HTTP requests to the fake website.
- Users attempting to log in to a cryptocurrency site were redirected to a fake version of the website.
- The hackers were thus able to steal approximately \$152,000 in cryptocurrency.
- Learn more: <https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies/>

Source: <https://www.cloudflare.com/learning/security/glossary/bgp-hijacking/>

BGP security issues - BGP hijacking

IP prefix filtering

- Most networks should only accept IP prefix declarations if necessary, and should only declare their IP prefixes to certain networks, not the entire Internet. (In practice difficult to enforce)

BGP hijacking detection

- Increased latency, degraded network performance, and misdirected Internet traffic are all possible signs of a BGP hijack. Many larger networks will monitor BGP updates to ensure their clients do not face latency issues.

Making BGP more secure

- More secure routing solutions for the Internet as a whole (such as BGPsec) are being developed, but there is no adoption of them yet.

BGP security issues - BGP Route Leak

BGP Route Leak

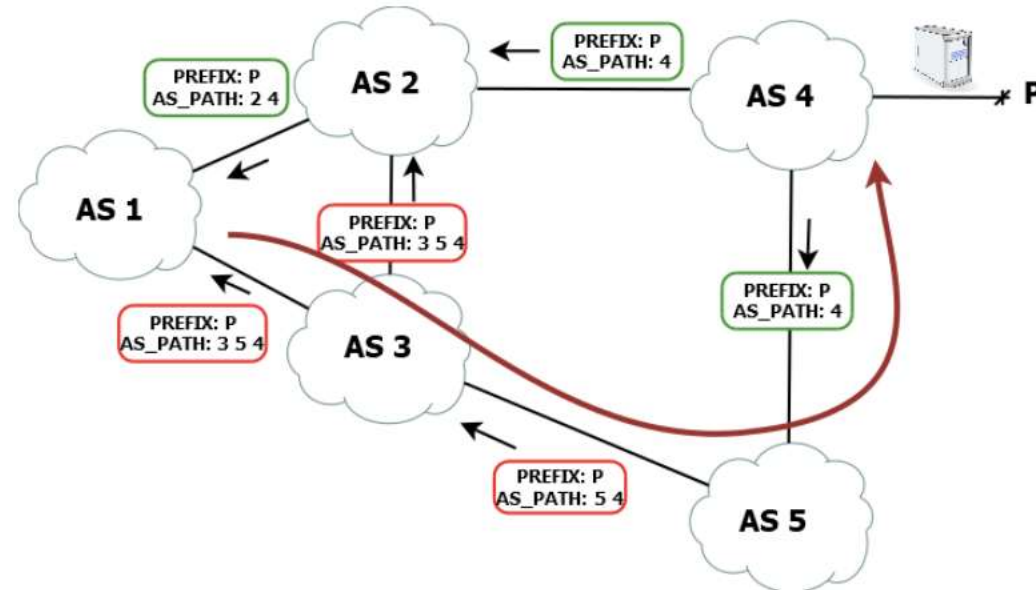
A route leak is formally defined as the “propagation of a BGP announcement(s) beyond their intended scope” [RC7908]. The scope is defined by BGP import and export policies that AS’s use to regulate the set of routes exchanged over a BGP session.

BGP security issues - BGP Route Leak

AS5 is a customer of AS4 and a peer of AS3.

AS4 announces to AS5 the reachability of prefix P and AS5 propagates this route to its peer AS3, causing a provider-to-peer leak.

If AS3 accepts the leaked route, then other AS's may be affected (AS1), but more importantly, AS5 becomes a transit between a peer (AS3) and a provider (AS4) for the traffic directed to P coming from AS3.



BGP security issues - BGP Route Leak

- The AS causing the leak needs to handle an unexpected amount of traffic and its network may be **under-provisioned to handle** it.
- A leak could also be caused **intentionally** with the purpose to inspect the traffic before it gets directed to the legitimate destination.

BGP security issues - BGP Route Leak

- China Telecom leaked routes from Verizon's Asia-Pacific network that were learned through a common South Korean peer AS.
- The result was that a portion of internet traffic from around the world destined for Verizon Asia-Pacific was misdirected through mainland China.
- Without this leak, China Telecom would have only been in the path to Verizon Asia-Pacific for traffic originating from its customers in China.
- For ten days in 2017, Verizon passed its US routes to China Telecom through the common South Korean peer causing a portion of US-to-US domestic internet traffic to be misdirected through mainland China.

BGP security issues - BGP Route Leak



Dijkstra's link-state routing algorithm

Dijkstra's link-state routing algorithm

1 *Initialization:*

2 $N' = \{u\}$ /* compute least cost path from u to all other nodes */

3 for all nodes v

4 if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */

5 then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */

6 else $D(v) = \infty$

7



8 *Loop*

9 find w not in N' such that $D(w)$ is a minimum

10 add w to N'

11 update $D(v)$ for all v adjacent to w and not in N' :

12 **$D(v) = \min (D(v), D(w) + c_{w,v})$**

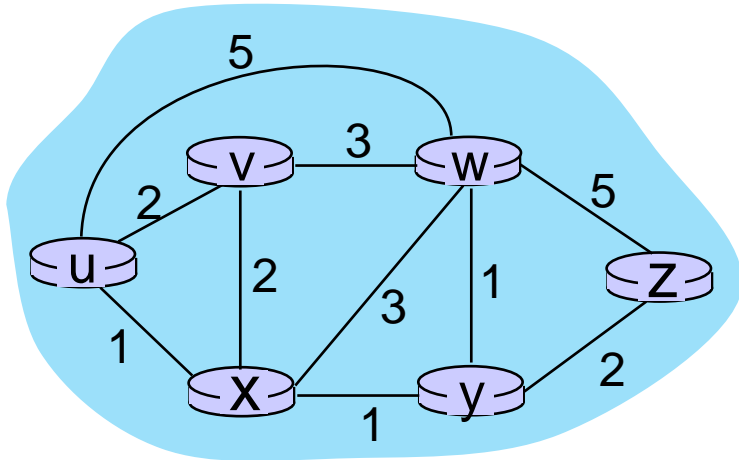
13 /* new least-path-cost to v is either old least-cost-path to v or known

14 least-cost-path to w plus direct-cost from w to v */

15 *until all nodes in N'*

Dijkstra's algorithm: an example

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

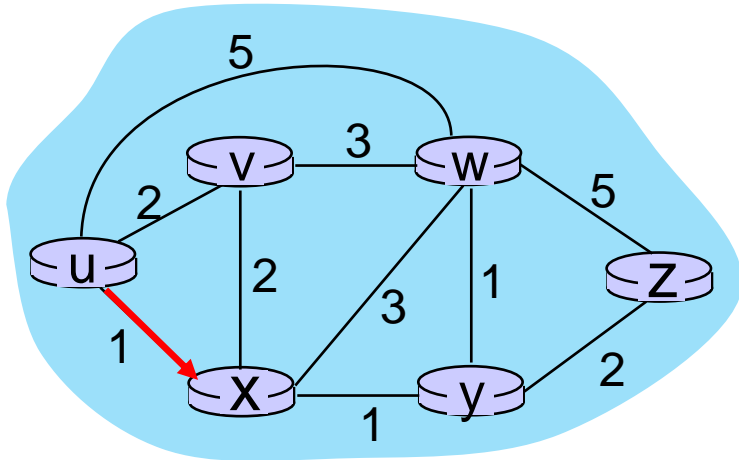


Initialization (step 0):

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

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						
3						
4						
5						



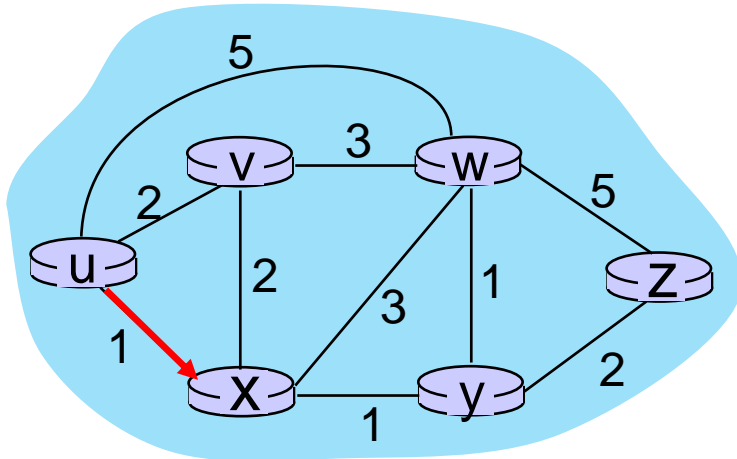
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

		v	w	x	y	z
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						
3						
4						
5						



8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

11 update $D(b)$ for all b adjacent to a and not in N' :

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

$$D(v) = \min (D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$$

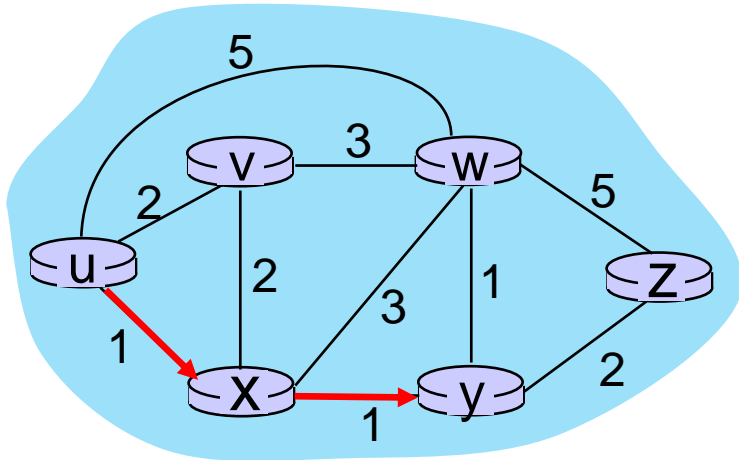
$$D(w) = \min (D(w), D(x) + c_{x,w}) = \min(5, 1+3) = 4$$

$$D(y) = \min (D(y), D(x) + c_{x,y}) = \min(\infty, 1+1) = 2$$



Dijkstra's algorithm: an example

		v	w	x	y	z
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					
3						
4						
5						



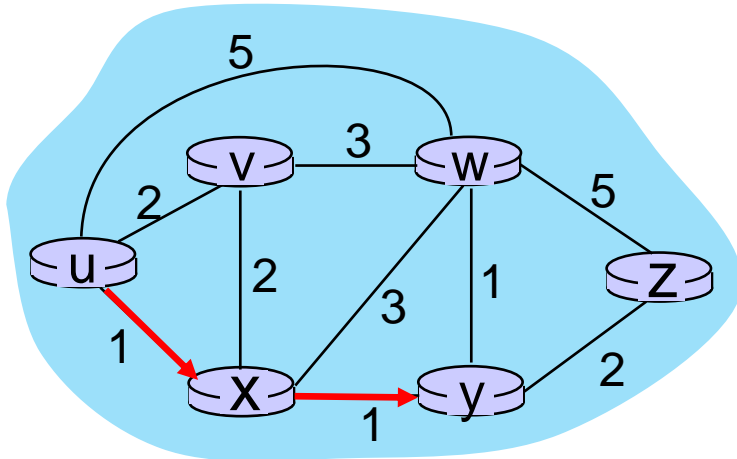
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

		v	w	x	y	z
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						
4						
5						



8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

11 update $D(b)$ for all b adjacent to a and not in N' :

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

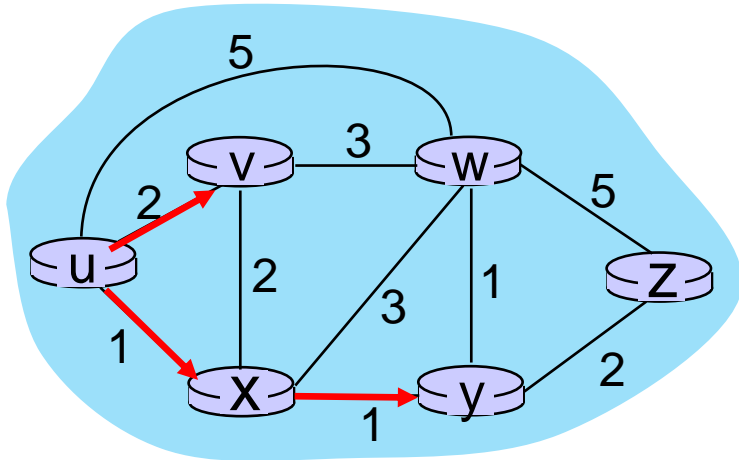
$$D(w) = \min (D(w), D(y) + c_{y,w}) = \min (4, 2+1) = 3$$

$$D(z) = \min (D(z), D(y) + c_{y,z}) = \min (\infty, 2+2) = 4$$



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					
4						
5						



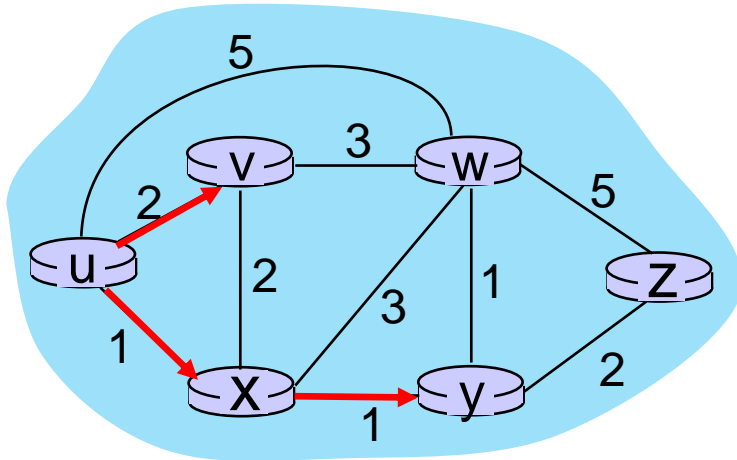
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

		v	w	x	y	z
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						
5						



8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

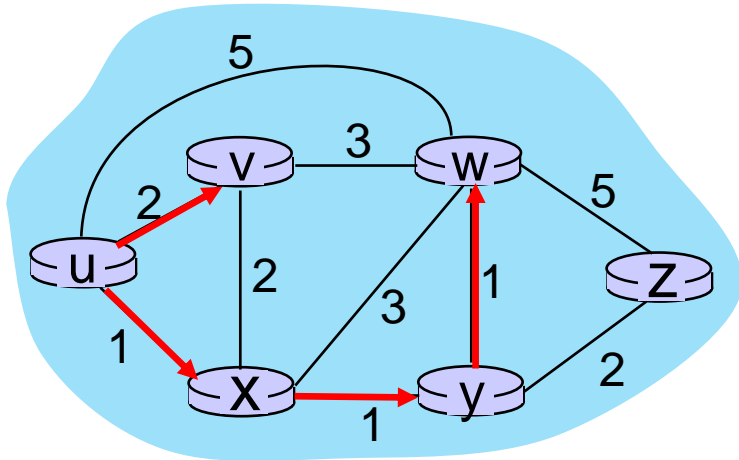
11 update $D(b)$ for all b adjacent to a and not in N' :

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

$$D(w) = \min (D(w), D(v) + c_{v,w}) = \min (3, 2+3) = 3$$

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					
5						



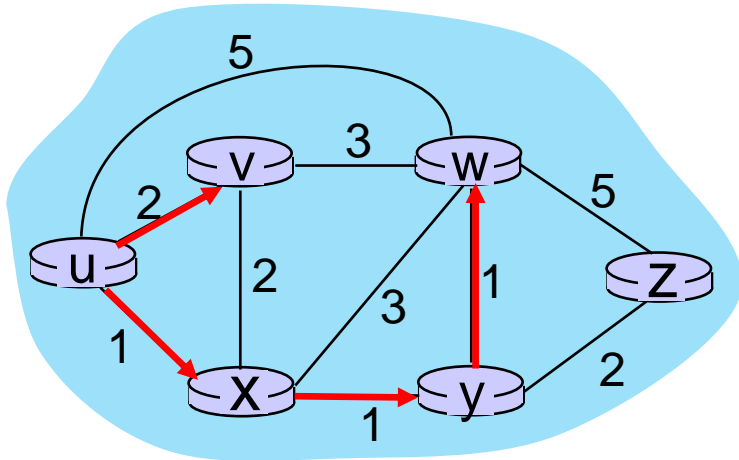
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

		v	w	x	y	z
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						



8 *Loop*

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

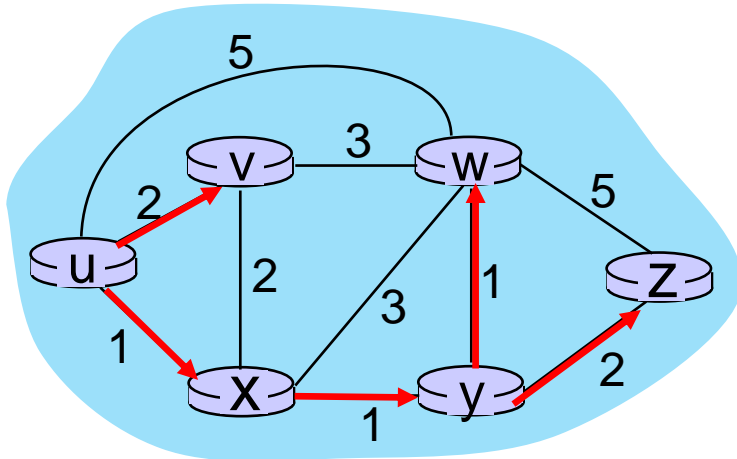
11 update $D(b)$ for all b adjacent to a and not in N' :

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

$$D(z) = \min (D(z), D(w) + c_{w,z}) = \min (4, 3+5) = 4$$

Dijkstra's algorithm: an example

		v	w	x	y	z
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					



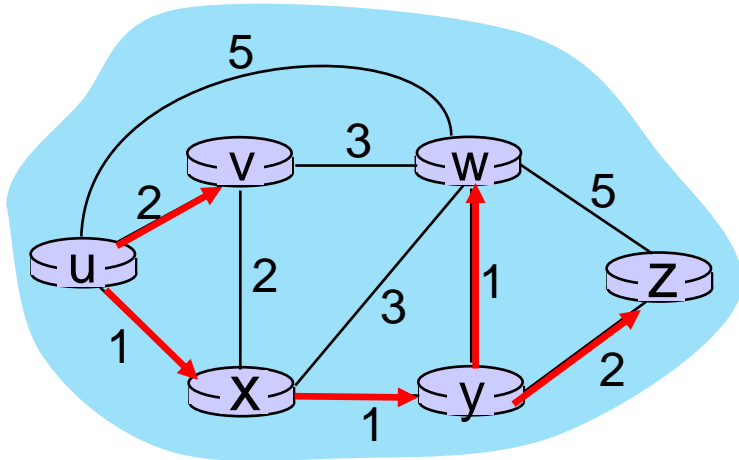
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

		v	w	x	y	z
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					



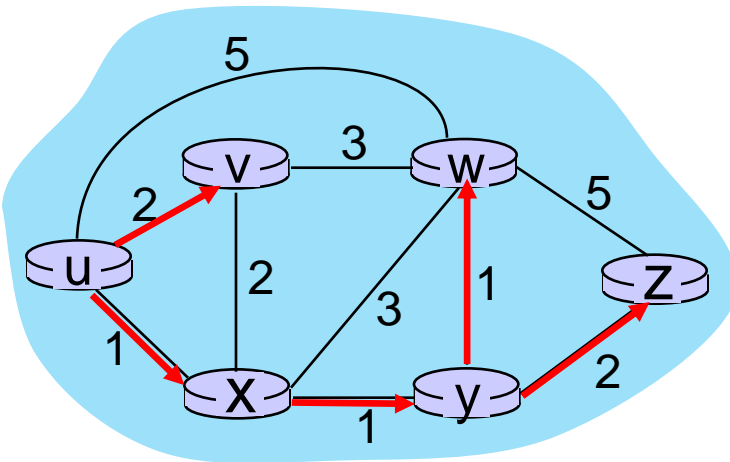
8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 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

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

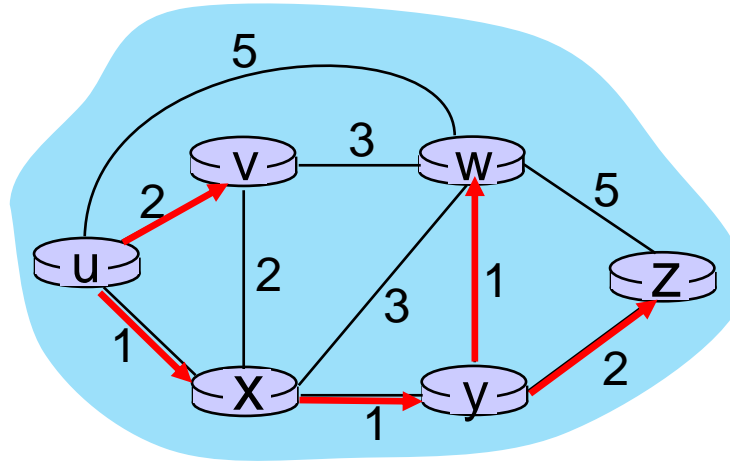
find a not in N' such that $D(a)$ is a minimum

add a to N'

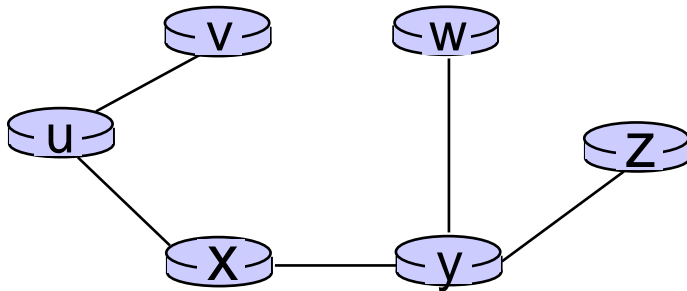
update $D(b)$ for all b adjacent to a and not in N' :

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

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

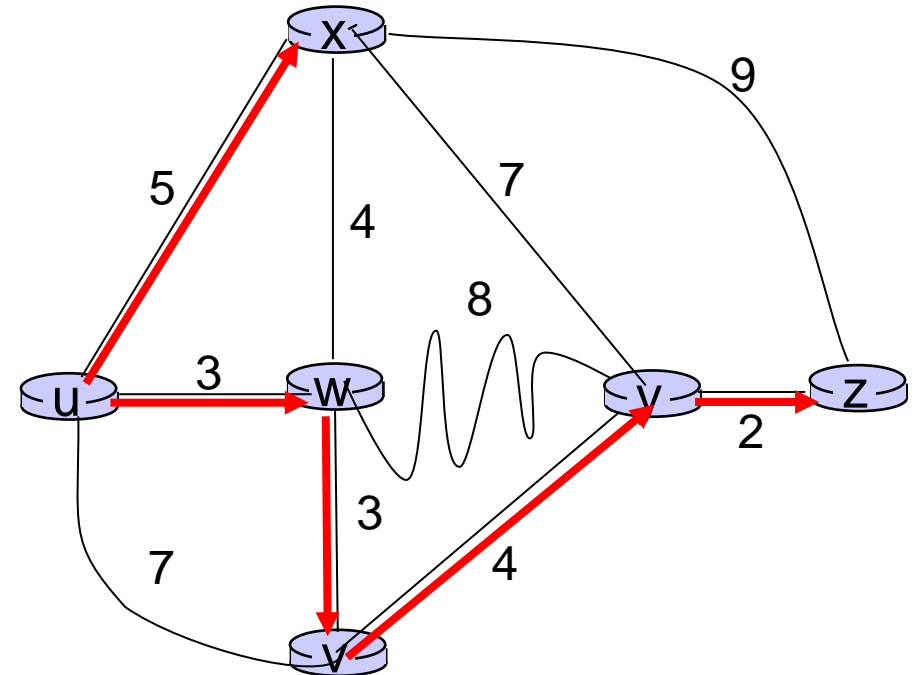
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from u to v directly

route from u to all other destinations via 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

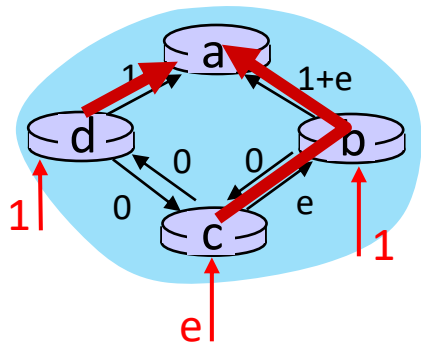
- each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

message complexity:

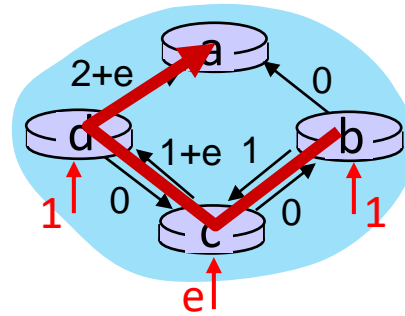
- each router must *broadcast* its link state information to other n routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

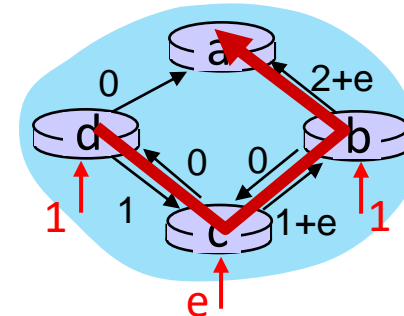
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



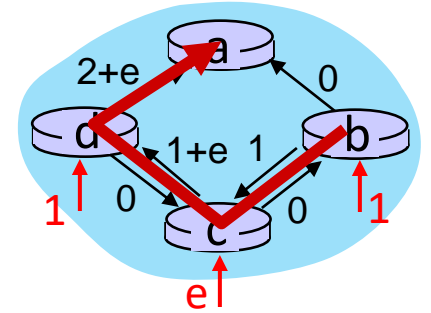
initially



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs

Distance Vector

Distance vector algorithm

key idea:

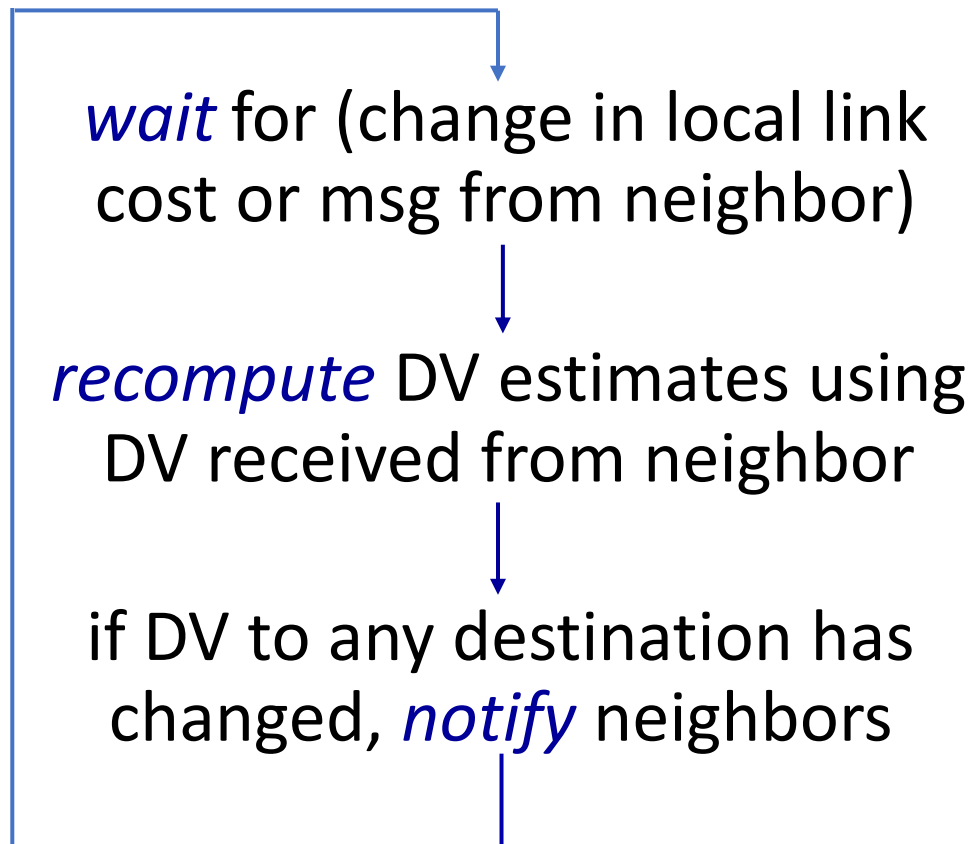
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

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

Distance vector algorithm:

each node:



iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

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

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

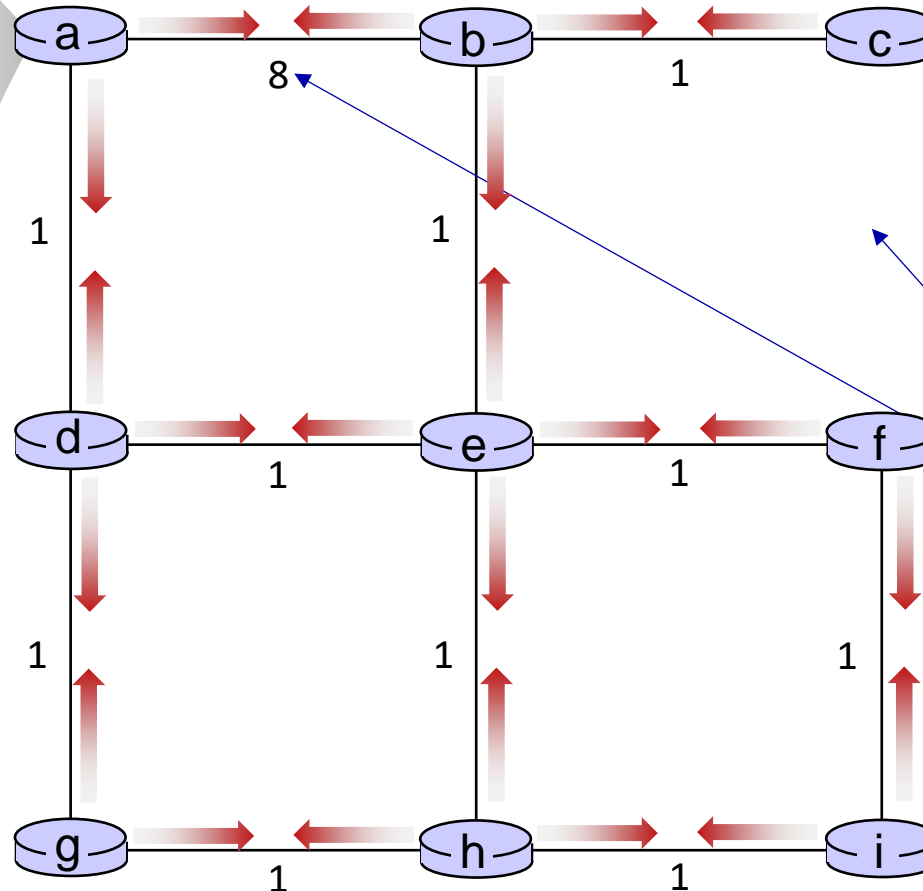
Distance vector: example



t=0

- 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

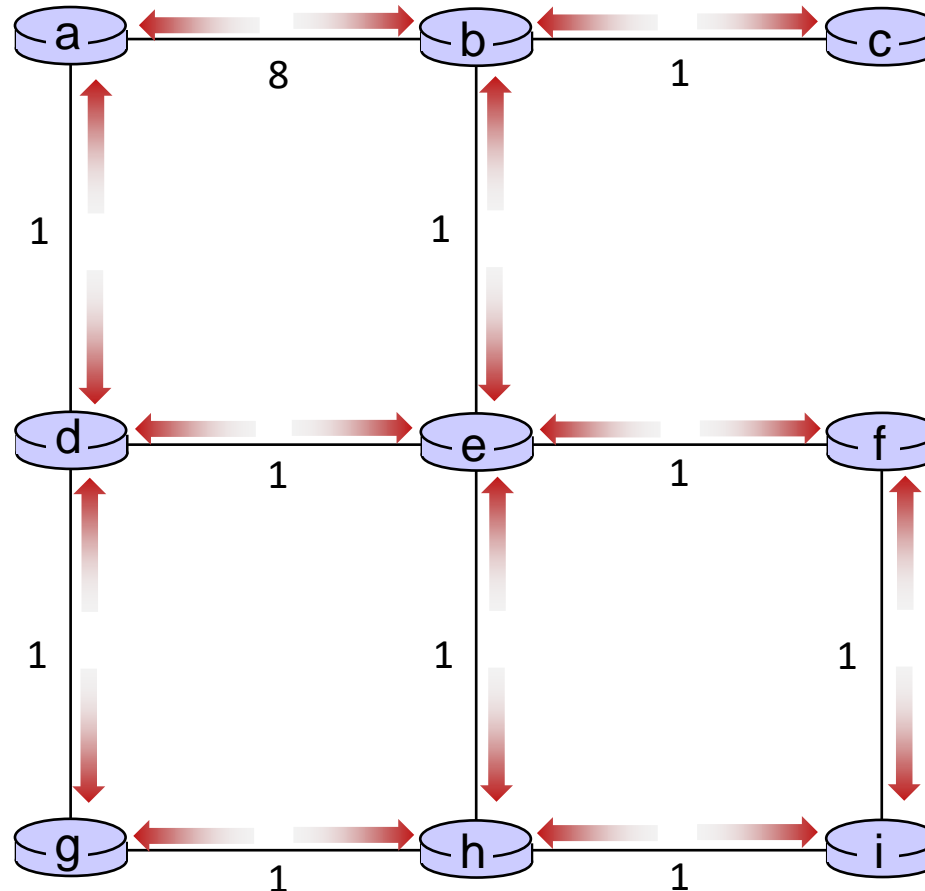
Distance vector example: iteration



$t=1$

All nodes:

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



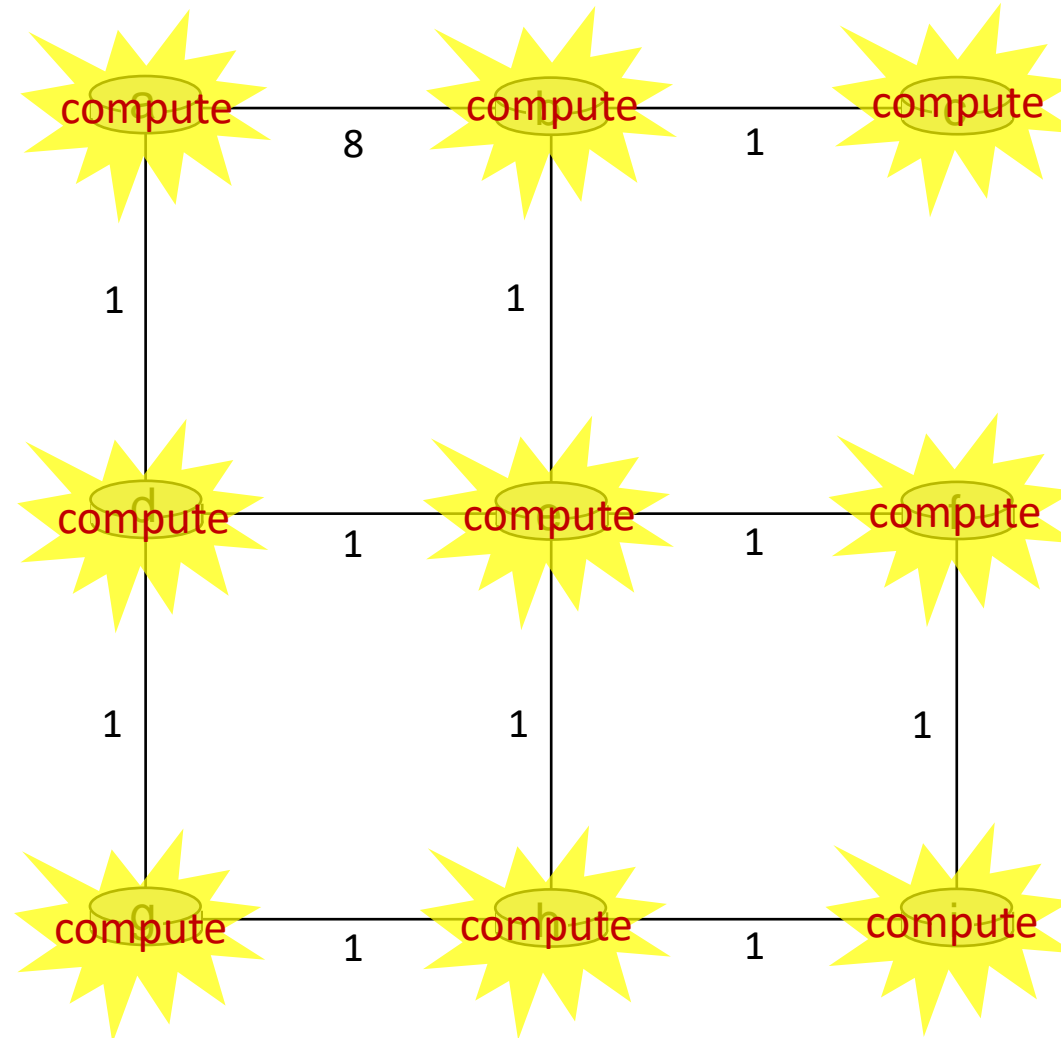
Distance vector example: iteration



$t=1$

All nodes:

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



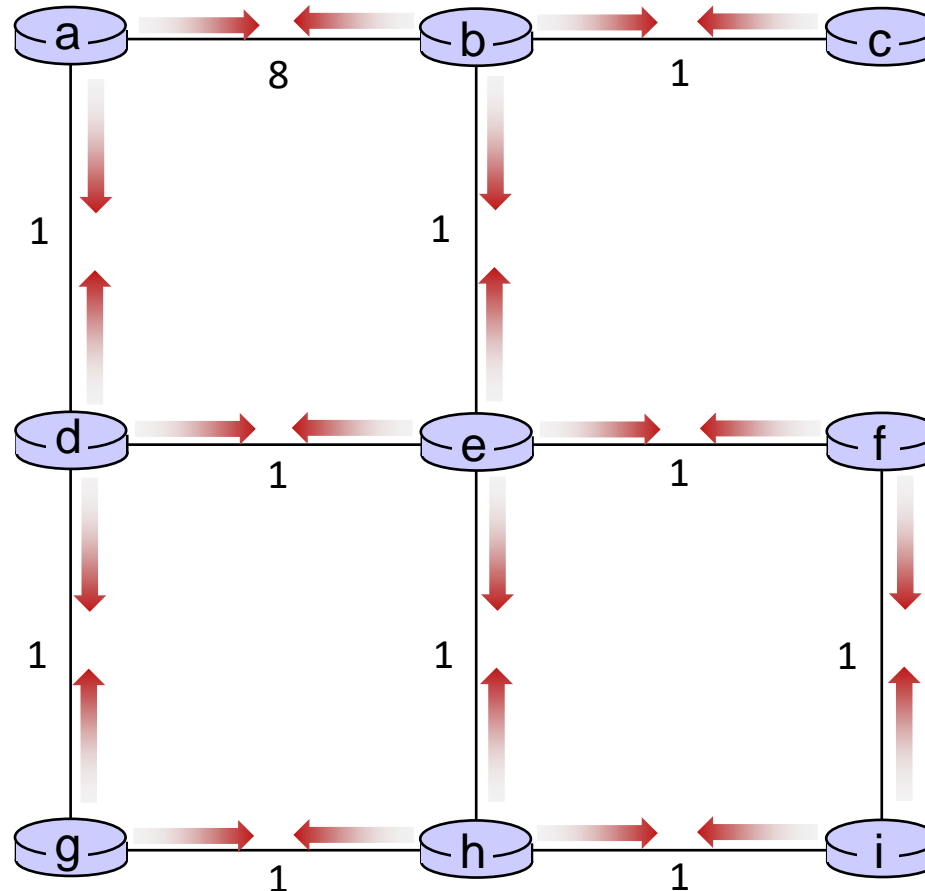
Distance vector example: iteration



$t=1$

All nodes:

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



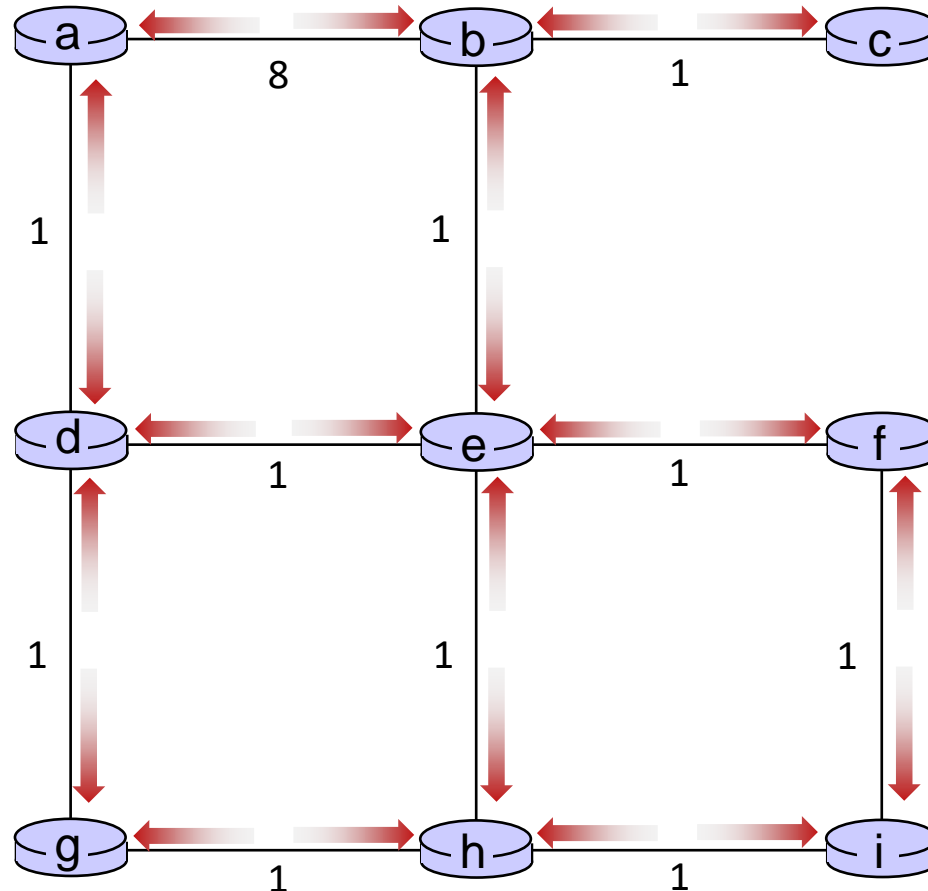
Distance vector example: iteration



$t=2$

All nodes:

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



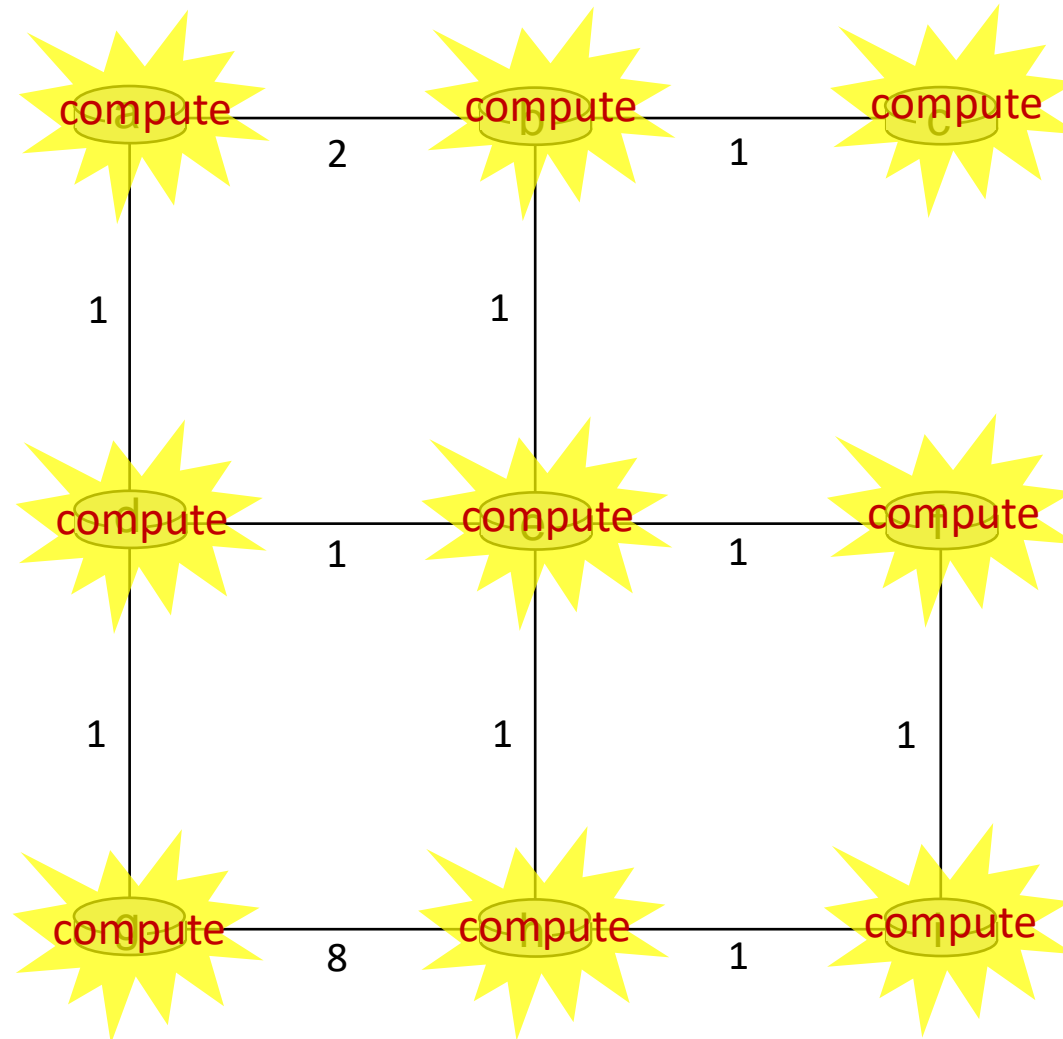
Distance vector example: iteration



t=2

All nodes:

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



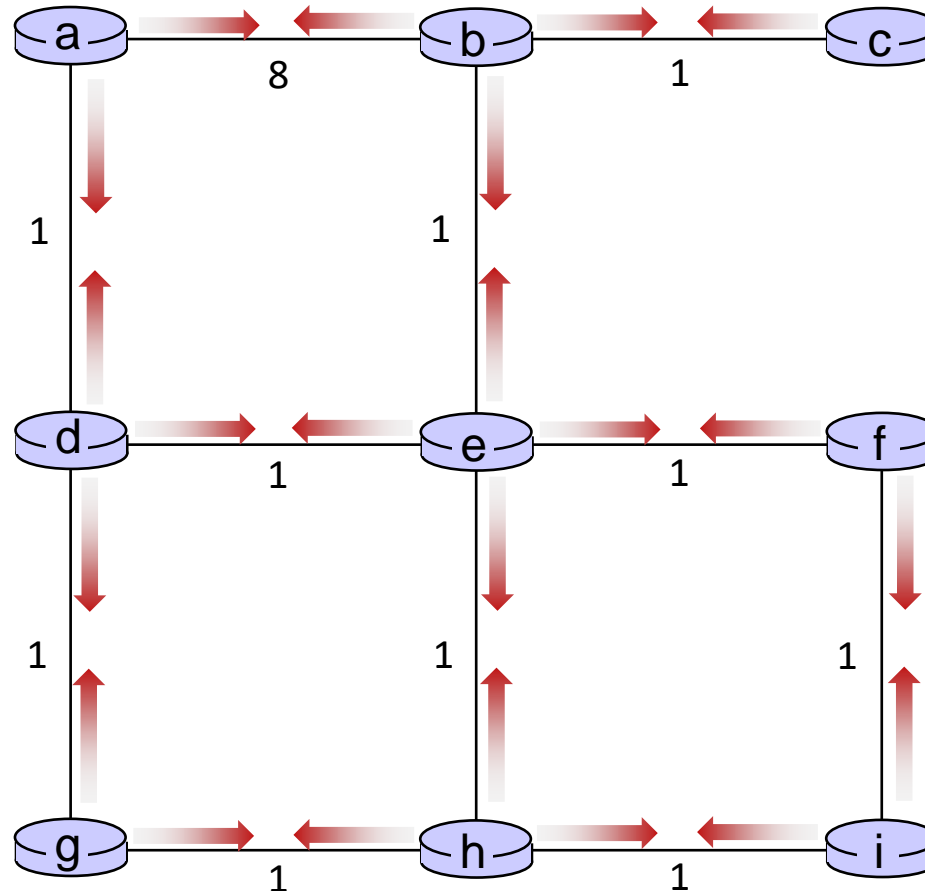
Distance vector example: iteration



t=2

All nodes:

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



Distance vector example: iteration

.... and so on

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

Distance vector example:



t=1

- b receives DVs from a, c, e

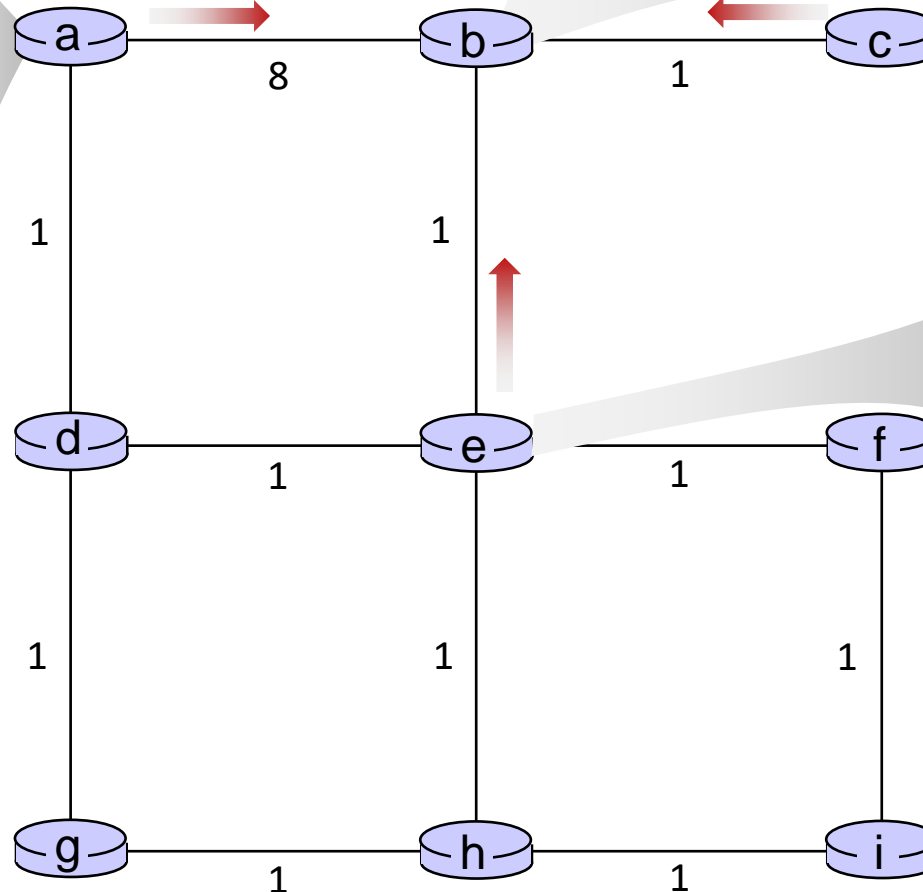
DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

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

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

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



Distance vector example:

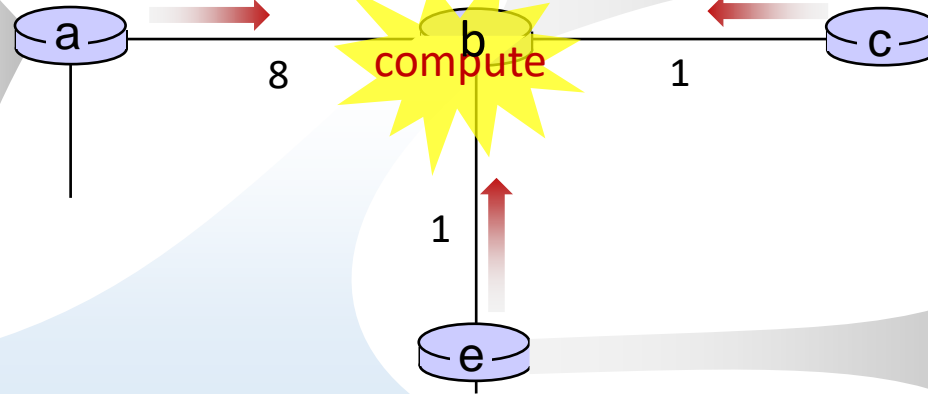


t=1

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

$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{9, 2, \infty\} = 2 \\
 D_b(e) &= \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1 \\
 D_b(f) &= \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(g) &= \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty \\
 D_b(h) &= \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(i) &= \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty
 \end{aligned}$$

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:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

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

DV in b:

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

Distance vector example:



t=1

- c receives DVs from b

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

DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

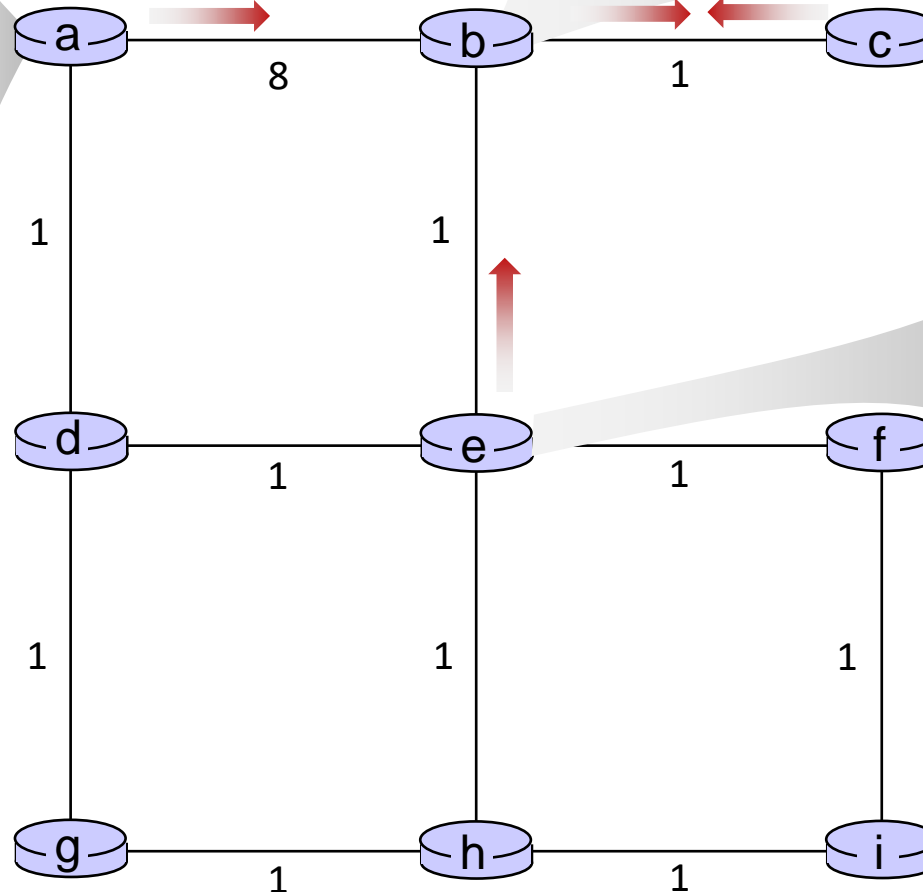
$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

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

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



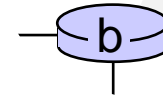
Distance vector example:



t=1

- c receives DVs from b computes:

$$\begin{aligned} D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\ D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\ D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\ D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\ D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\ D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\ D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\ D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty \end{aligned}$$



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

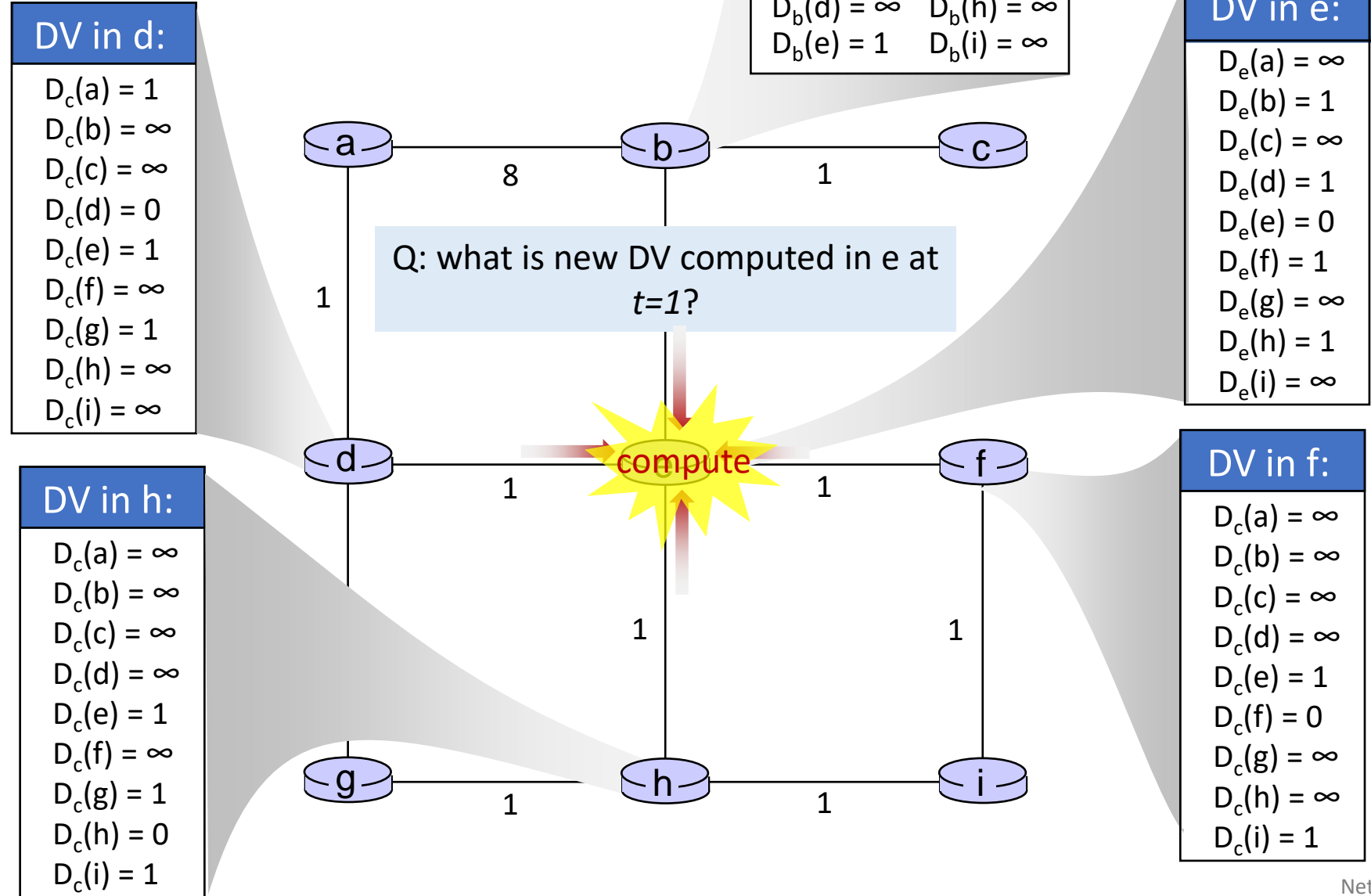
* Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance vector example:








t=1

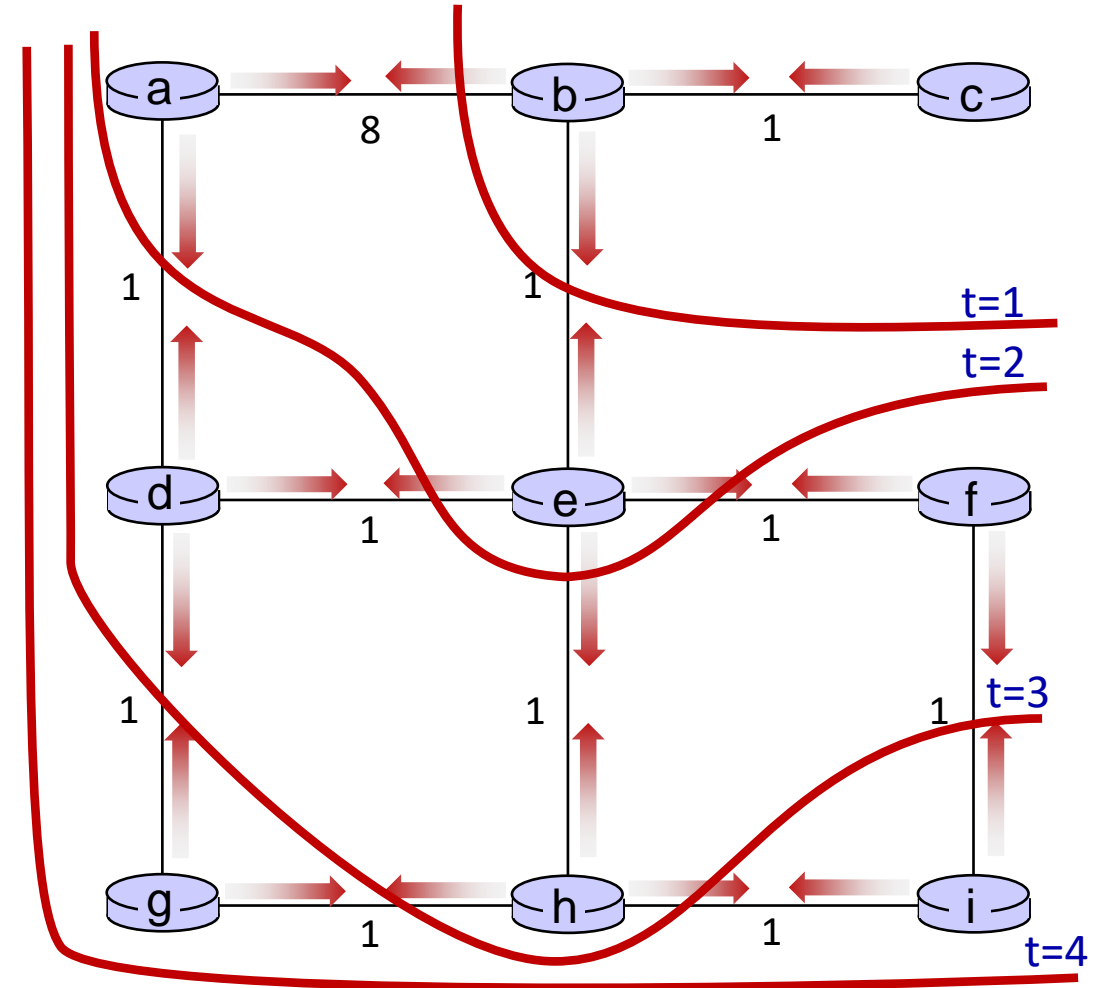
- e receives DVs from b, d, f, h



Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

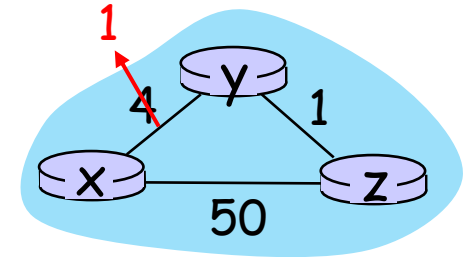
-  $t=0$ c's state at $t=0$ is at c only
-  $t=1$ c's state at $t=0$ has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  $t=2$ c's state at $t=0$ may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  $t=3$ c's state at $t=0$ may influence distance vector computations up to **3** hops away, i.e., at d, f, h
-  $t=4$ c's state at $t=0$ may influence distance vector computations up to **4** hops away, i.e., at g, i



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 DV, computes new least cost to x, sends its neighbors its DV.

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

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- “bad news travels slow” – count-to-infinity problem:
 - y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
 - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
 - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
 - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
 - ...
- see text for solutions. *Distributed algorithms are tricky!*

