



**BITS Pilani**

Pilani Campus

# Computer Networks (CS F303)

Virendra Singh Shekhawat  
Department of Computer Science and Information Systems



**BITS Pilani**  
Pilani Campus

# **Second Semester 2020-2021**

## **Module-4 <Network Layer>**

# Agenda



- **Routing Algorithms**
  - Dijkstra (Link State Routing) and Bellman ford (Distance Vector Routing)
- **Routing Protocols**
  - OSPF, RIP

# OSPF Protocol

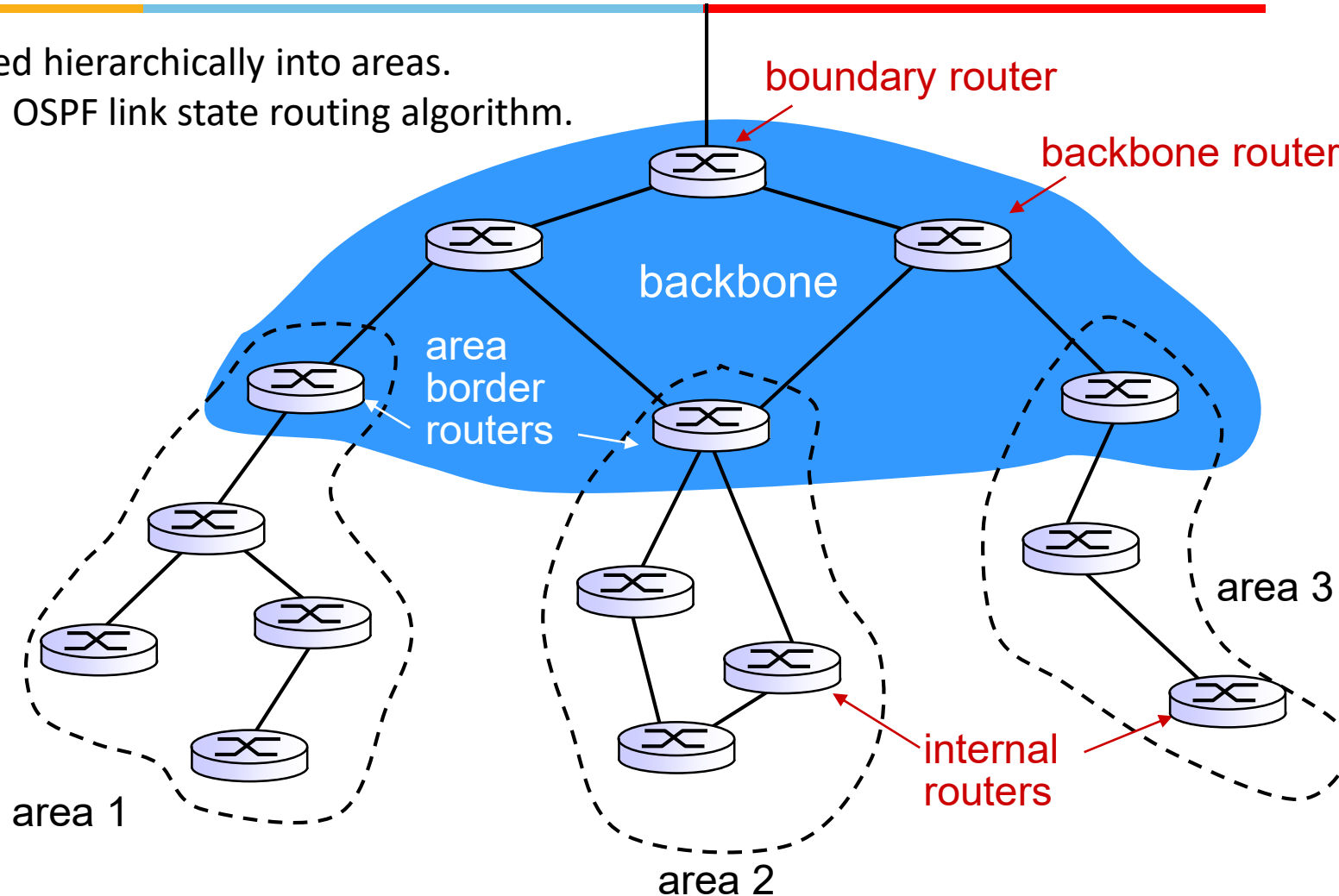


- “open”: publicly available
- Uses link state algorithm
  - LS packet dissemination
  - Topology map at each node
  - Route computation using Dijkstra’s algorithm
- OSPF advertisement carries one entry per neighbor
- Advertisements flooded to *entire AS*
  - Carried in OSPF messages directly over IP (rather than TCP or UDP)
  - Link state broadcast and reliable message transfer must be implemented in the OSPF itself
  - Broadcasts LSA whenever there is a change in link’s state and also send periodic updates (after every 30 mins)

# Hierarchical OSPF Routing



An AS can be configured hierarchically into areas.  
Each area runs its own OSPF link state routing algorithm.



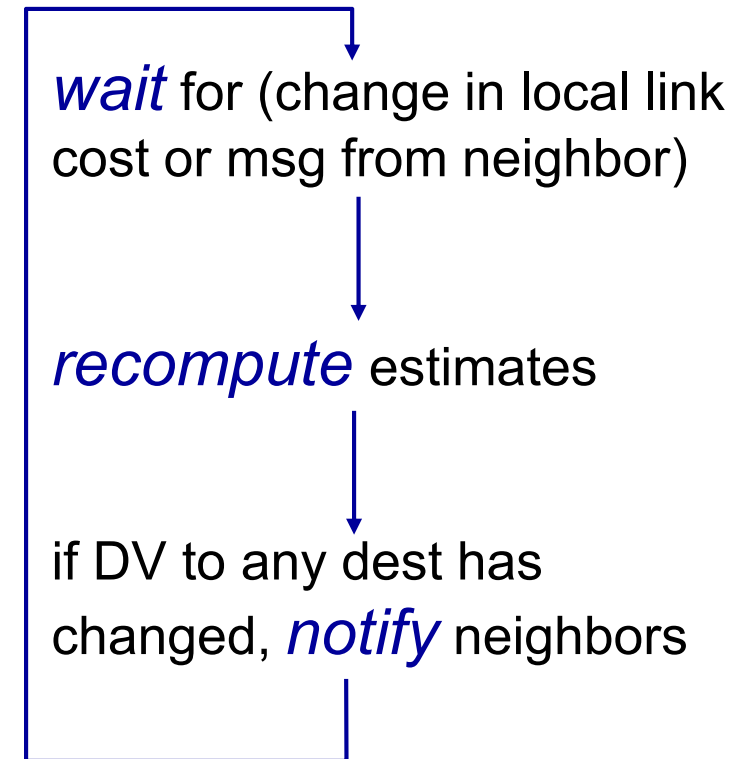
# OSPF Messages



- **HELLO**
  - To check whether links are operational or not
- **Database Description**
  - contain descriptions of the topology of the AS or area
- **Link State Request**
  - used by one router to request updated information about a portion of the Link State Database Description (LSDB) from another router
- **Link State Update**
  - contain information about an updated portion of the LSDB. These messages are sent in response of a Link State Request message
- **Link State Acknowledgement**
  - acknowledges a Link State Update message

# Distance Vector (DV) Routing

- **Distributed**
  - Node receives some information from its one or more neighbors
- **Iterative**
  - Process continues until no more info is exchanged
- **Asynchronous**
  - Does not require nodes to operate in lockstep manner



# Distance Vector Routing Algorithm

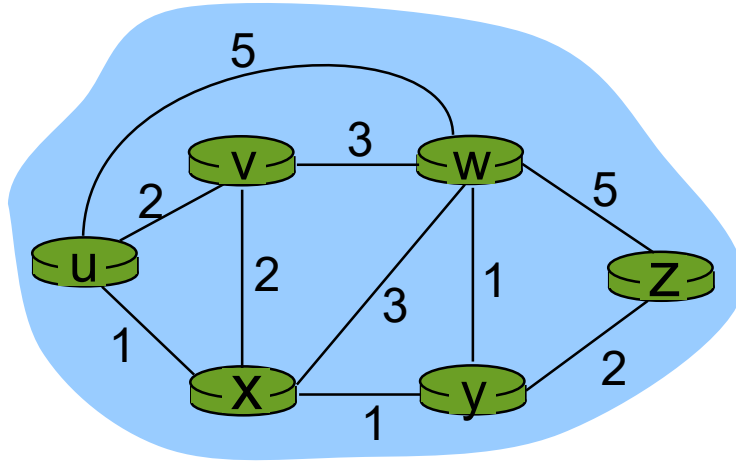
- From time-to-time, each node sends its own distance vector estimate to neighbors
  - DV contains estimate of its cost to all destinations in the network
- When x receives new DV estimate from neighbor, it updates its own DV using B-F eq.:

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

- Eventually, estimate  $D_x(y)$  converge to the actual least cost  $d_x(y)$



# Bellman-Ford Example



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

Cost of least cost path from u to z

$$d_u(z) = ???$$

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

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

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	∞	∞	∞

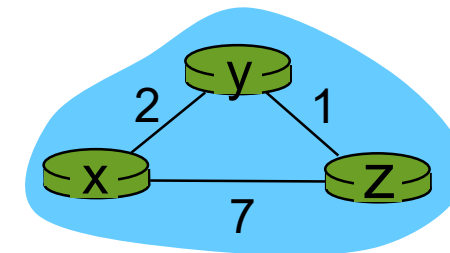
node z  
table

		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

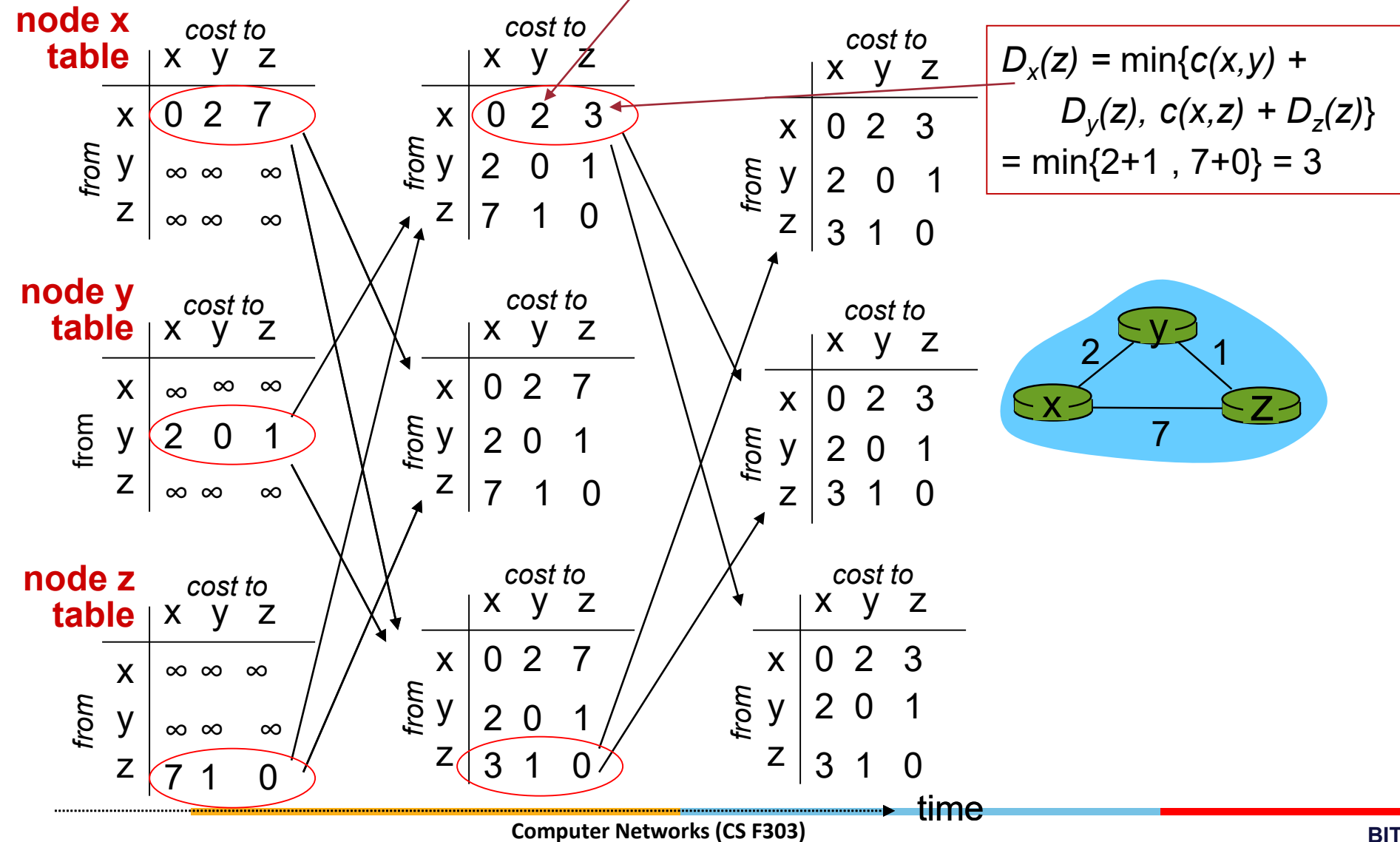
$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

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



time

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



# DV Algorithm

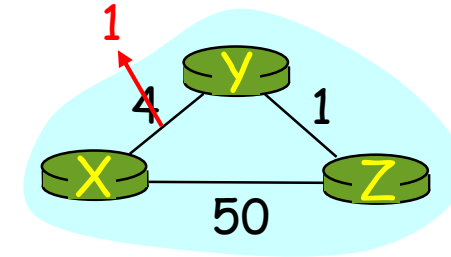
```
1  Initialization:
2    for all destinations y in N:
3       $D_x(y) = c(x,y)$  /* if y is not a neighbor then  $c(x,y) = \infty$  */
4    for each neighbor w
5       $D_w(y) = ?$  for all destinations y in N
6    for each neighbor w
7      send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to w
8
9  loop
10   wait (until I see a link cost change to some neighbor w or
11         until I receive a distance vector from some neighbor w)
12
13   for each y in N:
14      $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16   if  $D_x(y)$  changed for any destination y
17     send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to all neighbors
18
19  forever
```

# Distance Vector: Link Cost Changes [.1]



## Link cost changes:

- Node detects local link cost change
- Updates the distance table
- If cost change in least cost path, notify neighbors

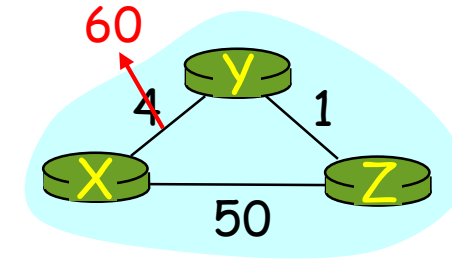


# Distance Vector: Link Cost Changes [..2]



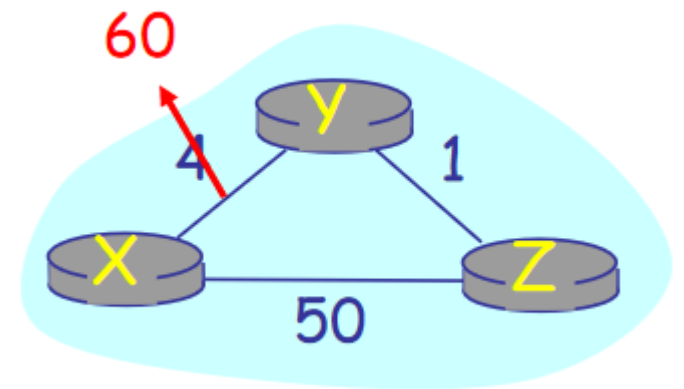
## Link cost changes:

- Bad news travels slow - “count to infinity” problem!



# Poisson Reverse and Split Horizon

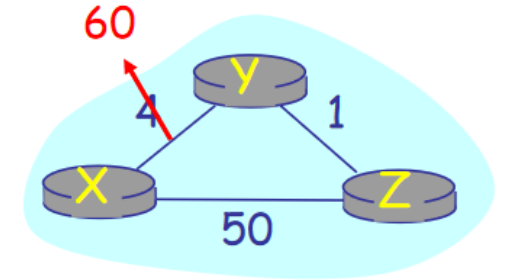
- **Split Horizon** rule states that a route can't be advertised out of the interface if the next hop for the advertised route is found on that interface.
- **Poison Reverse** rule states that routes received via one interface have to be advertised back out from that interface with an unreachable metric



# Split Horizon with Poisson Reverse



- If Z routes through Y to get to X:
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)





# RIP ( Routing Information Protocol)

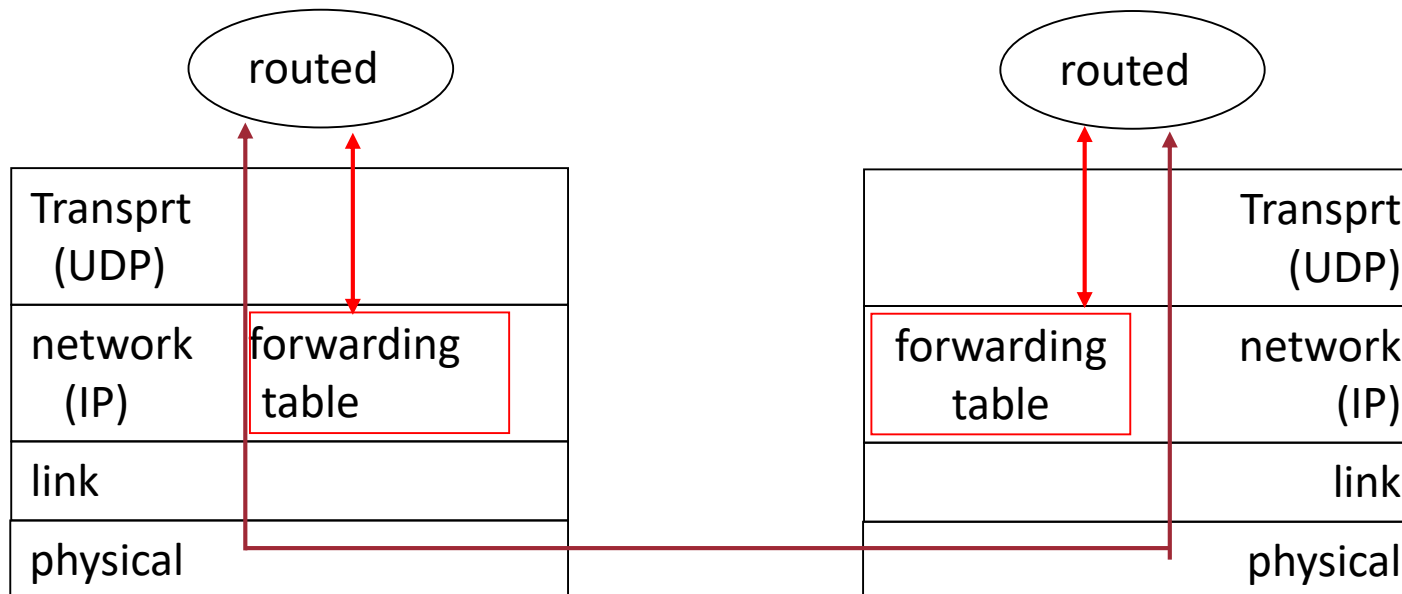


- Distance vector algorithm
- Included in BSD-UNIX Distribution in 1982
- Distance metric: # of hops (max = 15 hops)
- Distance vectors: exchanged among neighbors every 30 sec via Response Message (also called **advertisement**)
- Each advertisement: list of up to 25 destination nets within AS

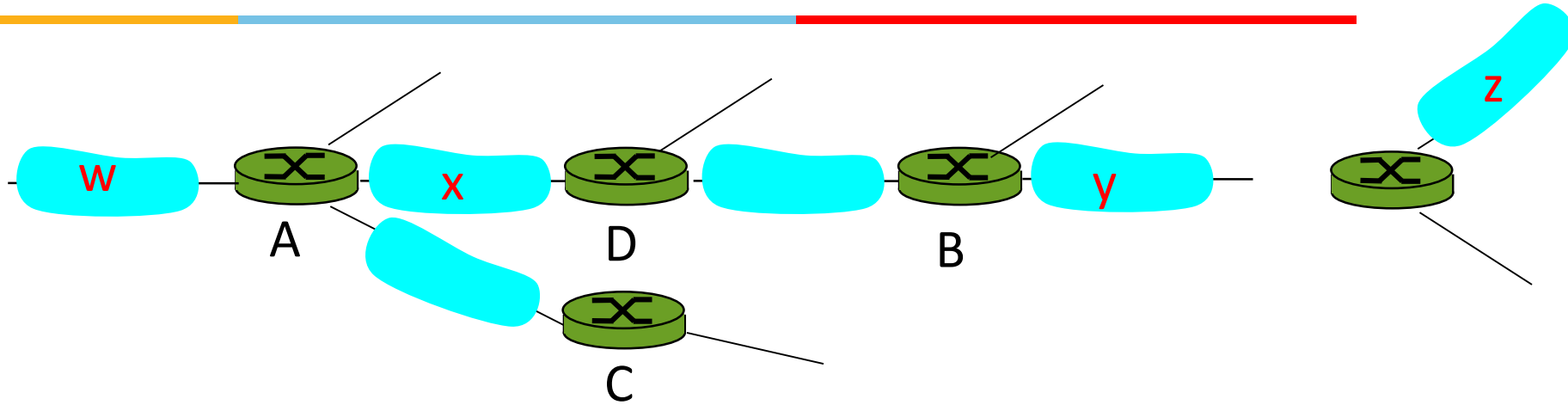
# RIP Table processing



- RIP routing tables managed by **application-level** process called route-d (daemon)
- Advertisements sent in UDP packets, periodically repeated



# RIP: Example



Destination Network	Next Router	Num. of hops to dest.
<b>W</b>	<b>A</b>	<b>2</b>
<b>Y</b>	<b>B</b>	<b>2</b>
<b>Z</b>	<b>B</b>	<b>7</b>
<b>X</b>	--	<b>1</b>
....	....	....

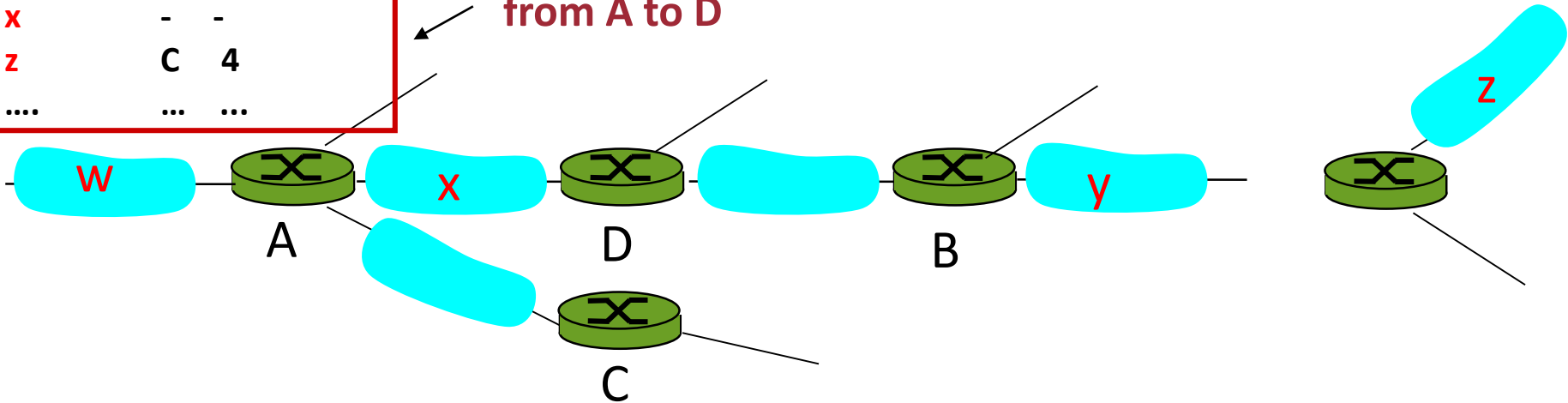
Routing table in D

# RIP: Example



Dest	Next hops	
w	-	-
x	-	-
z	C	4
....	...	...

Advertisement  
from A to D



Destination Network	Next Router	Num. of hops to dest.
w	A	2
y	B	2
z	<del>B</del> A	<del>7</del> 5
x	--	1
....	....	....

Routing table in D

# RIP: Link Failure and Recovery

If no advertisement heard after 180 sec --> neighbor/link declared dead

- Routes via neighbor invalidated
- New advertisements sent to neighbors
- Neighbors in turn send out new advertisements (if tables changed)
- Link failure info quickly propagates to entire net
- Poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)

# Weaknesses of RIP



- INFINITY defined as 15, thus RIP cannot be used in networks where routes are more than 15 hops
- Difficulty in supporting multiple metrics (default metric: # of hops)
  - The potential range for such metrics as bandwidth, throughput, delay, and reliability can be large
  - Thus the value for INFINITY should be large; but this can result in slow convergence of RIP due to count-to-infinity problem

# Internet Routing System: Two Tier



- **Inter-domain routing: Between ASes**
  - Routing policies based on *business relationships*
  - No common metrics, and limited cooperation
  - BGP: policy-based, path-vector routing protocol
- **Intra-domain routing: Within an AS**
  - Shortest-path routing based on *link metrics*
  - Routers are managed by a single institution
  - **OSPF and IS-IS**: link-state routing protocol
  - **RIP and EIGRP**: distance-vector routing protocol

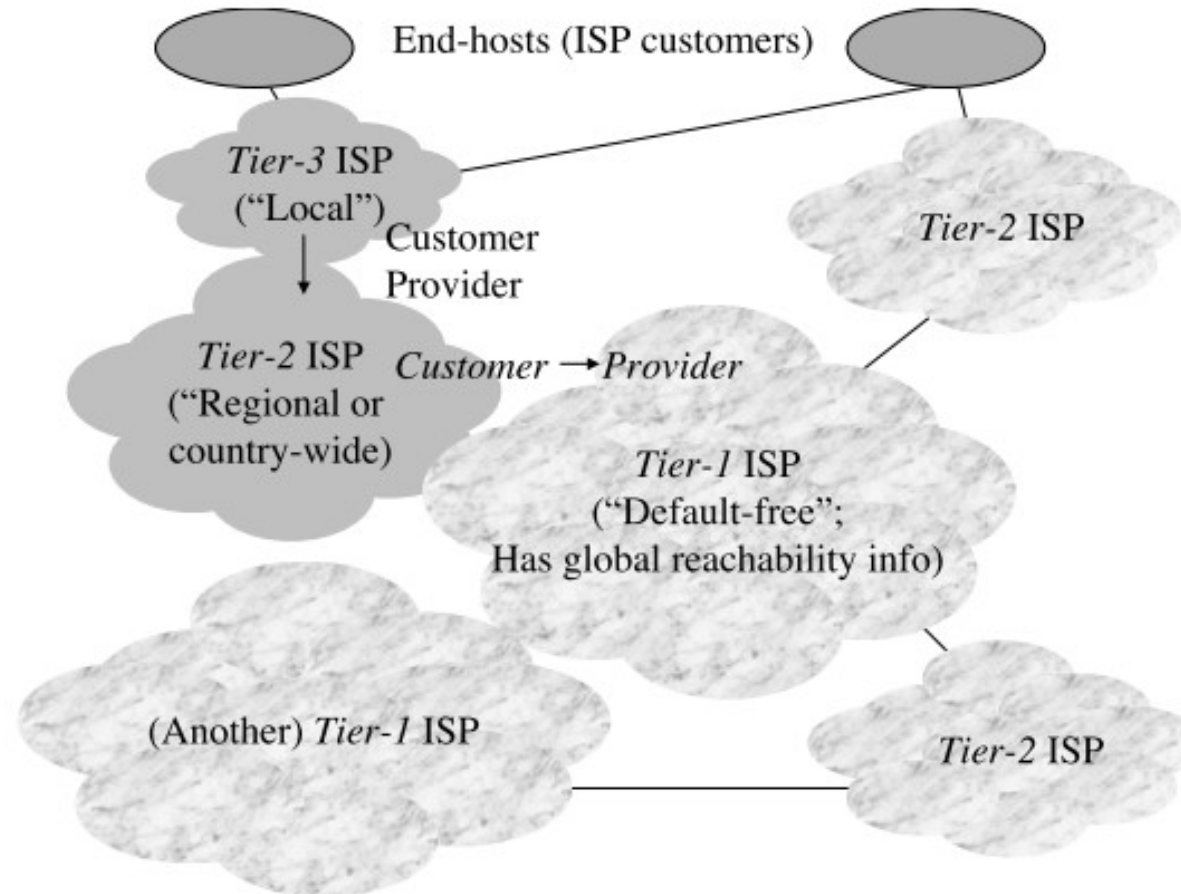
# Next...



- **BGP**
  - ASes, Policies
  - BGP Attributes
  - BGP Path Selection
  - I-BGP vs. E-BGP



# The BIG Picture



# Autonomous Systems (ASes)



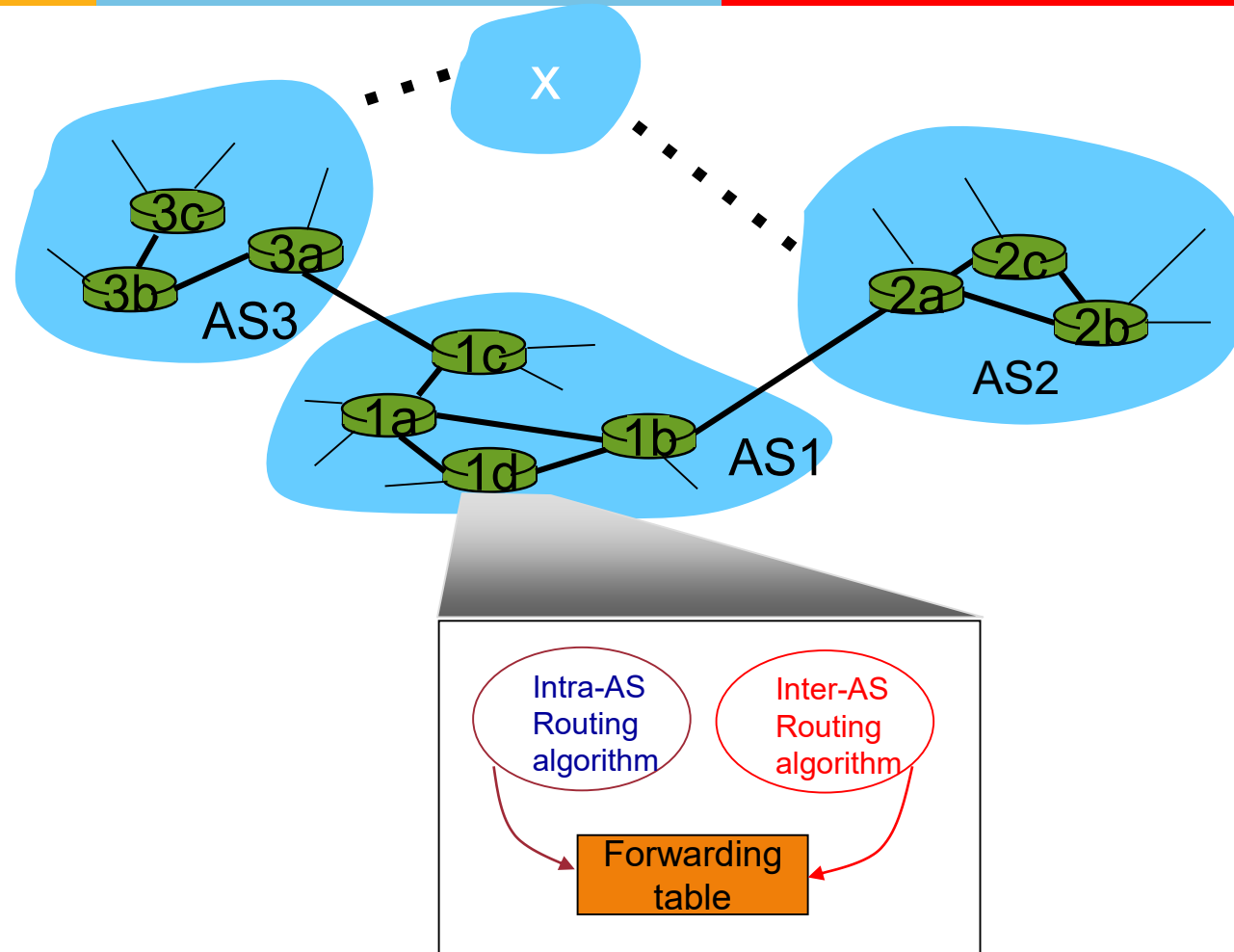
- **Autonomous system**
  - AS is an actual entity that participates in routing
  - Has an unique 16 bit ASN (now 32 bit [**RFC 4893 @ 2007**]) assigned to it and typically participates in inter-domain routing
- **Examples:**
  - MIT: 3, CMU: 9
  - AT&T: 7018, 6341, 5074, ...
  - UUNET: 701, 702, 284, 12199, ...
  - Sprint: 1239, 1240, 6211, 6242, ...

# Let's Find out...

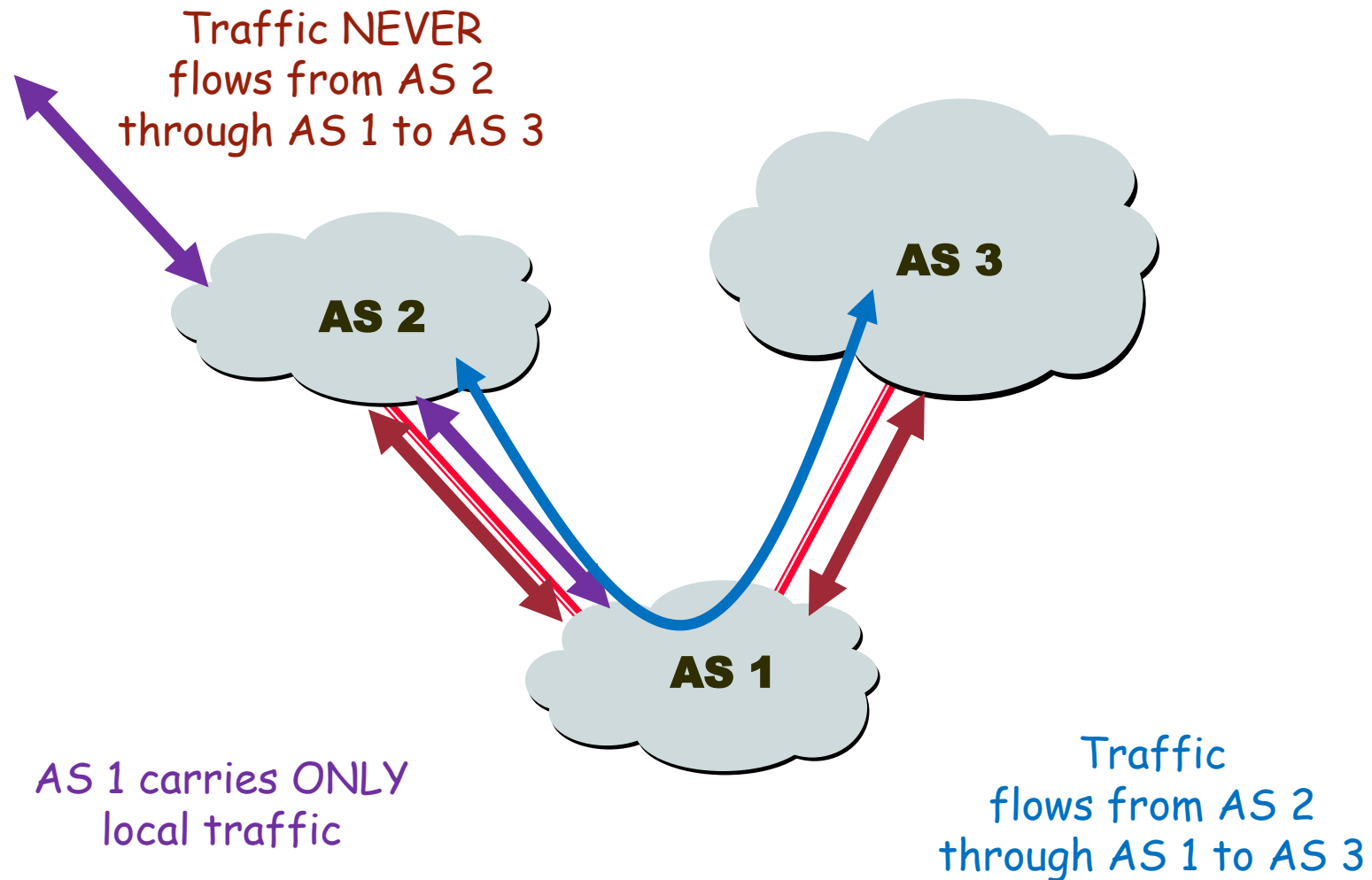


- How do ASes interconnect to provide global connectivity?
- How does routing information get exchanged?

# Interconnected ASes



# AS Categories [Stub/Multi-homed/Transit]



# Inter-domain Routing in the Internet

---

- Link state or distance vector?
- Problems with distance-vector:
  - Bellman-Ford algorithm may not converge
- Problems with link state:
  - Metric used by routers not the same – loops
  - LS database too large – entire Internet
  - May expose policies to other AS's

# Solution: Distance Vector with Path

---

- Each routing update carries the entire path
- Loops are detected as follows:
  - When AS gets route, check if AS already in path
    - If yes, reject route
    - If no, add self and (possibly) advertise route further

# BGP-4



- BGP = Border Gateway Protocol
- Is a Policy-Based routing protocol
- It is the EGP of today's global Internet
- Relatively simple protocol, but configuration is complex

## **1989 : BGP-1 [RFC 1105]**

- Replacement for EGP (1984, RFC 904)

## **1990 : BGP-2 [RFC 1163]**

## **1991 : BGP-3 [RFC 1267]**

## **1995 : BGP-4 [RFC 1771]**

## **2006: BGP-4 [RFC 4271]**

- Support for Classless Interdomain Routing (CIDR) , Route Aggregation

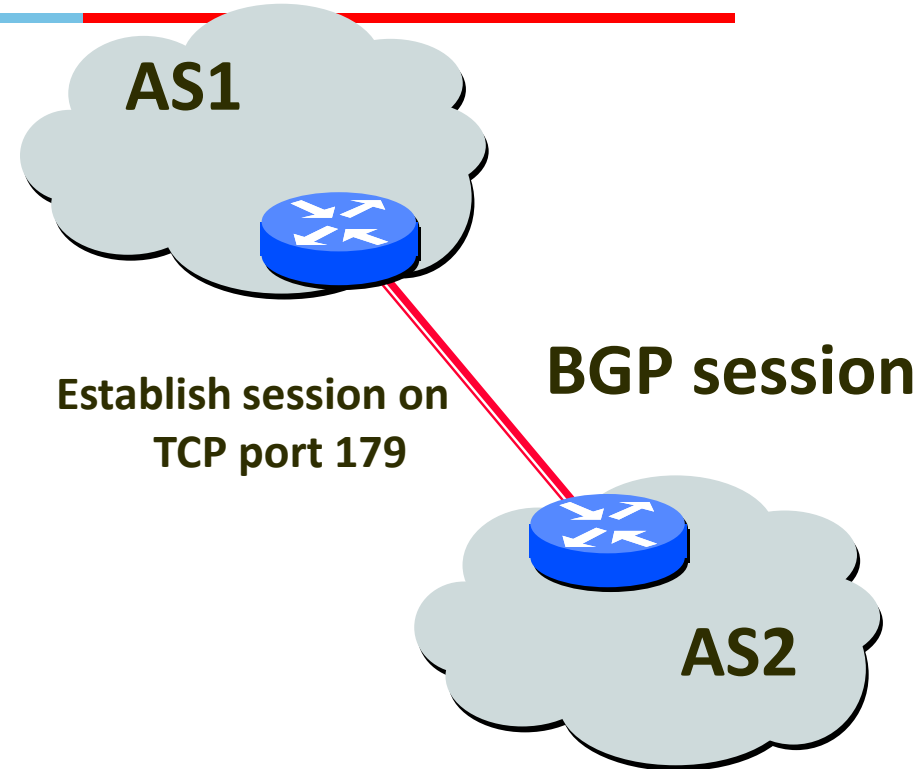


# BGP Operations



- Open** : Establish a peering session.
- Keep Alive** : Handshake at regular intervals.
- Notification** : Shuts down a peering session.
- Update** : Announcing new routes or withdrawing previously announced routes.

Route announcement =  
prefix + attributes values

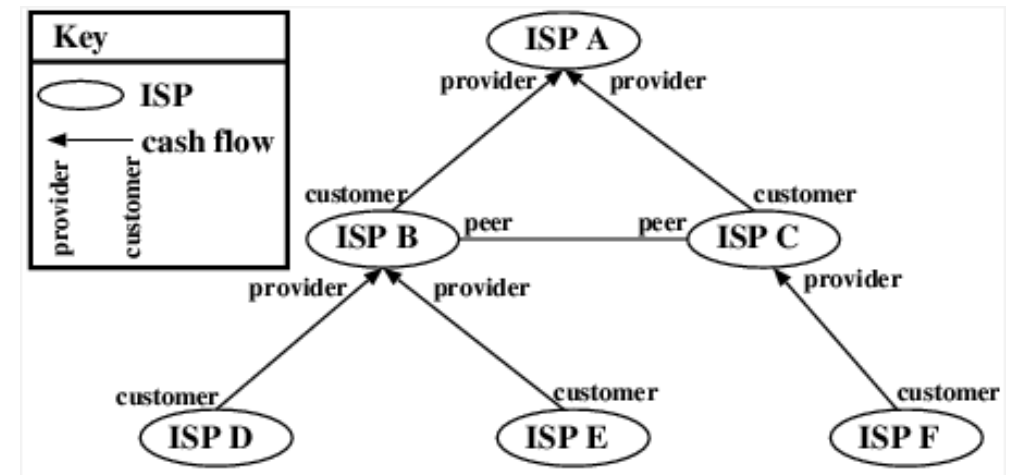


While connection  
is ALIVE exchange  
route UPDATE messages

# Fundamental Rules: BGP



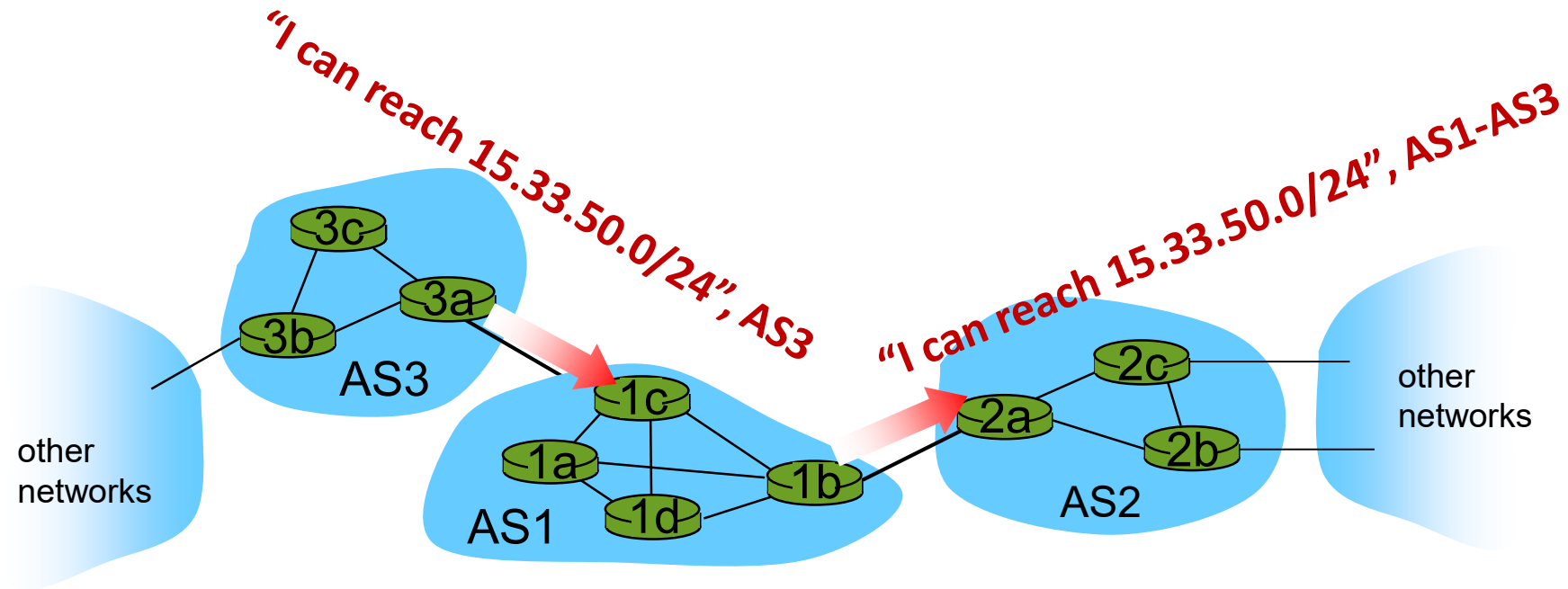
- BGP advertises to neighbors only those routes that it uses
  - Consistent with the hop-by-hop Internet paradigm
- No need for periodic refresh - routes are valid until withdrawn, or the connection is lost
- Incremental updates are possible



# Policy Decisions

- BGP provides capability for enforcing various policies
- BGP enforces policies by choosing paths from multiple alternatives and controlling advertisement to other AS's
- **Import policy**
  - What to do with routes learned from neighbors?
- **Export policy**
  - What routes to be announced to neighbors?
  - Depends on relationship with neighbors

# Distributing Path Information



# Export Policy



- Once the route is announced the AS is willing to transit traffic on that route
- **To Customers:** Announce all routes learned from **peers**, **providers** and **customers**, and self-origin routes
- **To Providers and Peers:** Announce routes learned from **customers** and self-origin routes

# How to implement export policies?

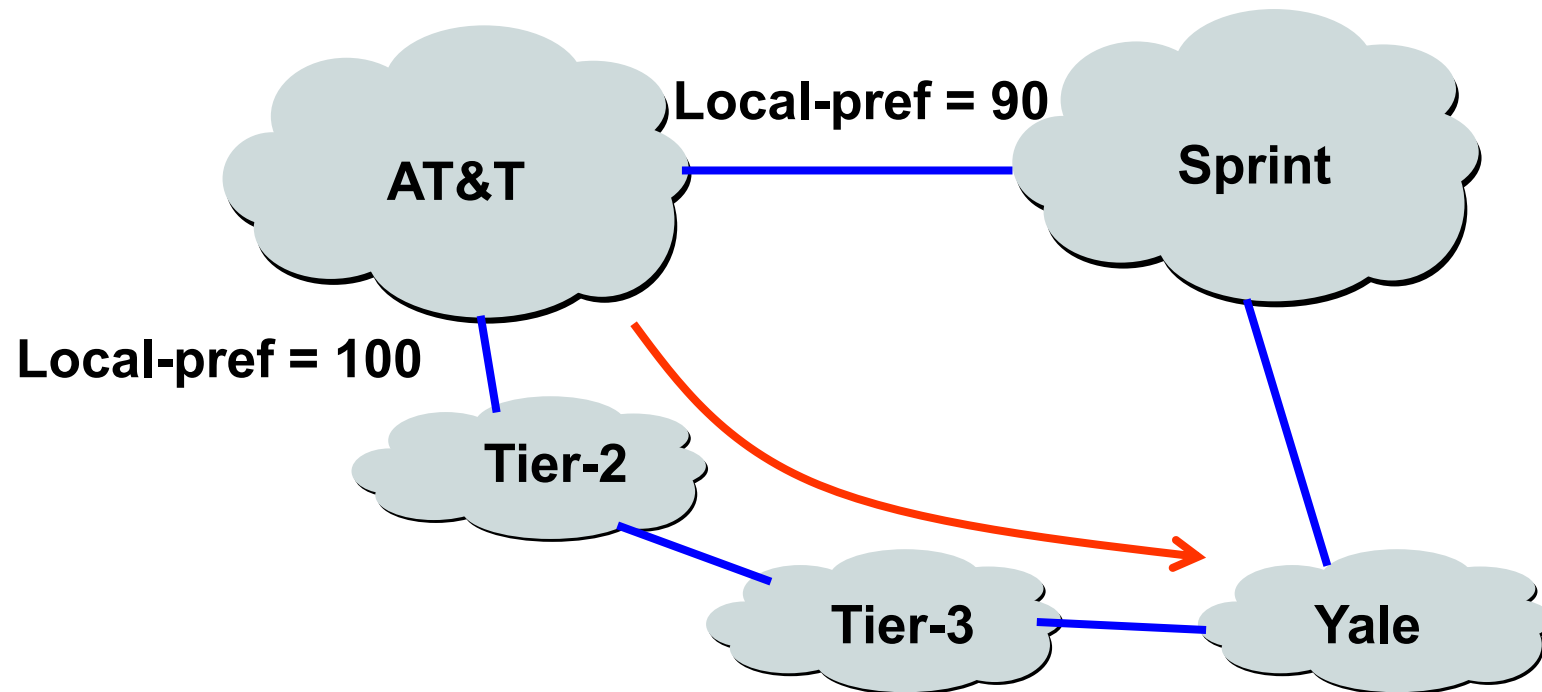


- **BGP Attributes**
  - Local Preference
  - AS-Path Length
  - MED (Multi Exit Discriminator)
  - NEXT-HOP

# Local Preference



- Used to choose outbound external **BGP** path.

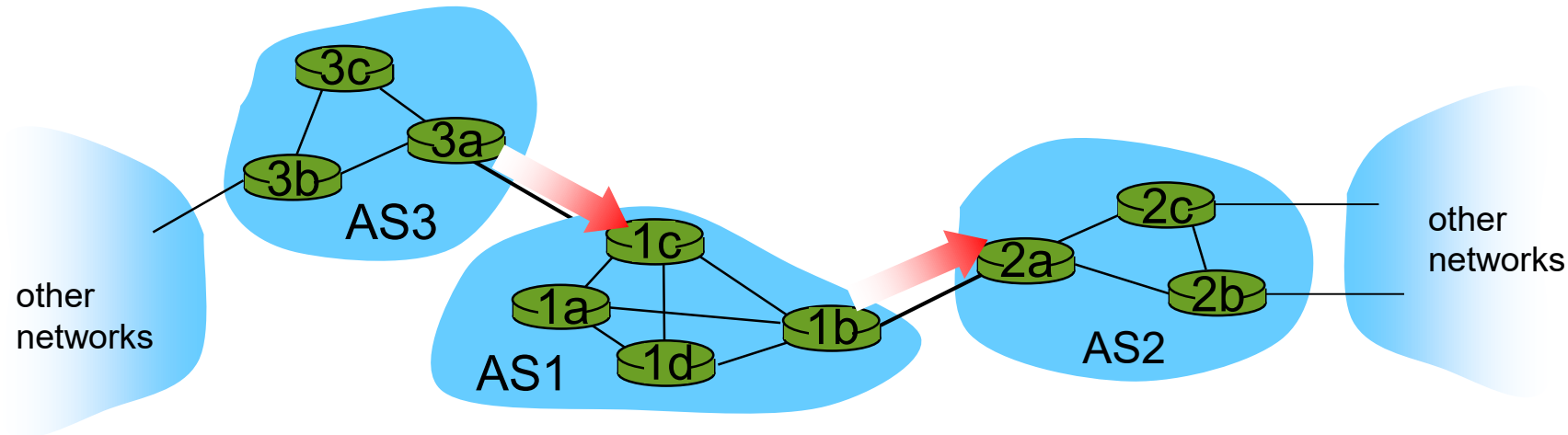


# NEXT-HOP



The **next hop** IP address that is going to be used to reach a certain destination.

**15.33.50.0/24, AS-PATH: AS3, NEXT-HOP: IP ADD of 3a-1c interface**



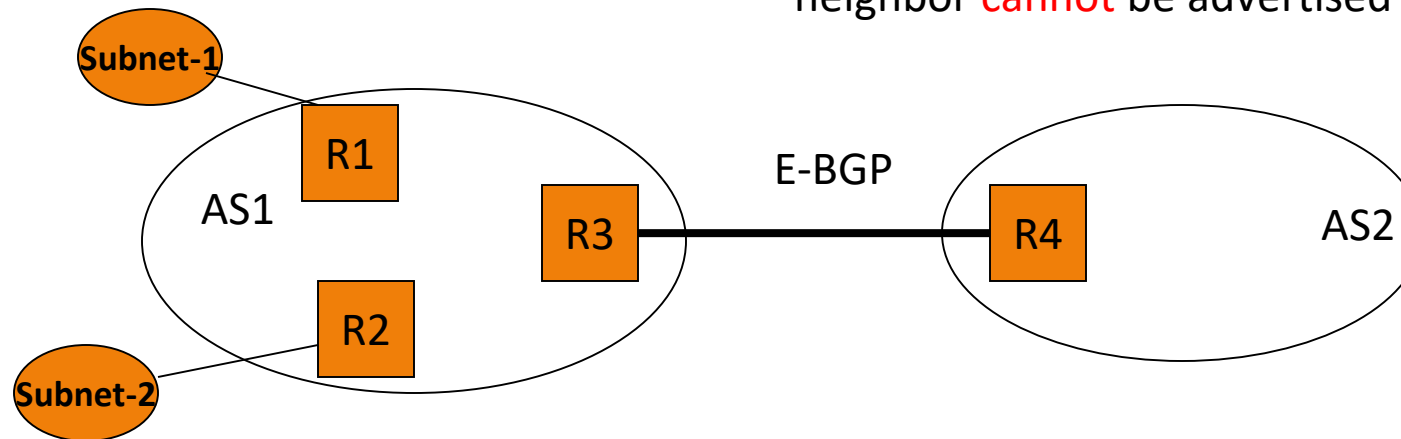


# Internal BGP vs. External BGP


- R3 and R4 can learn routes by using BGP
- How do R1 and R2 learn routes?
- Option 1: Inject routes in IGP
  - Only works for small routing tables
- Option 2: Use I-BGP

## Different rules about re-advertising prefixes in I-BGP

Prefix learned from E-BGP neighbor can be advertised to I-BGP neighbor and vice-versa, but Prefix learned from one I-BGP neighbor **cannot** be advertised to another I-BGP neighbor.



# Route Selection Process



Step	Attribute	Controlled by local or neighbor AS?
1.	Highest LocalPref	Local
2.	Lowest AS path length	Neighbor
3.	Lowest origin type	Neither
4.	Lowest MED	Neighbor
5.	eBGP-learned over iBGP-learned	Neither
6.	Lowest IGP cost to border router	Local
7.	Lowest router ID (to break ties)	Neither

Thank You!