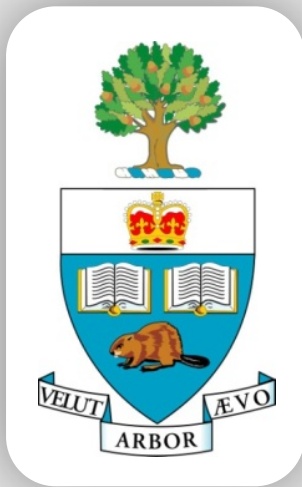


CSC 458/2209 – Computer Networks

Handout # 8: The Internet Protocol, Routing and Forwarding



Instructed by:

Joe Lim – L5101

joe.lim@utoronto.ca

Sajad Shirali-Shahreza – L0201

shirali@cs.toronto.edu

AbdulAziz AlHelali – L0101

a.alhelali@mail.utoronto.ca



Slides adapted from

Professor Yashar Ganjali

Announcements

- Don't forget the programming assignment.
 - Due: **Sunday Oct. 22 at 11:59pm.**
 - Take advantage of tutorials, and TA office hours.
 - Don't leave it to the last minute.
- Problem set 1 due this week
 - **Sunday Oct. 8th at 11:59 pm.**
 - Submit electronically on MarkUs.
 - File name: ps1.pdf.
- Midterm: Oct 17th 6:00 – 8:00PM
 - Location: TBA

The Story

- So far ...
 - Layers, and protocols
 - Link layer
 - Interconnecting LANs
 - Hubs, switches, and bridges
 - The Internet Protocol
 - IP datagram, fragmentation
 - Naming and addressing
 - CIDR, DNS
- This time
 - Routing and forwarding

| |
|--------------|
| Application |
| Presentation |
| Session |
| Transport |
| Network |
| Data Link |
| Physical |

Packet Routing and Forwarding

Forwarding IP datagrams

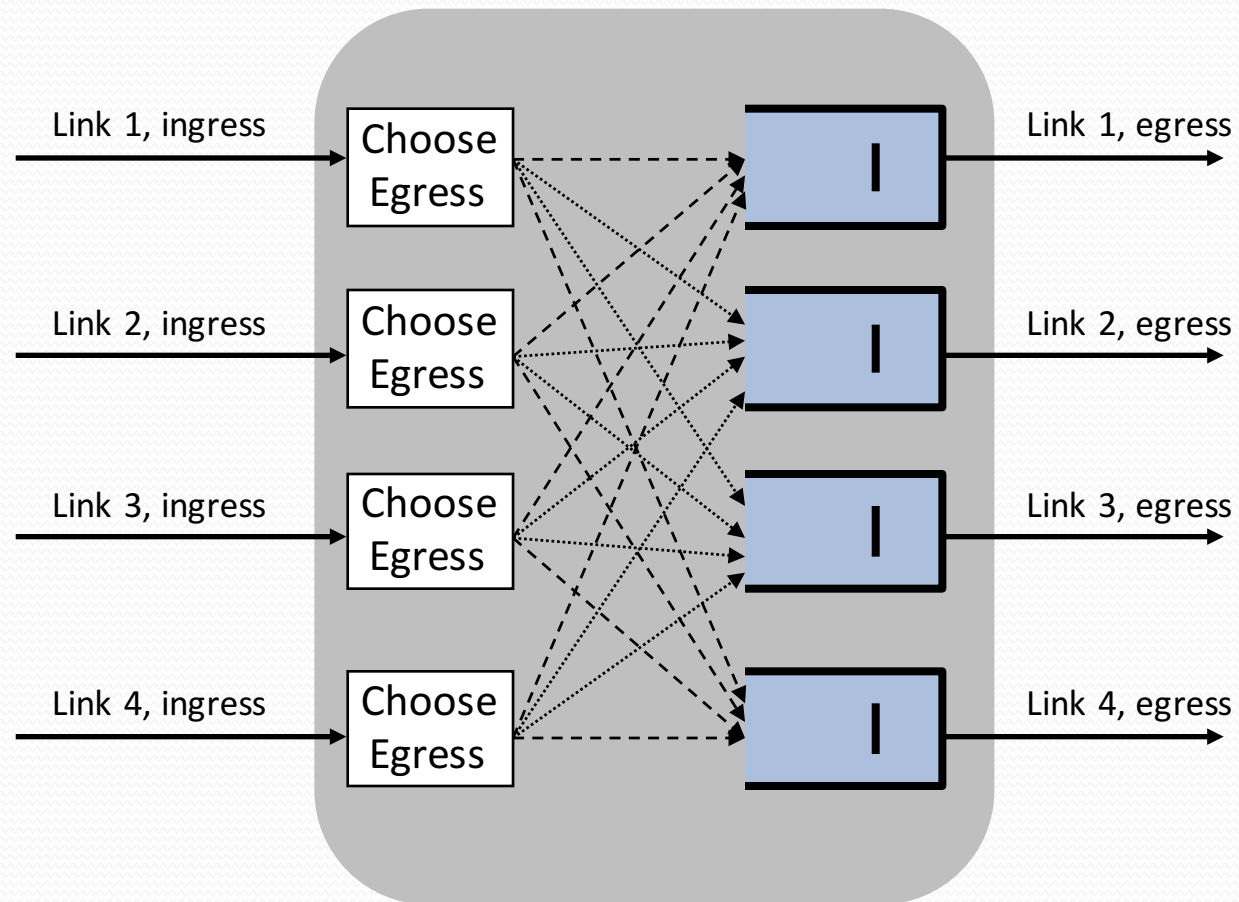
- Class-based vs. CIDR
- Routing Techniques
 - Naïve: Flooding
 - Distance vector: Distributed Bellman Ford Algorithm
 - Link state: Dijkstra's Shortest Path First-based Algorithm

Hop-by-Hop Packet Forwarding

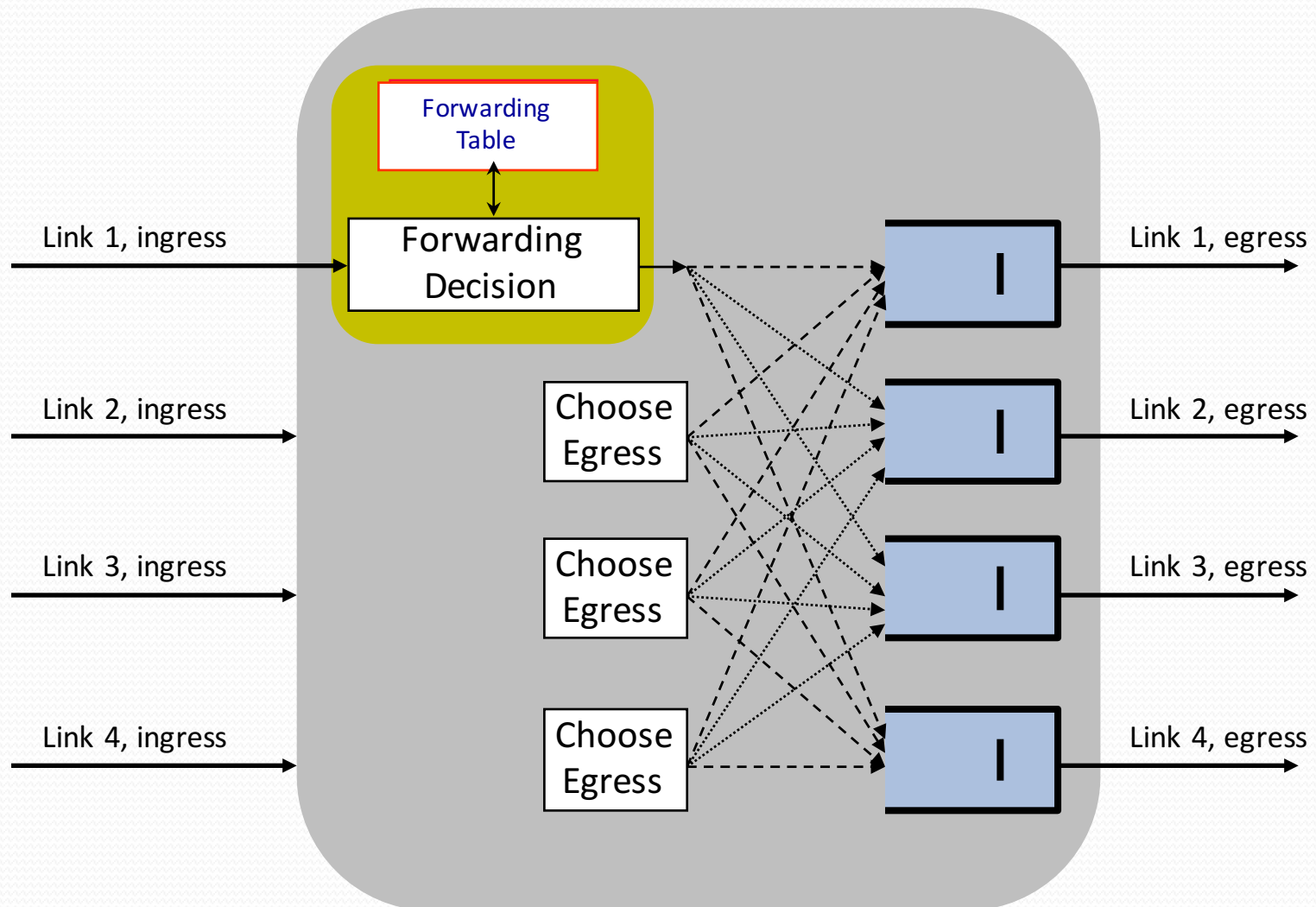
- Each router has a forwarding table
 - Maps destination addresses...
 - ... to outgoing interfaces
- Upon receiving a packet
 - Inspect the destination IP address in the header
 - Index into the table
 - Determine the outgoing interface
 - Forward the packet out that interface
- Then, the next router in the path repeats
 - And the packet travels along the path to the destination



Inside a Router



Inside a Router



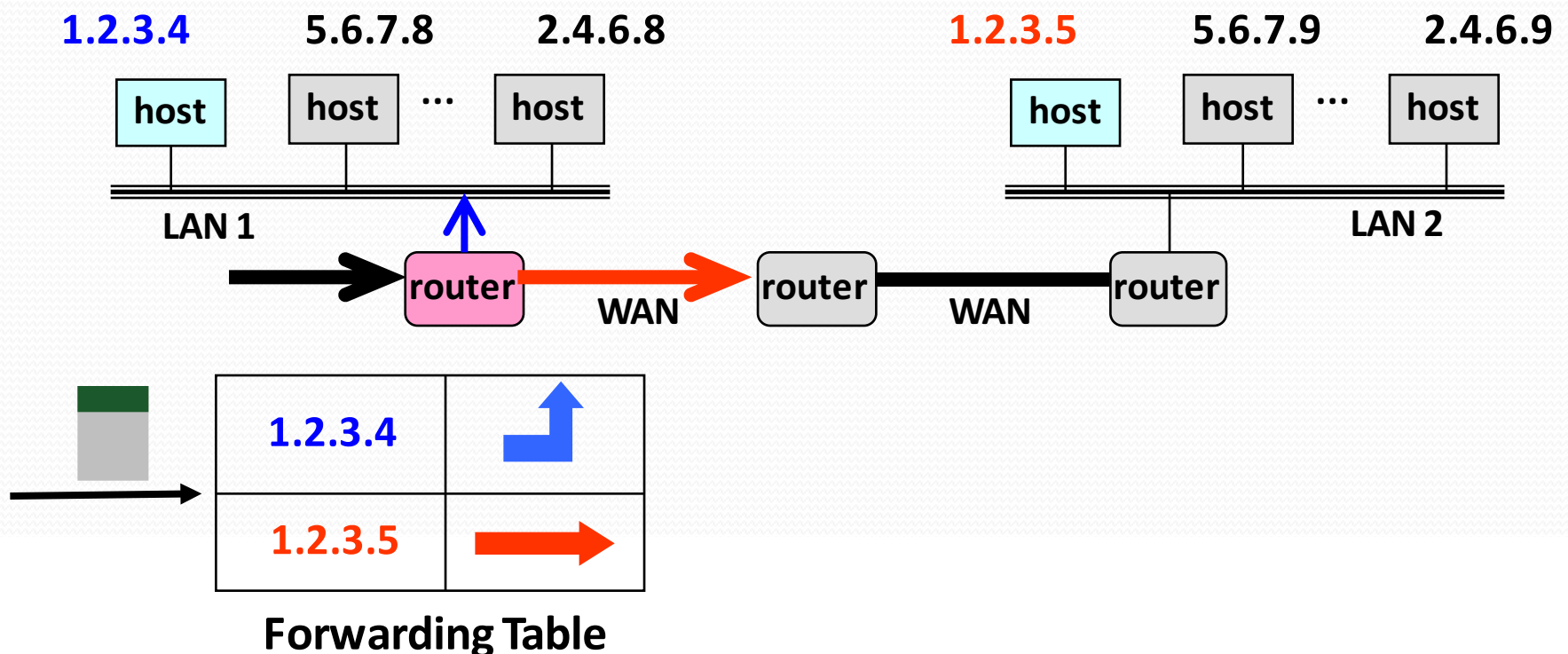
Forwarding in an IP Router

- Lookup packet DA in forwarding table. some hardware solution to lookup the table
 - If known, forward to correct port.
 - If unknown, drop packet.
- Decrement TTL, update header Checksum.
- Forward packet to outgoing interface.
- Transmit packet onto link.

Question: How is the address looked up in a real router?

Separate Table Entries Per Address

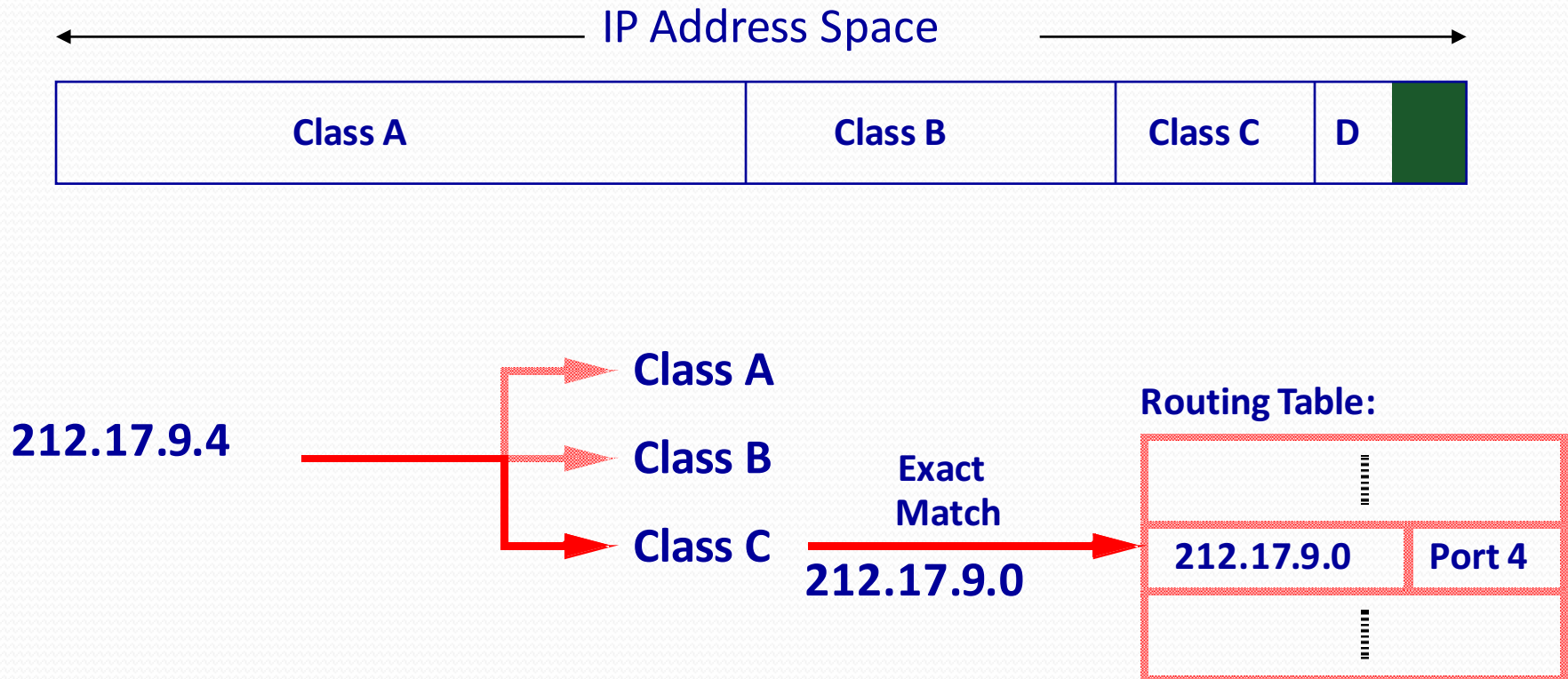
- If a router had a forwarding entry per IP address
 - Match destination address of incoming packet
 - ... to the forwarding-table entry
 - ... to determine the outgoing interface



Separate Entry Class-based Address

- If the router had an entry per class-based prefix
 - Mixture of Class A, B, and C addresses
 - Depends on the first couple of bits of the destination
- Identify the mask automatically from the address
 - First bit of 0: class A address (/8)
 - First two bits of 10: class B address (/16)
 - First three bits of 110: class C address (/24)
- Then, look in the forwarding table for the match
 - E.g., 1.2.3.4 maps to 1.2.3.0/24
 - Then, look up the entry for 1.2.3.0/24
 - ... to identify the outgoing interface

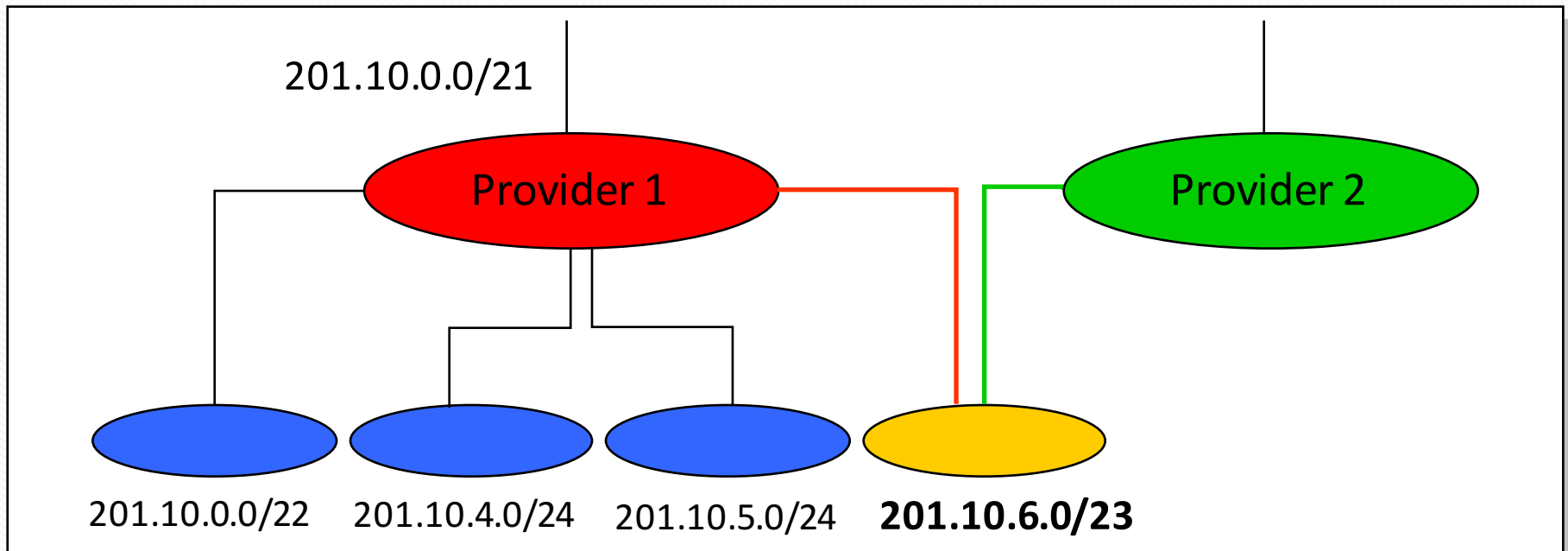
Example – *Class-based Addressing*



Exact Match: There are many well-known ways to find an exact match in a table.

CIDR Makes Packet Forwarding Harder

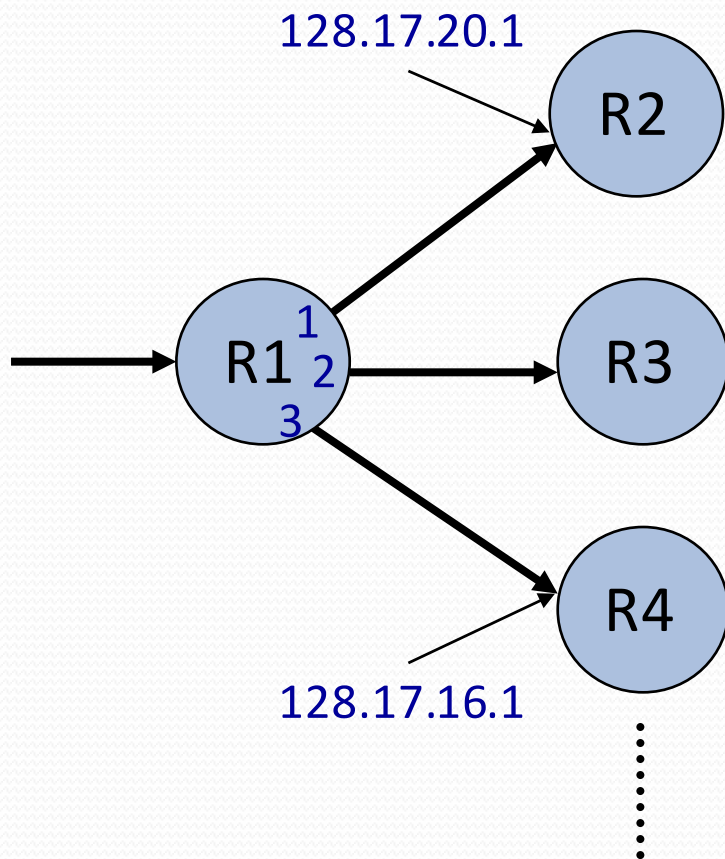
- There's no such thing as a free lunch
 - CIDR allows efficient use of the limited address space
 - But, CIDR makes packet forwarding much harder
- Forwarding table may have many matches
 - E.g., table entries for 201.10.0.0/21 and 201.10.6.0/23
 - The IP address 201.10.6.17 would match both!



Longest Prefix Match Forwarding

- Forwarding tables in IP routers
 - Maps each IP *prefix* to next-hop link(s)
- Destination-based forwarding
 - Packet has a destination address
 - Router identifies longest-matching prefix
 - Cute algorithmic problem: very fast lookups

How a Router Forwards Datagrams



e.g. 128.9.16.14 => Port 2

| Prefix | Next-hop | Port |
|--------------|-------------|------|
| 65/8 | 128.17.16.1 | 3 |
| 128.9/16 | 128.17.14.1 | 2 |
| 128.9.16/20 | 128.17.14.1 | 2 |
| 128.9.19/24 | 128.17.10.1 | 7 |
| 128.9.25/24 | 128.17.14.1 | 2 |
| 128.9.176/20 | 128.17.20.1 | 1 |
| 142.12/19 | 128.17.16.1 | 3 |

Forwarding Table

Simplest Algorithm is Too Slow

- Scan the forwarding table one entry at a time
 - See if the destination matches the entry
 - If so, check the size of the mask for the prefix
 - Keep track of the entry with longest-matching prefix
- Overhead is linear in size of the forwarding table
 - Today, that means 400,000-500,000 entries!
 - And, the router may have just a few nanoseconds
 - ... before the next packet is arriving
- Need greater efficiency to keep up with line rate
 - Better algorithms
 - Hardware implementations

Lookup Performance Required

| Line | Line Rate | Pktsize=40B | Pktsize=240B |
|-------|-----------|-------------|--------------|
| T1 | 1.5Mbps | 4.68 Kpps | 0.78 Kpps |
| OC3 | 155Mbps | 480 Kpps | 80 Kpps |
| OC12 | 622Mbps | 1.94 Mpps | 323 Kpps |
| OC48 | 2.5Gbps | 7.81 Mpps | 1.3 Mpps |
| OC192 | 10 Gbps | 31.25 Mpps | 5.21 Mpps |

Fast Lookups

- There are algorithms that are faster than linear scan
 - Proportional to number of bits in the address
- We can use special hardware
 - Content Addressable Memories (CAMs)
 - Allows look-ups on a key rather than flat address
- Huge innovations in the mid-to-late 1990s
 - After CIDR was introduced (in 1994)
 - ... and longest-prefix match was a major bottleneck

Where do Forwarding Tables Come From?

- Routers have forwarding tables
 - Map prefix to outgoing link(s)
- Entries can be statically configured
 - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn’t adapt
 - To failures
 - To new equipment
 - To the need to balance load
 - ...
- That is where other technologies come in...
 - Routing protocols, DHCP, and ARP

Packet Routing and Forwarding

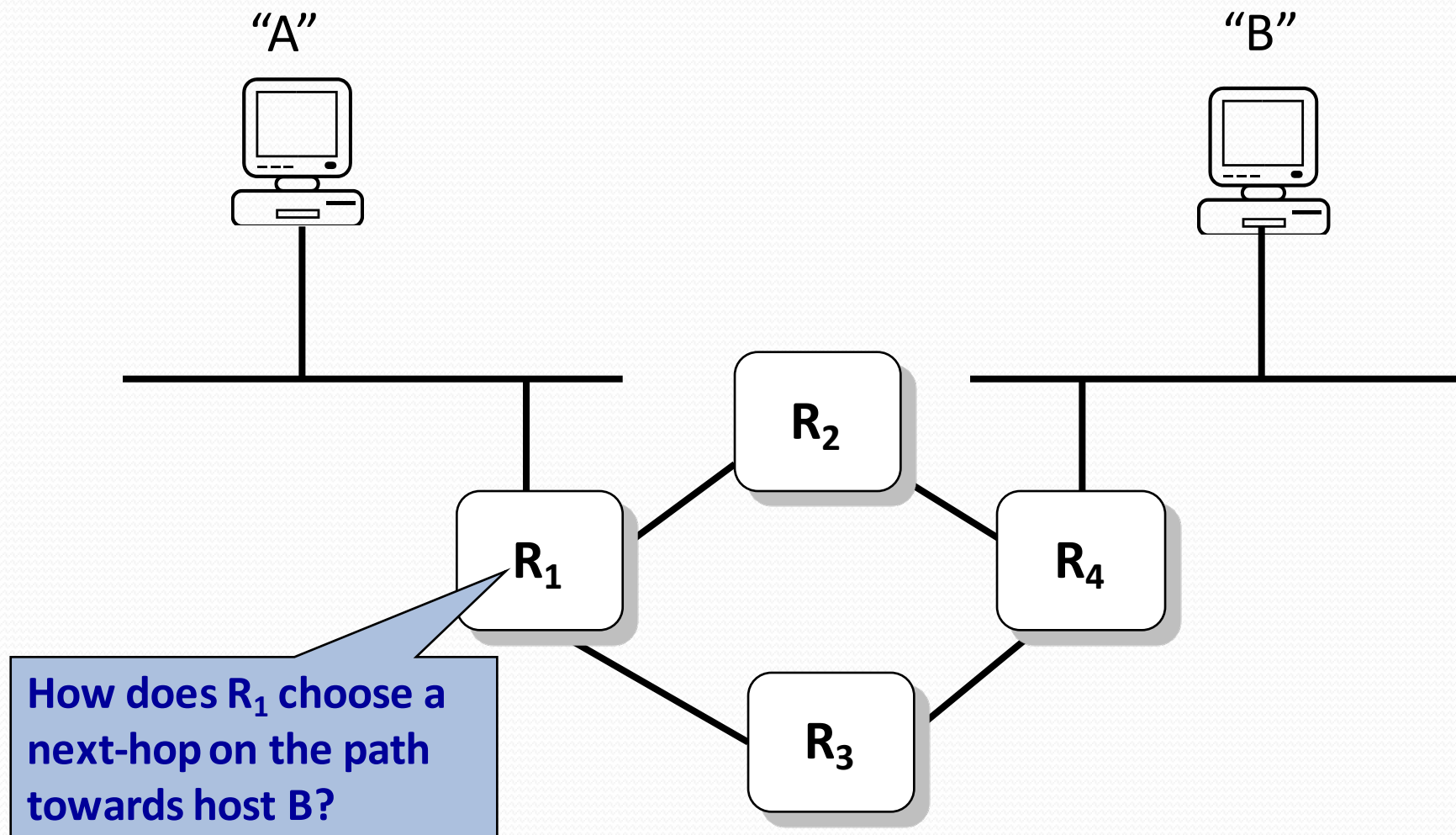
- Forwarding IP datagrams
 - Class-based vs. CIDR

Routing Techniques

- Naïve: Flooding
- Distance vector: Distributed Bellman Ford Algorithm
- Link state: Dijkstra's Shortest Path First-based Algorithm

**Routing is a very complex subject, and has many aspects.
Here, we will concentrate on the basics.**

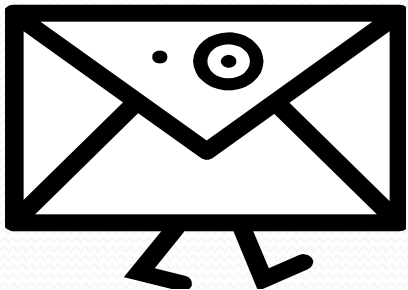
The Problem



with routing algorithm, take into consideration of traffic of possible routes

What is Routing?

- A famous quotation from RFC 791
 - “A name indicates what we seek.
An address indicates where it is.
A route indicates how we get there.”
-- Jon Postel



Forwarding vs. Routing

- Forwarding: *data plane*
 - Directing a data packet to an outgoing link
 - Individual router using a forwarding table
- Routing: *control plane*
 - Computing paths the packets will follow
 - Routers talking amongst themselves
 - Individual router creating a forwarding table

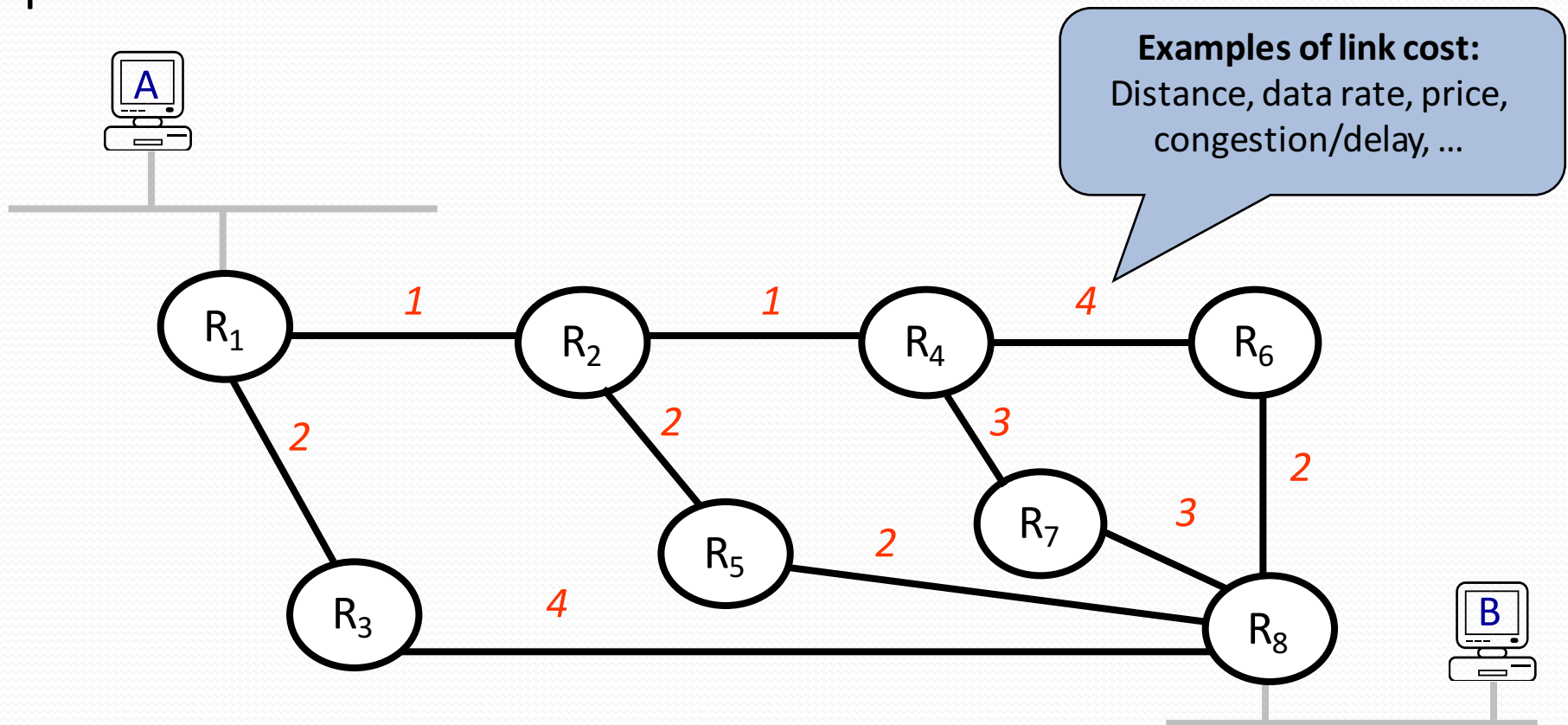


Why Does Routing Matter?

- End-to-end performance
 - Quality of the path affects user performance
 - Propagation delay, throughput, and packet loss
- Use of network resources
 - Balance of the traffic over the routers and links
 - Avoiding congestion by directing traffic to lightly-loaded links
- Transient disruptions during changes
 - Failures, maintenance, and load balancing
 - Limiting packet loss and delay during changes

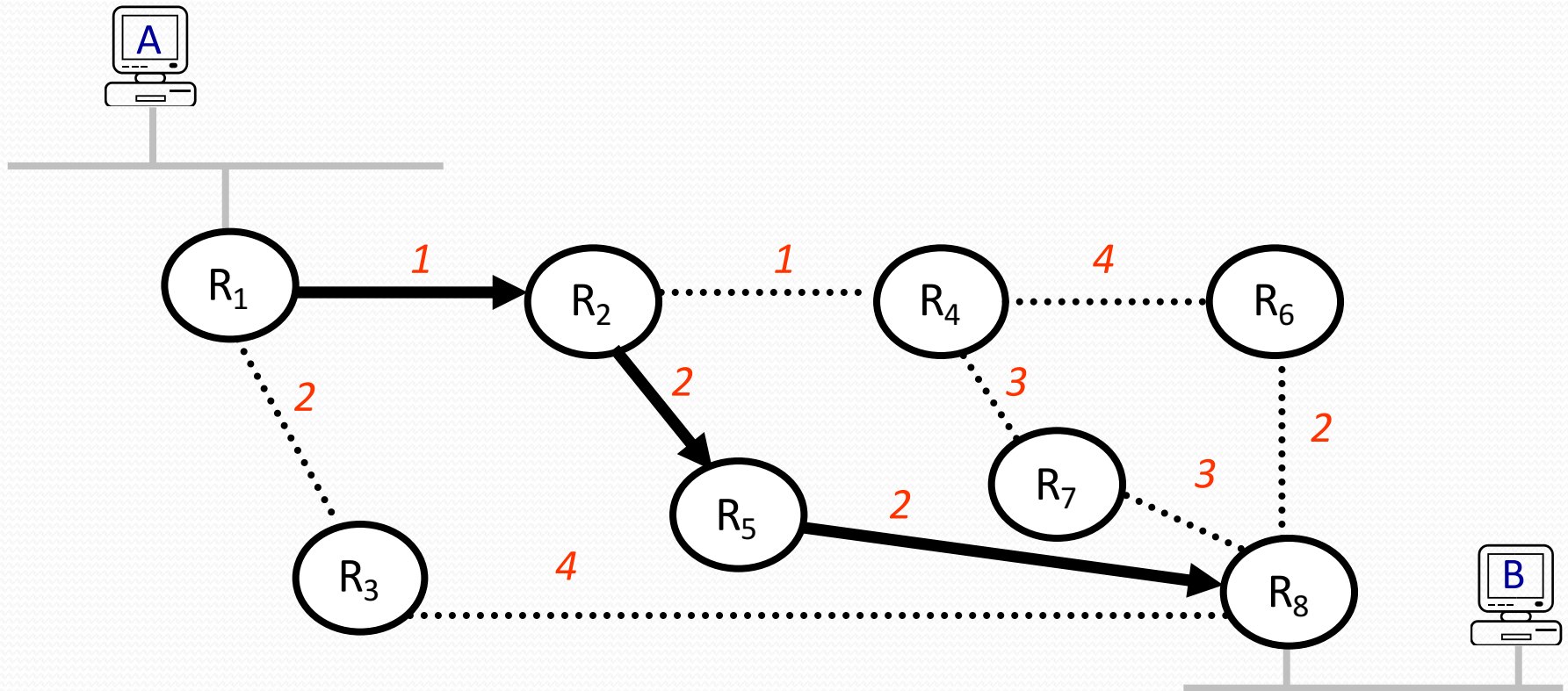
Example Network

Objective: Determine the route from A to B that minimizes the path cost.



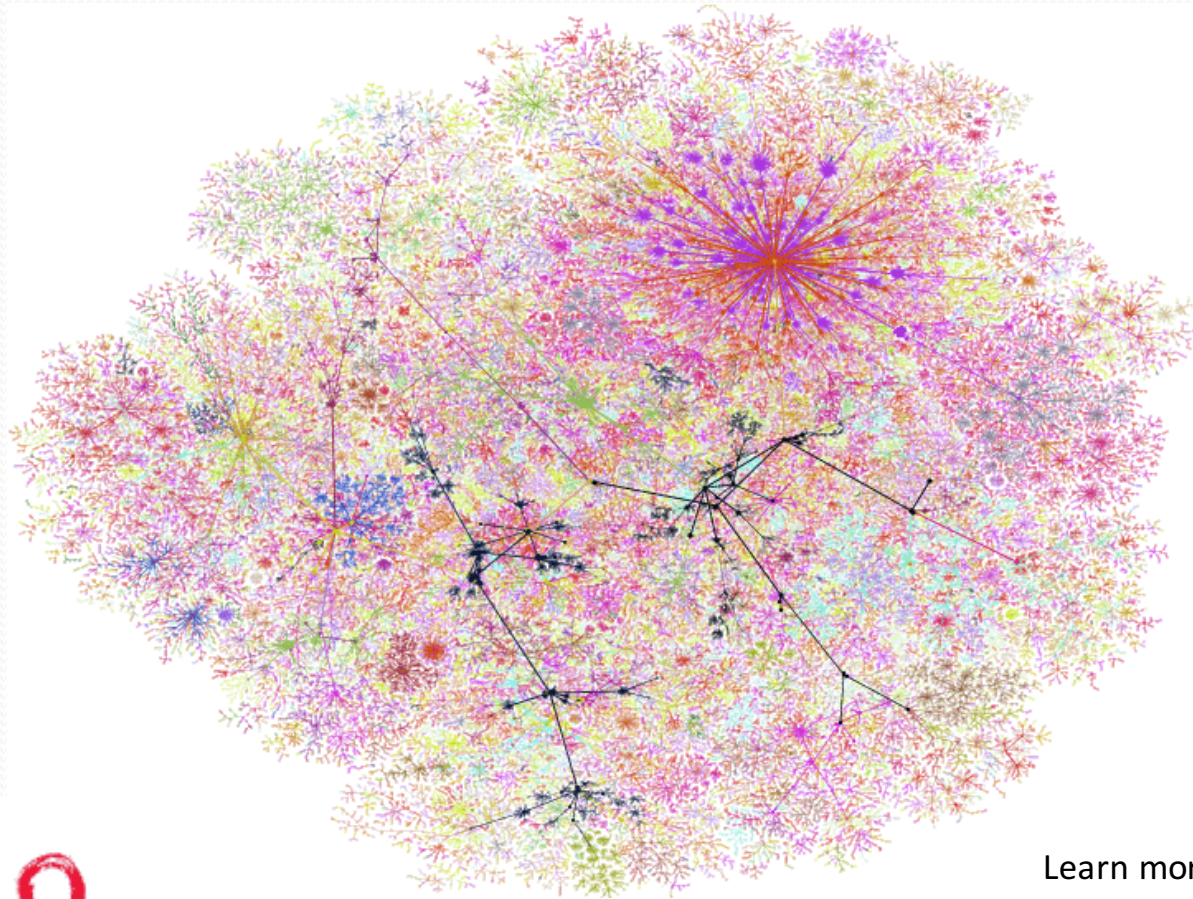
Example Network

In this simple case, solution is clear from inspection



What about this Network...!?

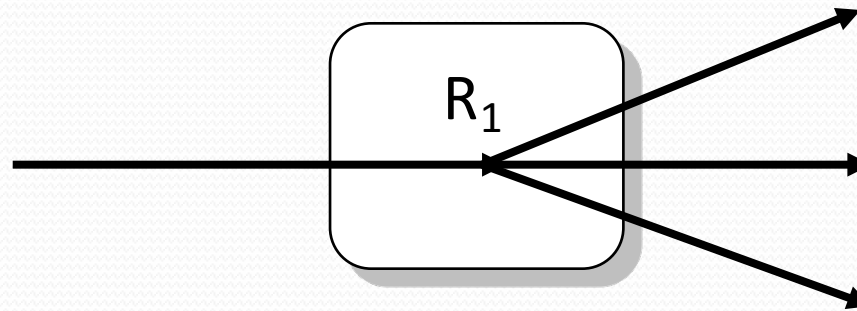
The public Internet in 1999



Learn more at
<http://www.lumeta.com>

Technique 1: Naïve Approach

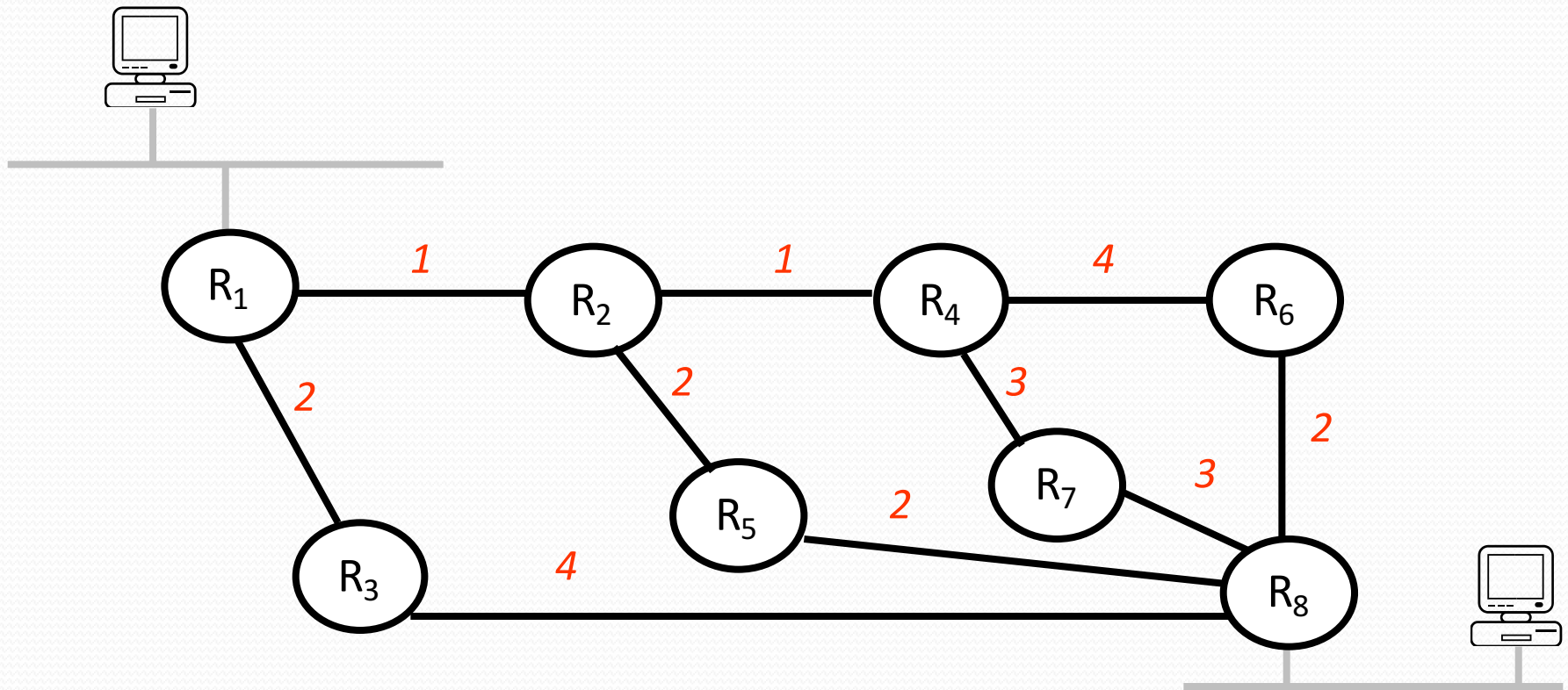
Flood! -- Routers forward packets to all ports except the ingress port.



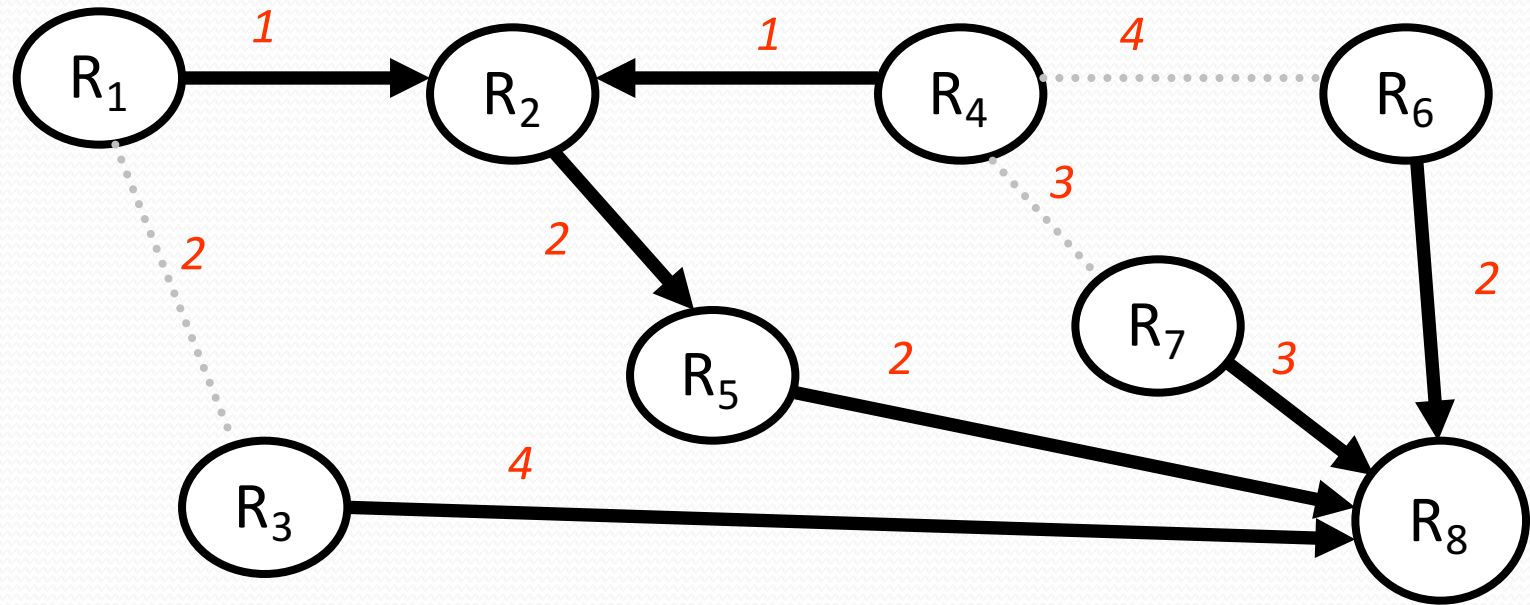
- Advantages:
 - Simple
 - Every destination in the network is reachable.
- Disadvantages:
 - Some routers receive a packet multiple times.
 - Packets can go round in loops forever. circular reference
 - Inefficient.

Lowest Cost Routes

Objective: Find the lowest cost route from each of (R_1, \dots, R_7) to R_8 .



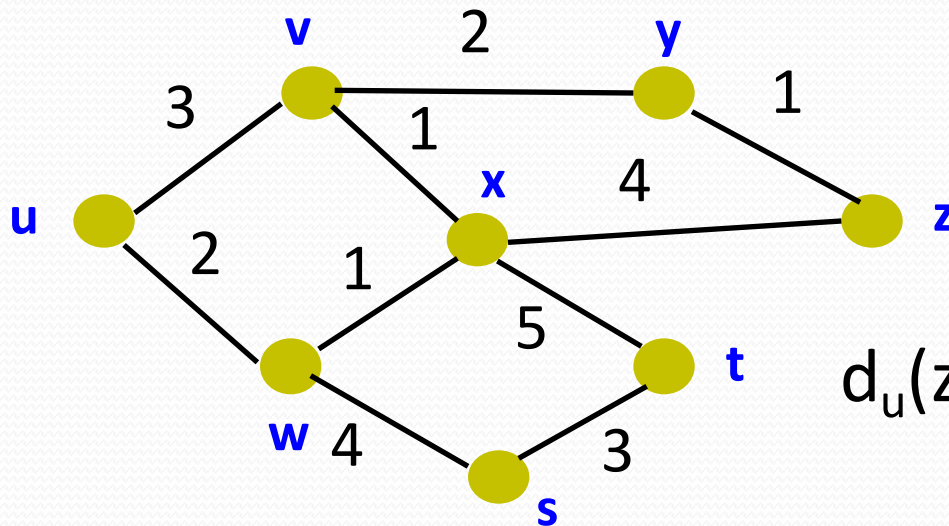
A Spanning Tree



- The solution is a **spanning tree** with R_8 as the root of the tree.
- **Tree:** There are no loops.
- **Spanning:** All nodes included.
- We'll see two algorithms that build spanning trees automatically:
 - The distributed Bellman-Ford algorithm
 - Dijkstra's shortest path first algorithm

Technique 2: Distance Vector Distributed Bellman-Ford Algorithm

- Define distances at each node x
 - $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
 - $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v



$$d_u(z) = \min\{c(u,v) + d_v(z), \\ c(u,w) + d_w(z)\}$$

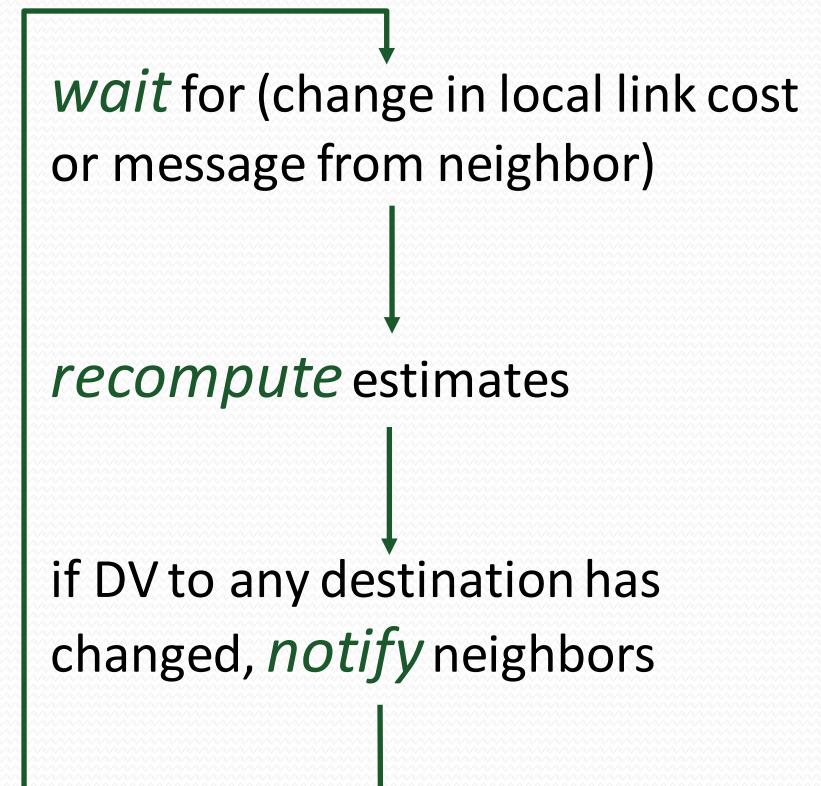
Distance Vector Algorithm

- $c(x,v)$ = cost for direct link from x to v
 - Node x maintains costs of direct links $c(x,v)$
- $D_x(y)$ = estimate of least cost from x to y
 - Node x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x maintains its neighbors' distance vectors
 - For each neighbor v , x maintains $\mathbf{D}_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
 - And neighbors update their own distance vectors
 - $D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Distance Vector Algorithm

- Iterative, asynchronous:
each local iteration caused by:
 - Local link cost change
 - Distance vector update message from neighbor
- Distributed:
 - Each node notifies neighbors only when its DV changes
 - Neighbors then notify their neighbors if necessary

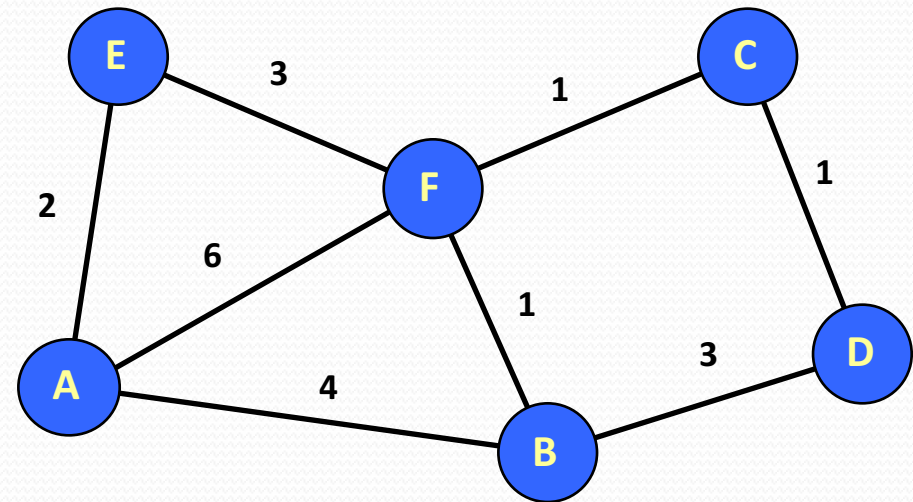
Each node:



Distance Vector Example: Step 1

Optimum 1-hop paths

| Table for A | | | Table for B | | |
|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | | — | C | | — |
| D | | — | D | 3 | D |
| E | 2 | E | E | | — |
| F | 6 | F | F | 1 | F |

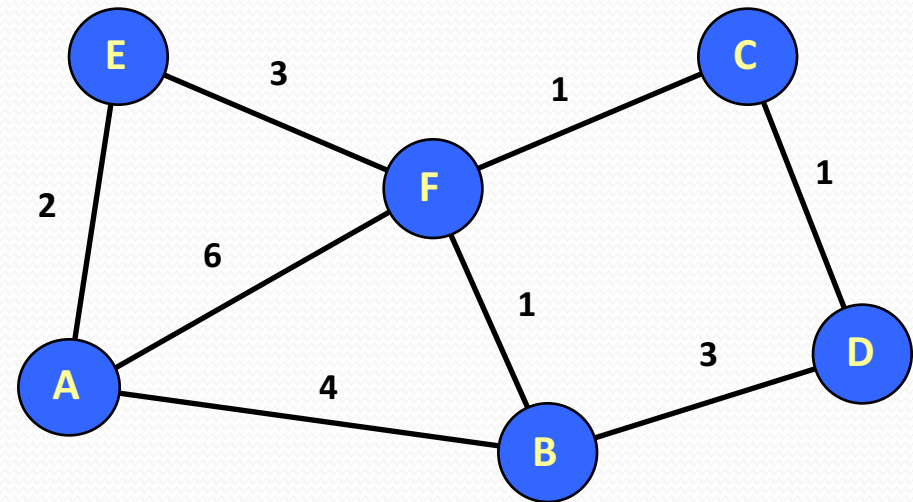


| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | | — | A | | — | A | 2 | A | A | 6 | A |
| B | | — | B | 3 | B | B | | — | B | 1 | B |
| C | 0 | C | C | 1 | C | C | | — | C | 1 | C |
| D | 1 | D | D | 0 | D | D | | — | D | | — |
| E | | — | E | | — | E | 0 | E | E | 3 | E |
| F | 1 | F | F | | — | F | 3 | F | F | 0 | F |

Distance Vector Example: Step 2

Optimum 2-hop paths

| Table for A | | | Table for B | | |
|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | 7 | F | C | 2 | F |
| D | 7 | B | D | 3 | D |
| E | 2 | E | E | 4 | F |
| F | 5 | E | F | 1 | F |

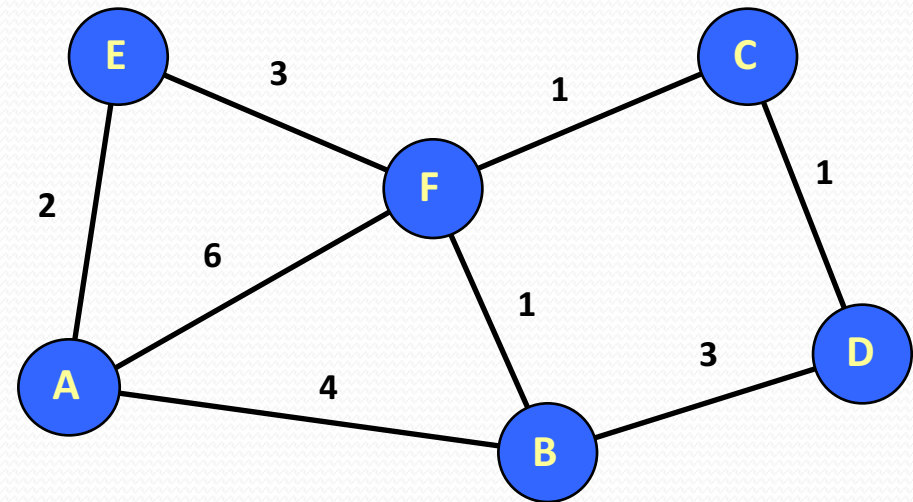


| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | 7 | F | A | 7 | B | A | 2 | A | A | 5 | B |
| B | 2 | F | B | 3 | B | B | 4 | F | B | 1 | B |
| C | 0 | C | C | 1 | C | C | 4 | F | C | 1 | C |
| D | 1 | D | D | 0 | D | D | ∞ | — | D | 2 | C |
| E | 4 | F | E | ∞ | — | E | 0 | E | E | 3 | E |
| F | 1 | F | F | 2 | C | F | 3 | F | F | 0 | F |

Distance Vector Example: Step 3

Optimum 3-hop paths

| Table for A | | | Table for B | | |
|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop |
| A | 0 | A | A | 4 | A |
| B | 4 | B | B | 0 | B |
| C | 6 | E | C | 2 | F |
| D | 7 | B | D | 3 | D |
| E | 2 | E | E | 4 | F |
| F | 5 | E | F | 1 | F |



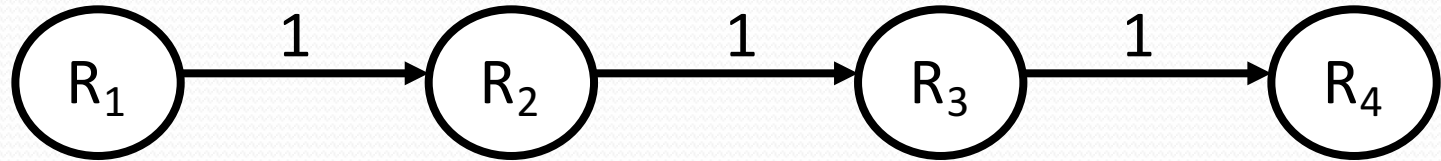
| Table for C | | | Table for D | | | Table for E | | | Table for F | | |
|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|
| Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop | Dst | Cst | Hop |
| A | 6 | F | A | 7 | B | A | 2 | A | A | 5 | B |
| B | 2 | F | B | 3 | B | B | 4 | F | B | 1 | B |
| C | 0 | C | C | 1 | C | C | 4 | F | C | 1 | C |
| D | 1 | D | D | 0 | D | D | 5 | F | D | 2 | C |
| E | 4 | F | E | 5 | C | E | 0 | E | E | 3 | E |
| F | 1 | F | F | 2 | C | F | 3 | F | F | 0 | F |

Bellman-Ford Algorithm

- Questions:
 - How long can the algorithm take to run?
 - How do we know that the algorithm always converges?
 - What happens when link costs change, or when routers/links fail?
- Topology changes make life hard for the Bellman-Ford algorithm...

A Problem with Bellman-Ford

Bad news travels slowly



Consider the calculation of distances to R_4 :

| Time | R_1 | R_2 | R_3 |
|------|------------------------|----------|----------|
| 0 | 3, R_2 | 2, R_3 | 1, R_4 |
| 1 | 3, R_2 | 2, R_3 | 3, R_2 |
| 2 | 3, R_2 | 4, R_3 | 3, R_2 |
| 3 | 5, R_2 | 4, R_3 | 5, R_2 |
| ... | "Counting to infinity" | | ... |

$R_3 \longrightarrow R_4$ fails
←

Counting to Infinity Problem – Solutions

- Set infinity = “some small integer” (e.g. 16). Stop when count = 16.
- Split Horizon: Because R_2 received lowest cost path from R_3 , it does not advertise cost to R_3
- Split-horizon with poison reverse: R_2 advertises infinity to R_3
- There are many problems with (and fixes for) the Bellman-Ford algorithm.

Technique 3: Link State

Dijkstra's Shortest Path First Algorithm

- Routers send out update messages whenever the state of an incident link changes.
 - Called “Link State Updates”
- Based on all link state updates received each router calculates lowest cost path to all others, starting from itself.
 - Use Dijkstra's single-source shortest path algorithm
 - Assume all updates are consistent
- At each step of the algorithm, router adds the next shortest (i.e. lowest-cost) path to the tree.
- Finds spanning tree rooted at the router.

Dijkstra's Algorithm

1 **Initialization:**

2 $S = \{u\}$

3 for all nodes v

4 if v adjacent to u {

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

6 else $D(v) = \infty$

7

8 **Loop**

9 find w not in S with the smallest $D(w)$

10 add w to S

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

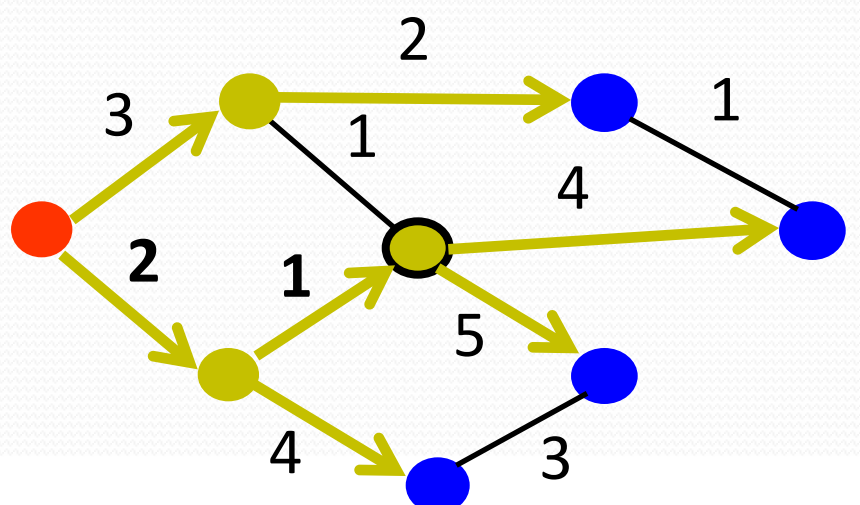
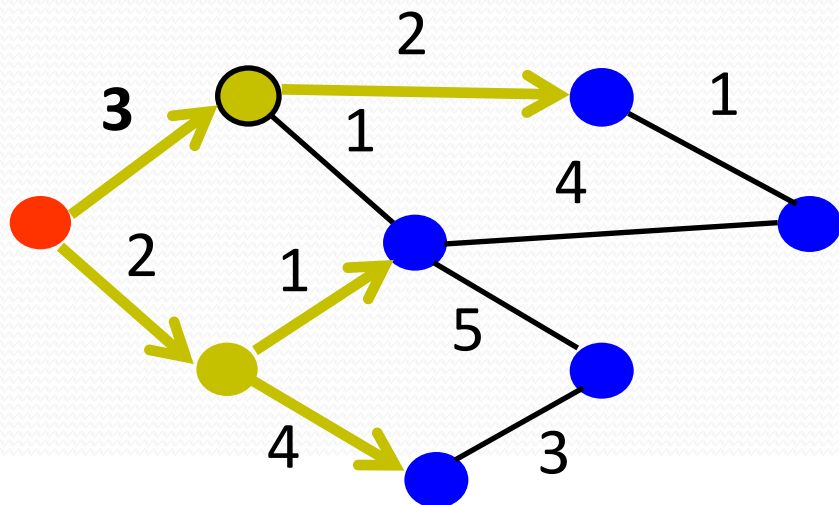
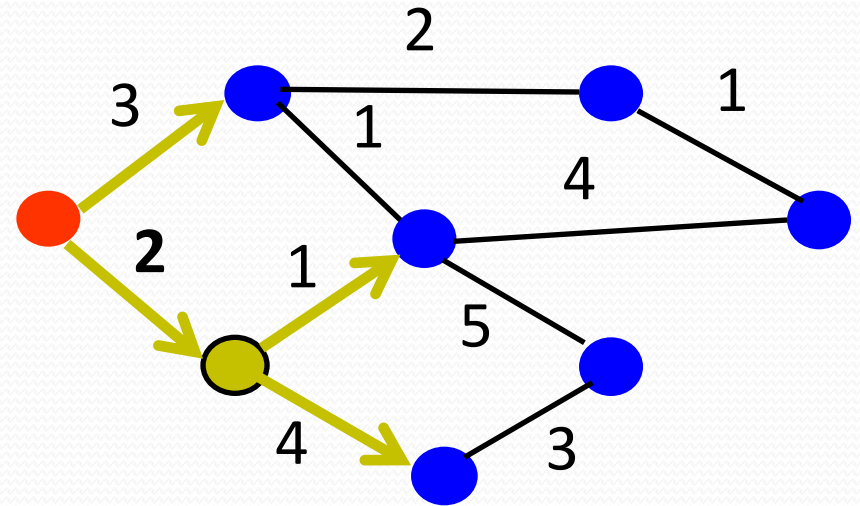
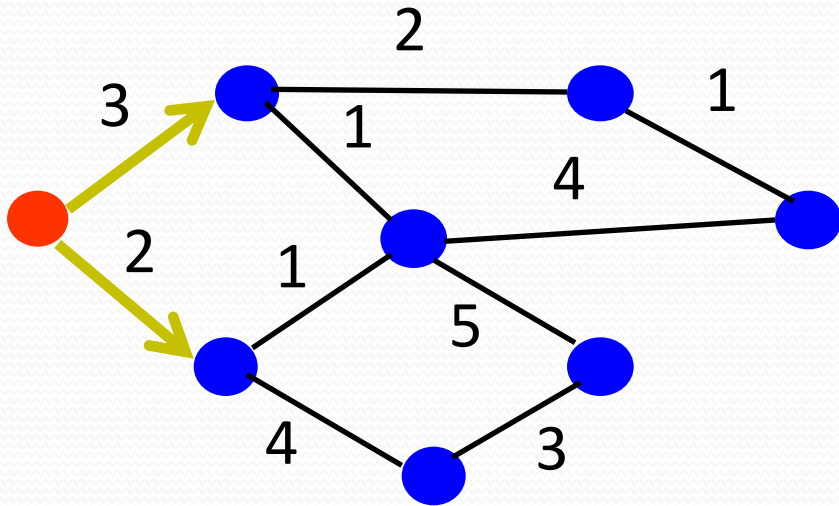
12 $D(v) = \min\{D(v), D(w) + c(w, v)\}$

13 **until all nodes in S**

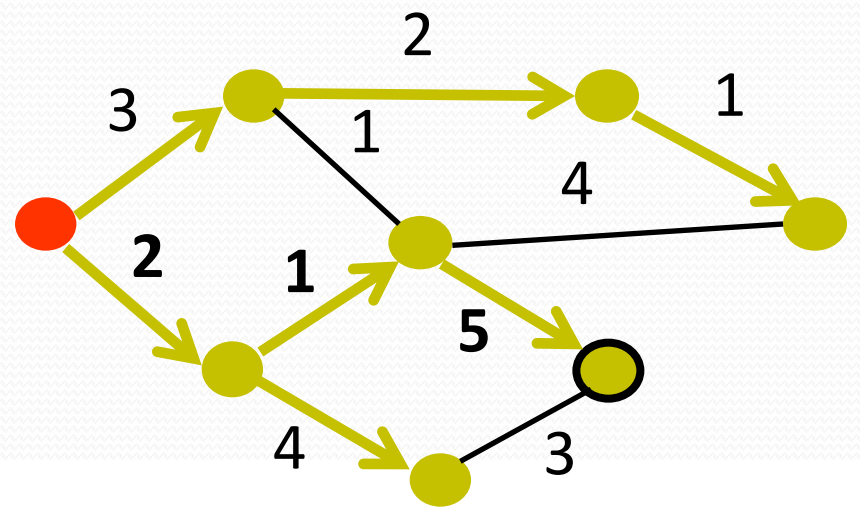
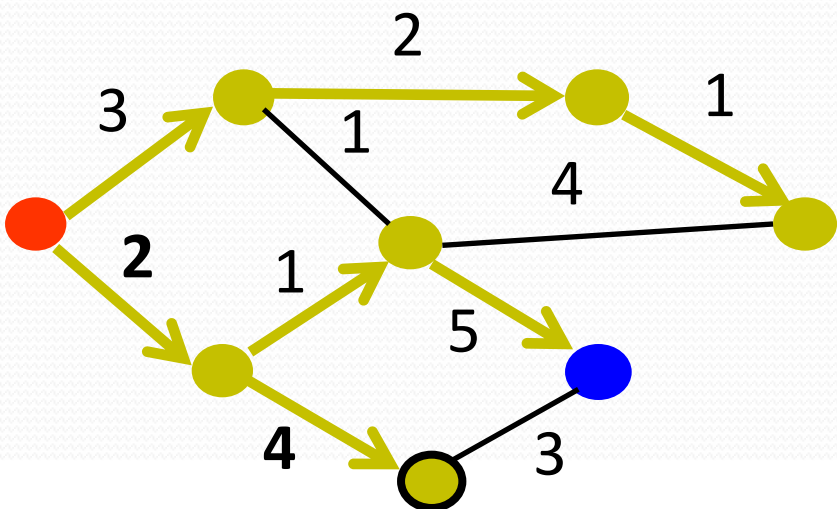
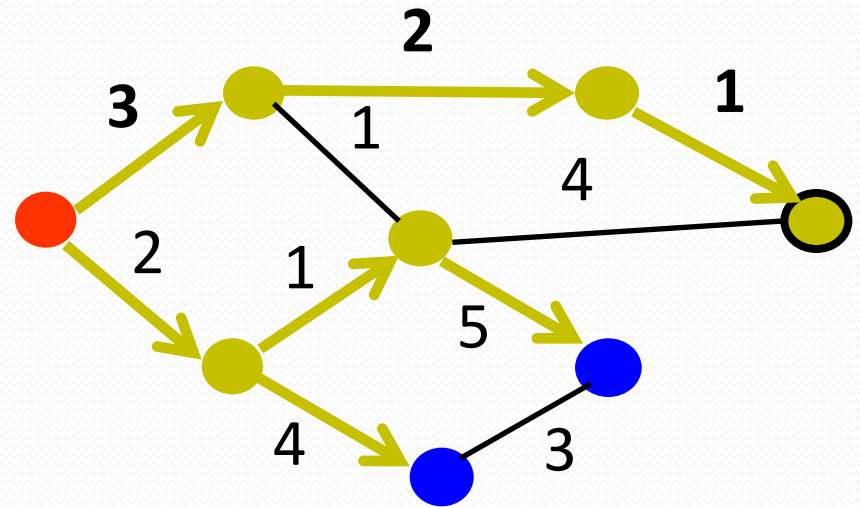
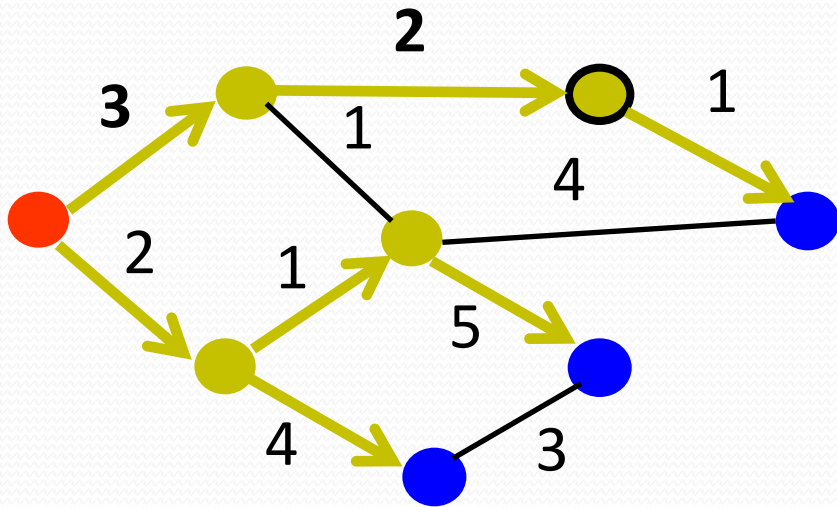


Dijkstra's Algorithm Example

Find Routes for the Red (Leftmost) Node

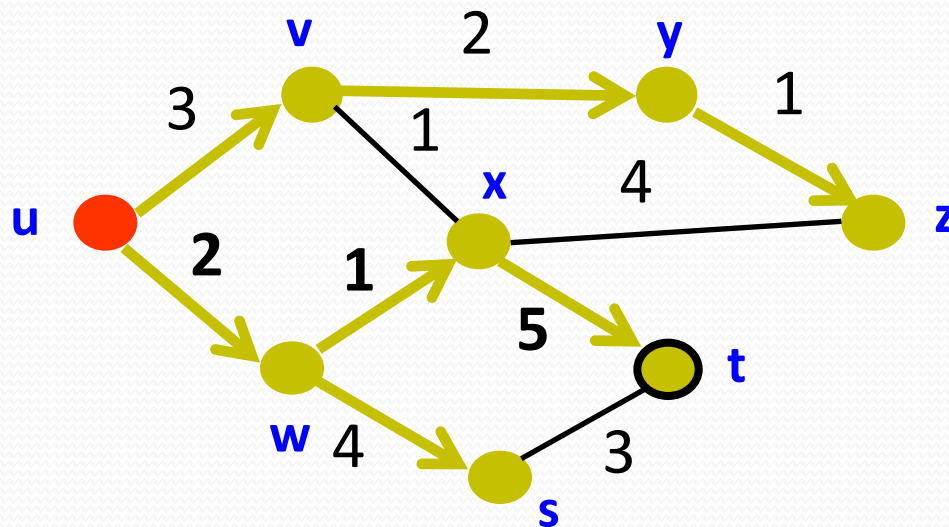


Dijkstra's Algorithm Example



Shortest-Path Tree

Shortest-path tree from u



Forwarding table at u

| | link |
|---|-------|
| v | (u,v) |
| w | (u,w) |
| x | (u,w) |
| y | (u,v) |
| z | (u,v) |
| s | (u,w) |
| t | (u,w) |

Reliable Flooding of LSP

- The Link State Packet:
 - The ID of the router that created the LSP
 - List of directly connected neighbors, and cost
 - Sequence number
 - TTL
- Reliable Flooding
 - Resend LSP over all links other than incident link, if the sequence number is newer. Otherwise drop it.
- Link State Detection:
 - Link layer failure
 - Loss of “hello” packets

Comparison of LS and DV algorithms

Message complexity

LS: with n nodes, E links, $O(nE)$ messages sent

DV: exchange between neighbors only

Convergence time varies

Speed of Convergence

LS: $O(n^2)$ algorithm requires $O(nE)$ messages

DV: convergence time varies

May be routing loops

Count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

Node can advertise incorrect *link* cost

Each node computes only its *own* table

DV:

DV node can advertise incorrect *path* cost

Each node's table used by others (error propagates)

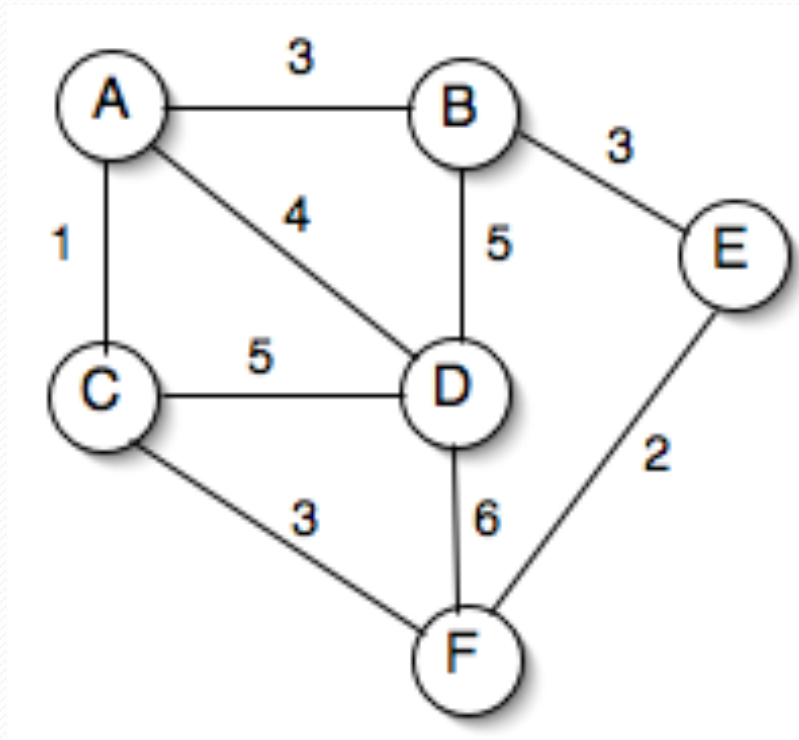
Questions on PA1 and PS1

- Any questions?

Review Questions – Last Week's Lecture

- What protocol does Ethernet use?
- What are the limitations of hubs and repeaters?
- What is the difference between a hub and a switch?
Between a switch and a router?
- What is a bridge?

Find the minimum spanning tree



More Questions

- What problem was CIDR trying to solve?
- What is ICMP used for?
- What is ARP?
- Which destination address does the link layer use?
 - a) IP address
 - b) MAC address
 - c) Subnet Mask
 - d) Broadcast Address

