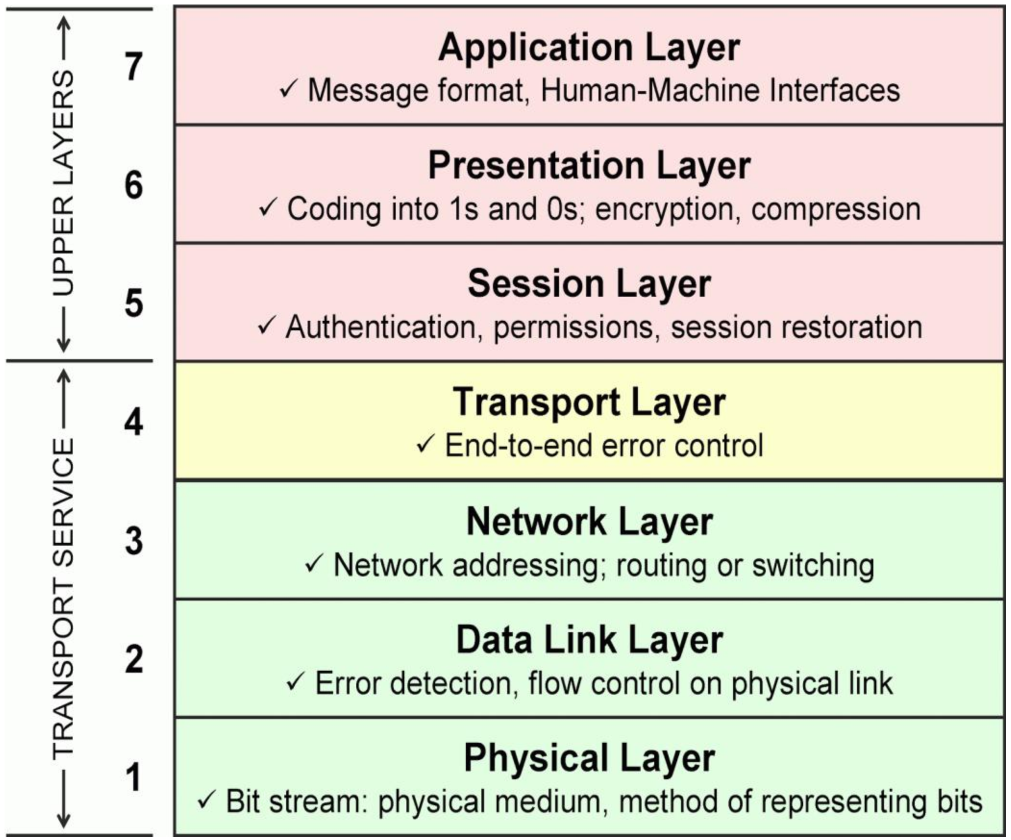
Network Layer Recap
(and a little data link layer)

Christian Svorre Jordan

UNIVERSITY OF COPENHAGEN

OSI Model



What we code (or from libraries)
A3 and A4

TCP & UDP

This presentation

The physical medium.
Super abstract and not relevant

Image source:
https://ciscoquestionsandanswers.blogspot.com/2014/12/osi-model-explained-summarydefinitions.html

OSI Model Continued

Network Layer:
Packets

Data Link Layer:
Data Frames

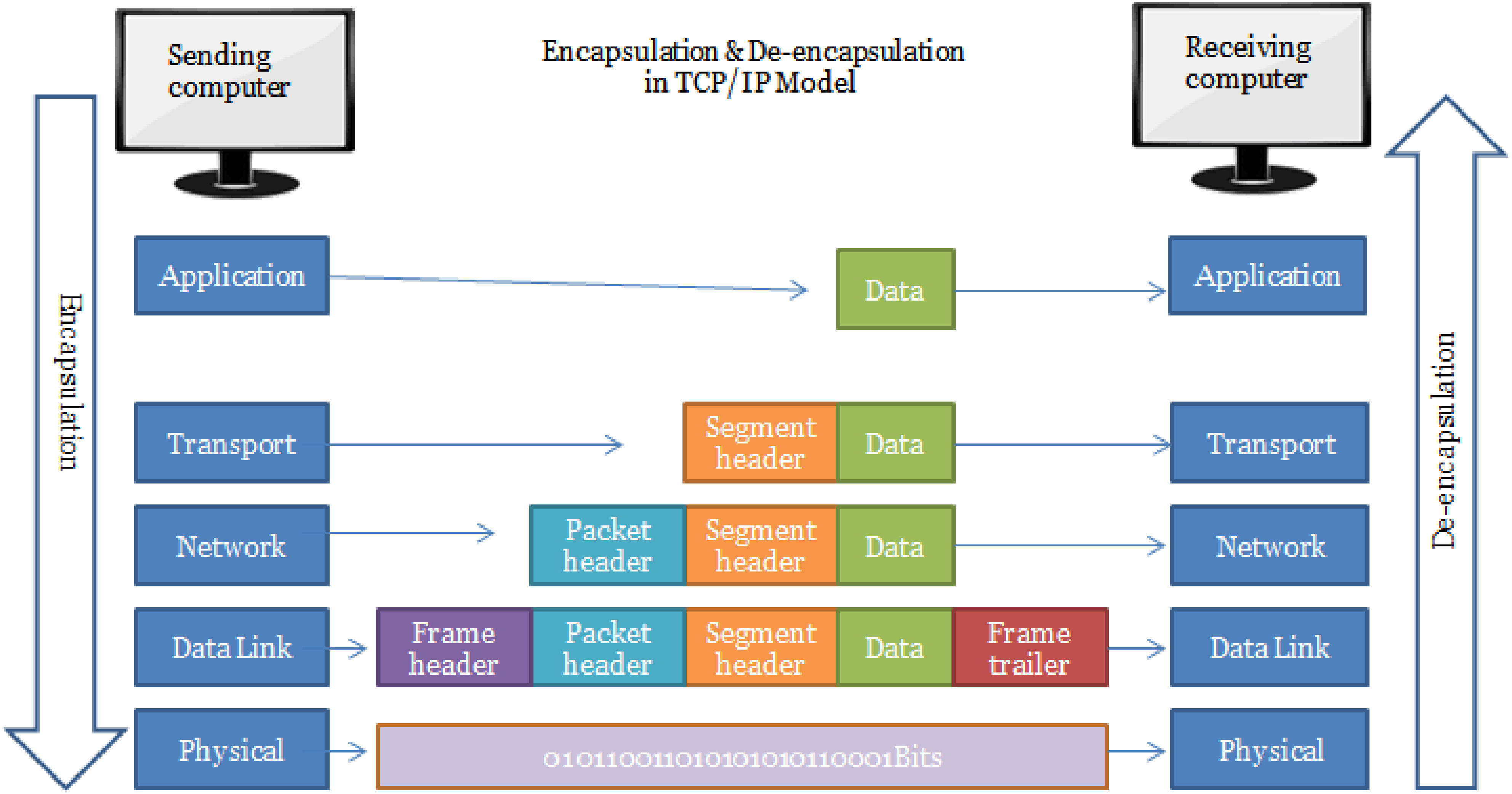


Image source:
https://stackoverflow.com/questions/31446777/difference-between-packets-and-frames

Data Link Layer Overview

- Goal: Routing within a network.
- Uses data frames not packets
- MAC Address (Media Access Control)
  - Unique identifier for all network devices
  - Hardcoded in NIC (Network Interface Card)
- ARP (Address Resolution Protocol)
  - Translate IP Address to MAC Address
- Ethernet
- Error checking for frames (CRC, Checksum, Parity check)

Hardware in The Data Link Layer

- Router or Gateway
  - Technically belongs to the Network Layer
  - Is a barrier between the internet (WAN) and local network (LAN)
- Hub
  - Connects multiple devices
  - Data frames are copied to everyone
- Switch
  - A smarter hub
  - Data frames are copied only to the recipient’s side
  - Uses a table to keep track of MAC addresses
  - Isolates traffic, decreasing congestion
- Network Interface Card
  - The interface between a device (computer) to the network
  - Has a MAC

Ethernet

- Communication protocol for data frames
- Uses CSMA/CD (Carrier sense, multiple access with collision detection)
  - Carrier Sense (CS)
    - Listen before speaking, and don't interrupt
  - Collision Detection (CD)
    - If collision, wait then retry
  - Exponential backoff
    - Wait a random but increasing amount of time after collision
    - Random, so two devices don't synchronize and talk at the same time

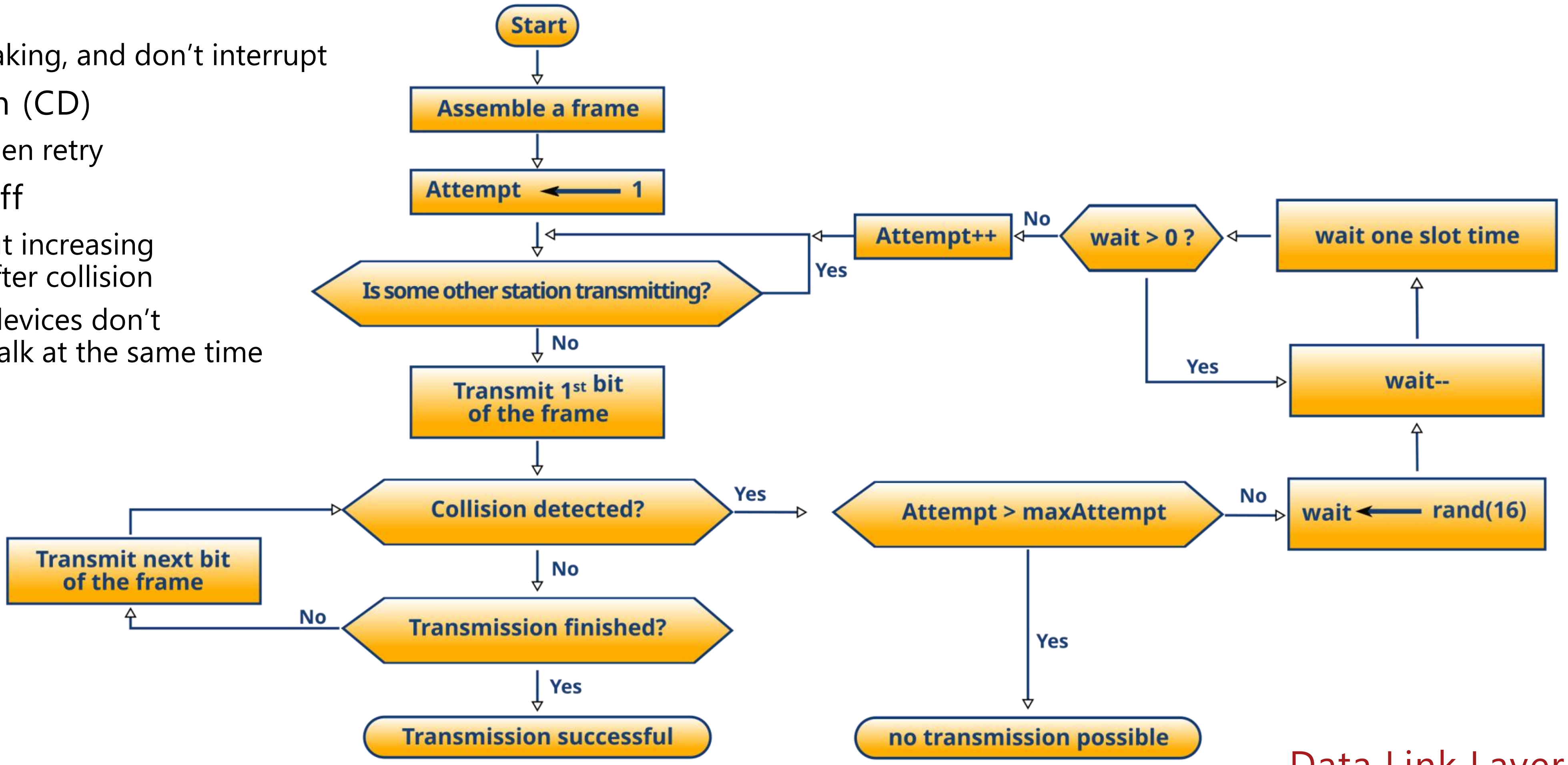


Image source:
https://en.wikipedia.org/wiki/Carrier-sense_multiple_access_with_collision_detection

Network Layer Overview

- Goal: Routing among networks
- IPv4 and IPv6
- IP is "good enough"
    - Other layers can worry about things like reliability
- NAT (Network Address Translation)
- DHCP (Dynamic Host Configuration Protocol)
- Middleboxes – Devices that modify packets
- Forwarding (Data Plane) – Moving a packet from A to B
- Routing (Control Plane) – Choosing the best (fastest) path to move packets along
    - Link State Algorithm
    - Distance Vector Algorithm

IPv4 Vs IPv6

IPv4	IPv6
4 bytes	16 bytes
1 byte per segment (octet)	2 byte per segment (hextet)
2^{32} = 4 billion addresses	2^{128} = 324 billion billion billion billion addresses
Decimal format	Hexadecimal format

Octet

173.80.228.101

Dot (.) separated

Hextet

2522:688e:dfe4:e699:7257:ed94:f345:9386

Colon (:) separated

Subnetting

- Problem: Expensive to figure out where IP addresses are located.
- Solution: Divide network into smaller subnets.
- Reduces traffic since less communication on a single network
- Security through isolation
- Split IP into 2 parts. Network (subnet) and host (device on subnet)

```
IP         = 173.80.228.101  = 10101101.01010000.11100100.01100101
Mask (24)  = 255.255.255.0   = 11111111.11111111.11111111.00000000
Subnet     = IP & mask       = 10101101.01010000.11100100.00000000
           = 173.80.228.0
```

Subnetting Sizes and Classes

- Classes (Legacy) (use bit prefix in address to choose)
    - A (0 prefix): 8 bits for network, 24 bits for hosts
        - Big organizations (Global ISP's, Google, Apple, governments).
    - B (10 prefix): 16 bits for network, 16 bits for hosts
        - Smaller organizations (Regional ISP's, universities, corporations)
    - C (110 prefix): 24 bits for network, 8 bits for hosts
        - Home networks or small businesses.

- Classless (Modern) (use extra information to choose subnet size)
    - Known as Classless Inter-Domain Routing (CIDR)
    - $\underbrace{173.80.228.101}_{\text{Ip address}}/\underbrace{24}_{\text{Network size}}$

Running out of IPv4

- More than 4 billion devices
- Why don’t we use IPv6?
  - Existing infrastructure needs to be replaced
  - NAT gives “more” addresses
  - Other countries do!

Network Address Translation (NAT)

- Problem: Running out of IPv4 addresses
- Solution: Allow multiple devices to use the same IP (or use IPv6)
- A hacky middlebox solution
- Router is the only one with a public IP
  - Translates private IPs from connected devices into a public IP.
- Gives security through isolation.
- Types:
  - Static NAT (one to one):
    - One public IP is mapped to one private IP
  - Dynamic NAT (many to some):
    - Uses a pool of public IPs
    - A private IP gets temporally mapped to a public IP from the pool
  - PAT aka. Port Address Translation (many to one)
    - Most common type
    - Uses port field in IP header to differentiate private IPs
    - Router may replace the port field of packets according to its internal table.

End-to-End Principle

- The internet is “dumb”.
  - Everything happens at the end terminals, not in between
- Net neutrality
- OSI model illustrates the end-to-end principle
- Middleboxes
  - Violates the end-to-end principle
  - Devices that transforms, inspects, filters, and manipulates traffic for purposes other than packet forwarding.
  - Examples:
    - NAT
    - Firewalls
    - Load balancers
    - Deep packet inspection (security)

Unicast vs Broadcast vs Anycast

- Broadcast -> To everyone
  - Special IP address or MAC address
- Unicast -> To someone specific
  - Specific IP address or MAC address
- Anycast -> To someone specific, but they have multiple devices
  - Specific IP address or MAC address
  - Multiple different devices with same IP or MAC
  - First come, first served principle
  - E.g. Google.com has one IP, but you connect to their nearest server
  - Happens in the Network Layer during routing

Dynamic Host Configuration Protocol (DHCP)

- Problem: New user joined the network, but they don't have a private IP
- Solution: Give them one!
- Device broadcast asking if anyone is willing to give them a private IP
  - They include their MAC address to identify
  - All DHCP servers on the network respond with IP offers
  - DHCP servers are typically running on the router
  - Device agrees to one offer
- IPs are leased for a fixed duration
  - If a device leaves the network, it will not inform the DHCP server (problem!)
  - Ensures that IPs are not permanently occupied
  - Forces devices to prove they are still present
  - Acts as a failsafe
- IPs can re renewed
  - Typically, after 50% of lease time is up. The device asks for renewal
  - Ensures smooth transition
- Special IPs:
  - Host portion is all 1 -> Broadcast IP
  - Host portion is all 0 -> (Typically) Router/Gateway Ip

Address Resolution Protocol (ARP)

- Problem: User has a private IP, but what is their MAC?
- Solution: Ask them!
- Device broadcasts a question
  - E.g. “Who has IP 192.168.1.42?”
  - Device with requested IP responds in unicast with their MAC
- ARP spoofing (poisoning)
  - Device responds to question with their MAC, despite IP not matching.
  - Allows interception and Man-in-the-middle attacks
- Time to live (TTL)
  - How long before asking for MAC again
  - In case IPs have changed
- Special MAC:
  - FF:FF:FF:FF:FF:FF -> MAC broadcast address

Forwarding vs Routing
Data Plane vs Control Plane

- Forwarding (Data Plane)
  - “Dumb”
  - Get data in and out as fast as possible
  - Speed is everything
  - Where to? Control plane figures it out
- Routing (Control Plane)
  - “Smart”
  - Figure out where to send data
  - Slower but that’s okay
  - Creates the routing table and network topology (idea of network structure)

Forwarding (Data Plane)

- Moving a packet from an incoming link to an outgoing link
- Incoming packet's destination IP is extracted from the header
- Checks forwarding table (aka. routing table) to find which direction to send

Forwarding (Data Plane)

- Moving a packet from an incoming link to an outgoing link
- Incoming packet's destination IP is extracted from the header
- Checks forwarding table (aka. routing table) to find which direction to send
- Problem: Destination IP matches multiple entries due to subnetting
  - E.g Destination IP 192.168.1.10 matches both 192.168.1.0/24 and 192.168.1.128/25
- Solution: Longest Prefix Match (LPM)
- Use the entry with the smallest host size (largest subnet mask size)
- In previous example 192.168.1.128/25 is more specific, so forward that way
- Remember, IPs and subnets are hierarchical!
- Both are valid, but one is better!
  - Sending to either IP address works
  - But one is a larger subnet and would thus create more congestion
  - So send to most specific that matches

Routing (Control Plane)

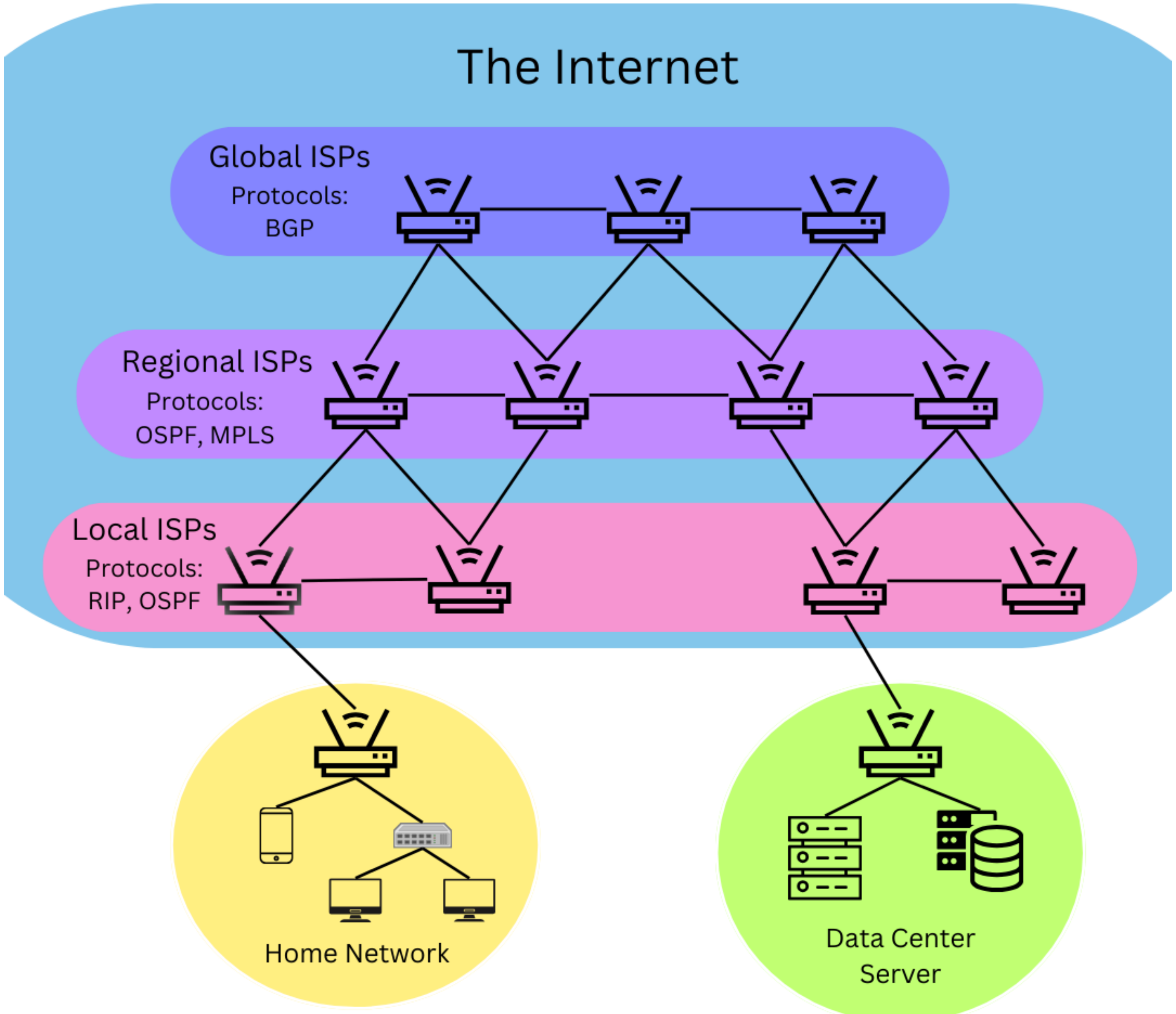
- Problem: How to get data from A to B as fast as possible
- Solution: The internet is a graph, so use shortest path algorithms like Djikstra's!

Routing (Control Plane)

- Problem: How to get data from A to B as fast as possible
- Solution: The internet is a graph, so use shortest path algorithms like Djikstra's!
- Problem 2: But Djikstra's requires we know what the whole graph (internet) looks like
- Solution 2: Figure out what the graph looks like somehow?
- Link State Algorithm and Distance Vector Algorithm to the rescue!

The Internet

- Is a hierarchy
- Protocols:
  - Open Shortest Path First (OSPF)
    - Link State Algorithm
  - Routing Information Protocol (RIP 💀)
    - Distance Vector Algorithm
  - Multiprotocol Label Switching (MLS)
  - Border Gateway Protocol (BGP)
  - Among others ⋯
    - Typically combines
      Link State and Distance Vector



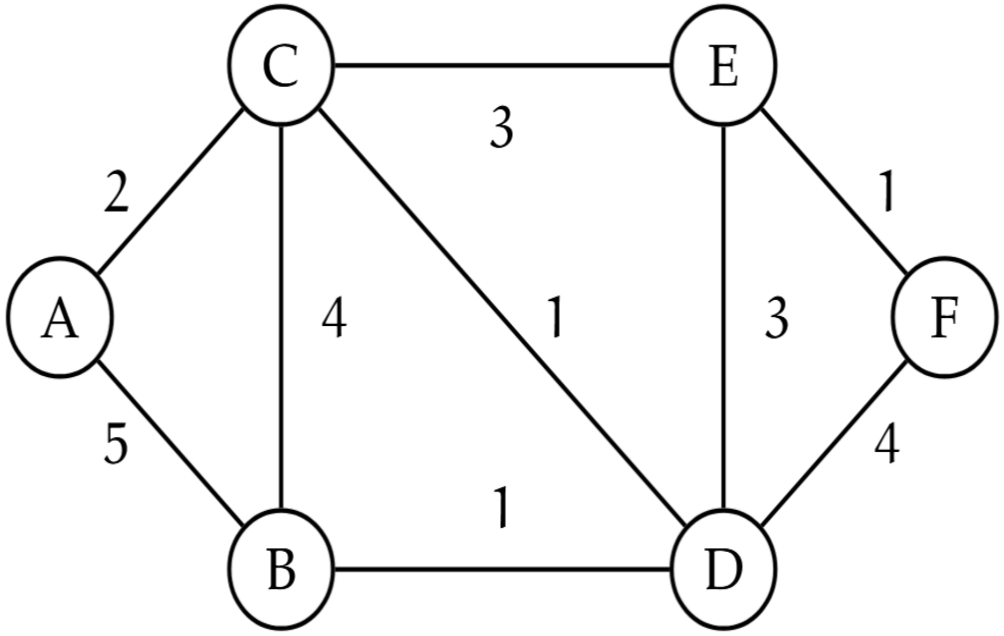
Link State Algorithm

- Steps
  1. Every router floods what its immediate connections looks like
     - Gives information such as latency and bandwidth
     - Flooding: Sends info to neighbors, who then sends it to their neighbors ⋯ and so on
  2. Every router uses the flooded information to build an idea of the network
  3. Use Dijkstra’s!

Link State Algorithm (Dijkstra's Algorithm)

1. Initialize a set of all visited nodes
2. Initialize a table with previous node and distance to all nodes and
    - Set the distance to the current/starting node as 0
    - Set distance to all other nodes as infinity
    - Set all previous nodes as NULL
3. Pick the node that is closest to current node and is not in the visited set
    - Set it as the new current node
    - Add it to the visited set
    - For the first iteration, this is the starting node
4. For each unvisited neighbor N, of the current node C
    - Update their distance in the table if their new distance is smaller than previous
        - $D(N) = \min(D(C) + E(C, N), D(N))$
          Where $D(X)$ is distance to node X from start and $E(C, N)$ is edge length from current to neighbor
        - If new distance was smaller, then set neighbor's previous node as the current node
5. Goto step 3, if there's still unvisited nodes

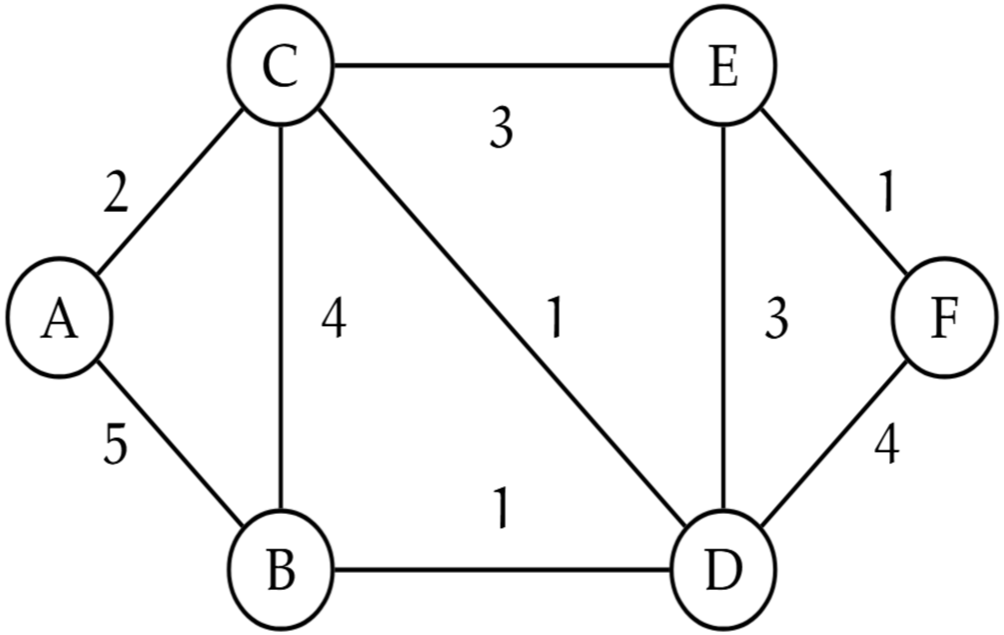
Link State Algorithm Example (Dijkstra’s Algorithm)



Step	N’	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)

D(X) = Distance to X from A
p(X) = Previous node to X
N’ = Visited nodes

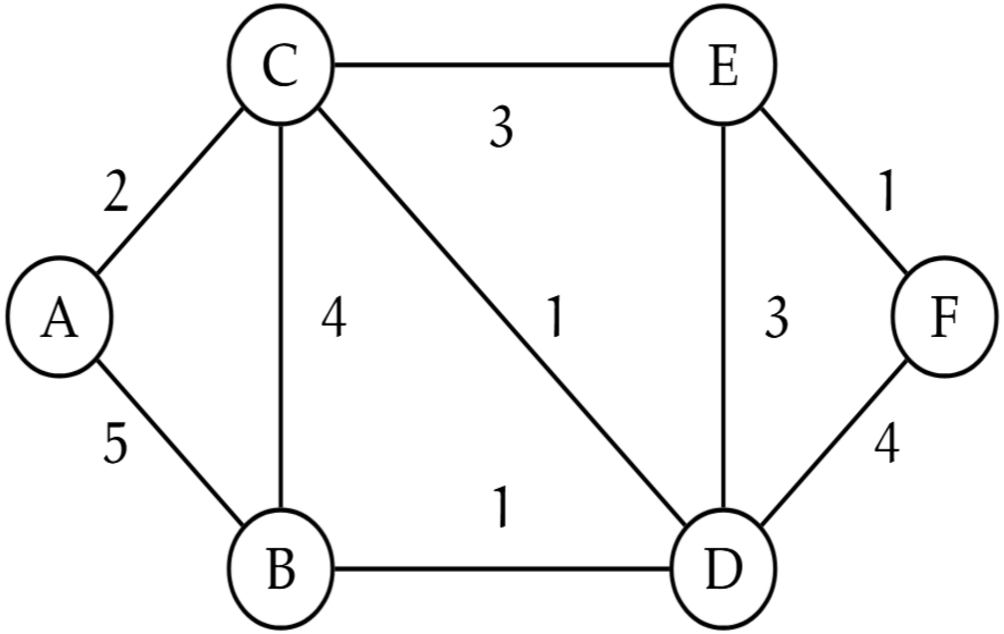
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

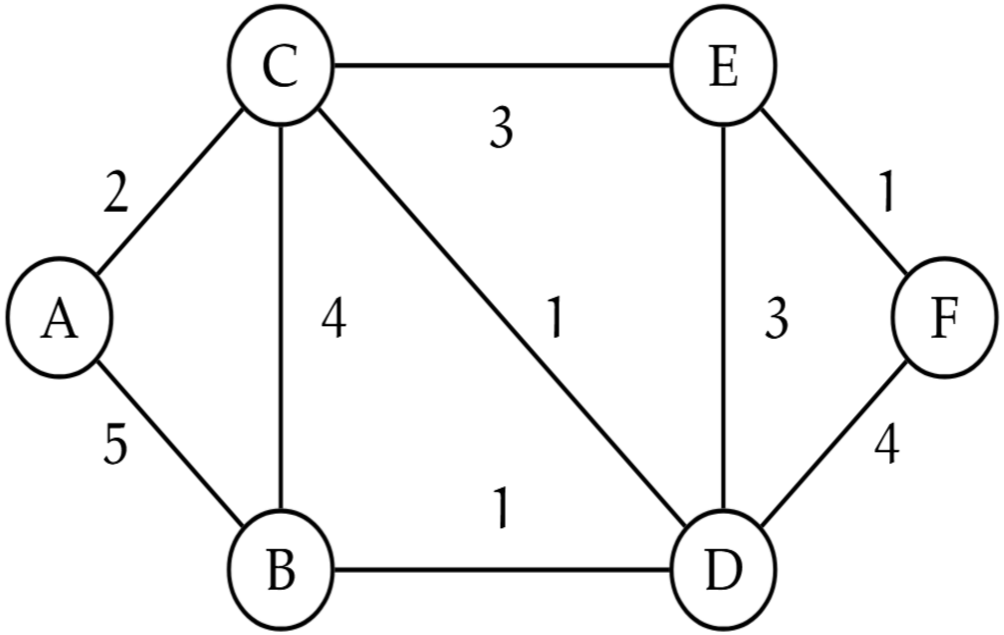
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

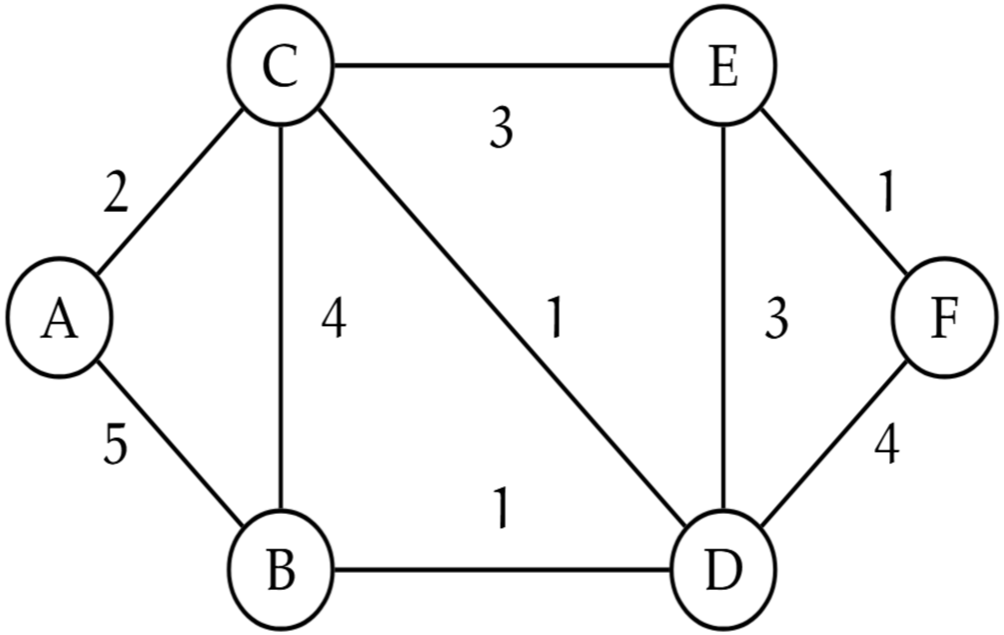
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

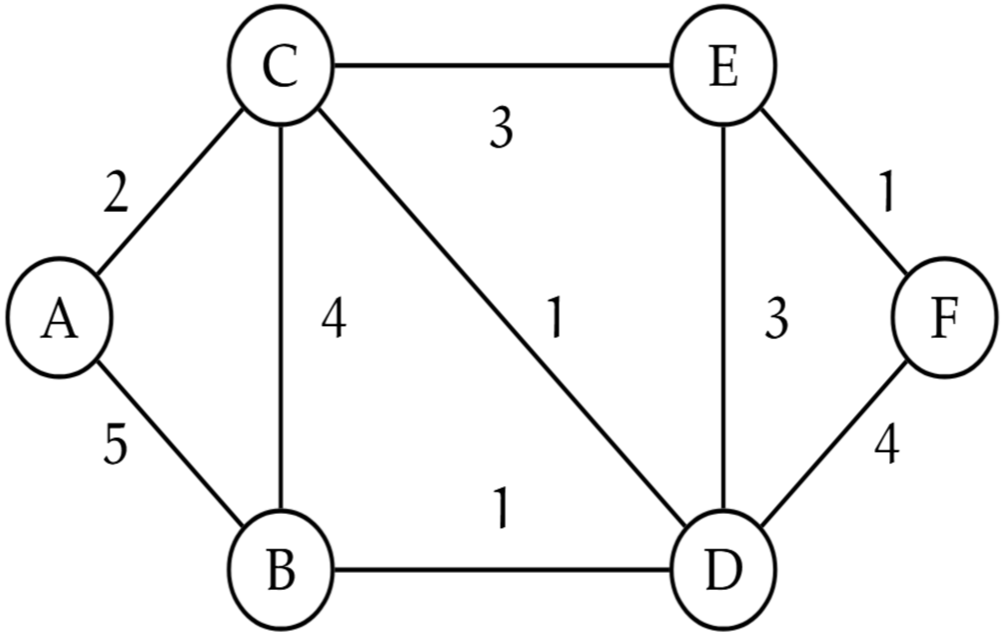
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D
3	A,C,D,B	4, D	2, A	3, C	5, C	7, D

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

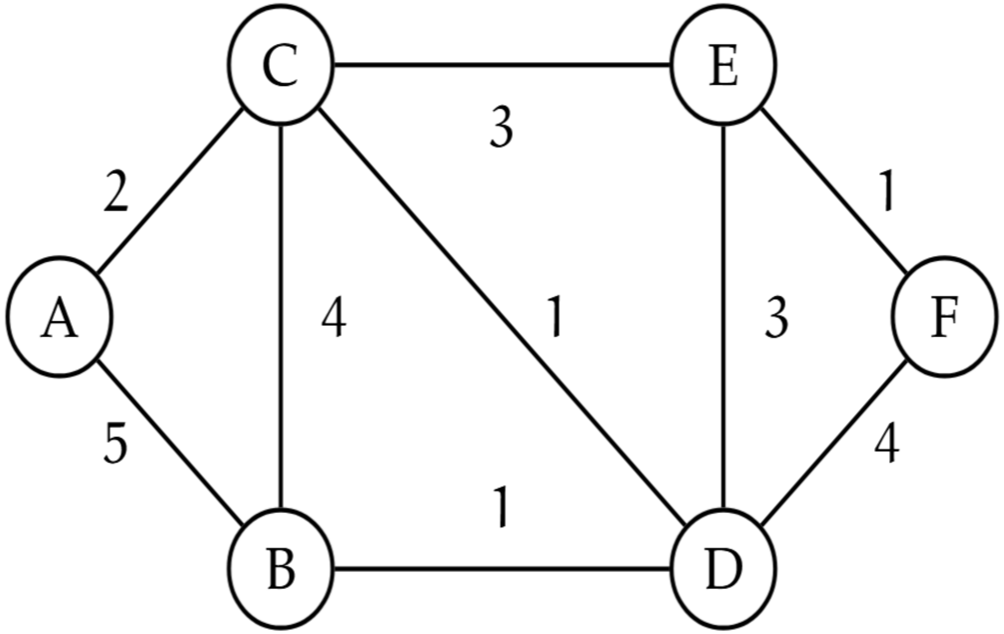
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D
3	A,C,D,B	4, D	2, A	3, C	5, C	7, D
4	A,C,D,B,E	4, D	2, A	3, C	5, C	6, E

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

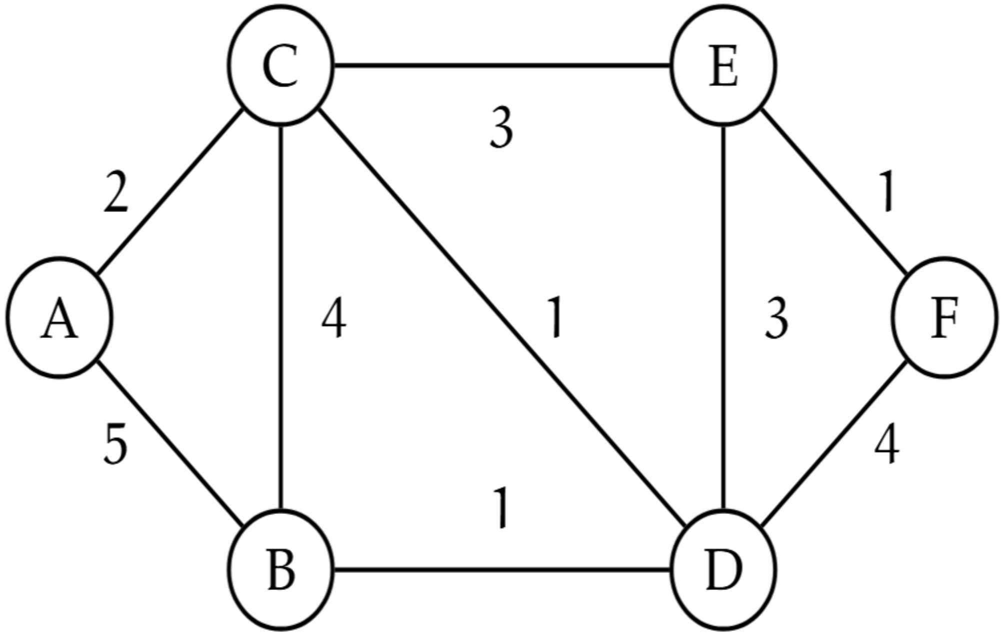
Link State Algorithm Example (Dijkstra's Algorithm)



Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D
3	A,C,D,B	4, D	2, A	3, C	5, C	7, D
4	A,C,D,B,E	4, D	2, A	3, C	5, C	6, E
5	A,C,D,B,E,F	4, D	2, A	3, C	5, C	6, E

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

Link State Algorithm Example (Dijkstra's Algorithm)

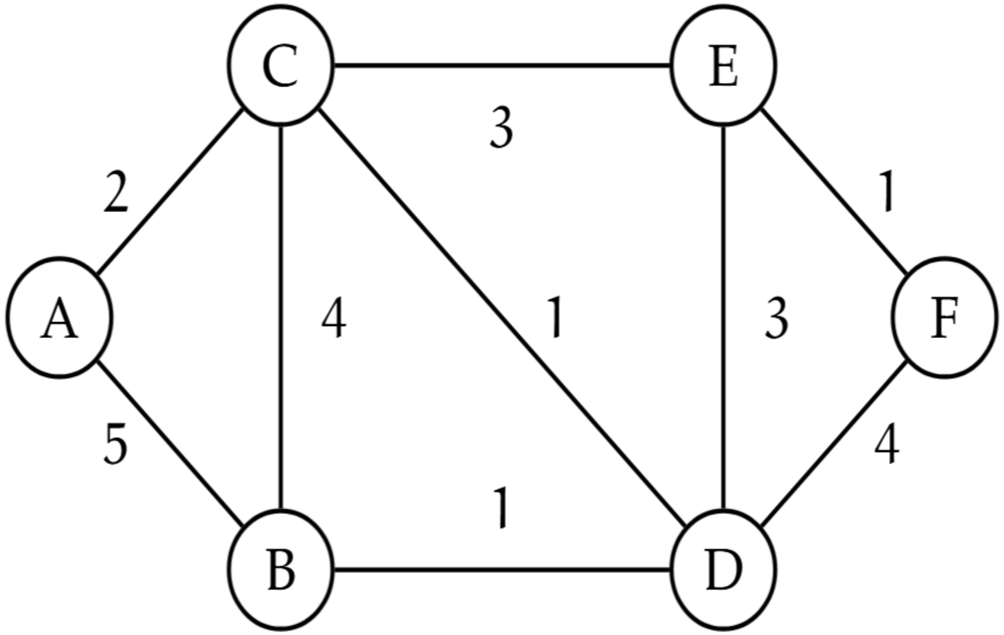


Step	N'	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D
3	A,C,D,B	4, D	2, A	3, C	5, C	7, D
4	A,C,D,B,E	4, D	2, A	3, C	5, C	6, E
5	A,C,D,B,E,F	4, D	2, A	3, C	5, C	6, E

D(X) = Distance to X from A
p(X) = Previous node to X
N' = Visited nodes

Shortest path from A to X?
Start from X and follow previous node until A!
E.g shortest path to F
F -> E -> C -> A = A -> C -> E -> F

Link State Algorithm Example (Dijkstra's Algorithm)



Step	N′	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	5, A	2, A	∞	∞	∞
1	A,C	5, A	2, A	3, C	5, C	∞
2	A,C,D	4, D	2, A	3, C	5, C	7, D
3	A,C,D,B	4, D	2, A	3, C	5, C	7, D
4	A,C,D,B,E	4, D	2, A	3, C	5, C	6, E
5	A,C,D,B,E,F	4, D	2, A	3, C	5, C	6, E

Forwarding Table:

Destination Node	Edge from A
B	(A, C)
C	(A, C)
D	(A, C)
E	(A, C)
F	(A, C)

D(X) = Distance to X from A
p(X) = Previous node to X
N′ = Visited nodes

Distance Vector Algorithm

- Based on Bellman-Ford algorithm
  - Which is based on Dijkstra's
    - Keeps track of next node instead of previous
- Cumulative/Iterative
  - Changes propagate slowly over time
- Steps
  1. Every router builds distance vector
     1. Entries for neighbors are initialized to their cost (latency and/or bandwidth)
     2. Every other entry is initialized to infinity
  2. Share own distance vector with neighbors
  3. Update own vector U, if new distances from neighbors are better than previous
     - For each node V in vector from neighbors, update distance
     - D(V) = min(D(U)+E(U,V), D(V))
       Where D(X) is distance to X from start and E(U, V) is edge cost from U to V
     - Update next node if new distance was better than previous
  4. If distance vector was updated, then share updated vector with neighbors
- This repeats naturally until no more changes

Distance Vector Algorithm Example

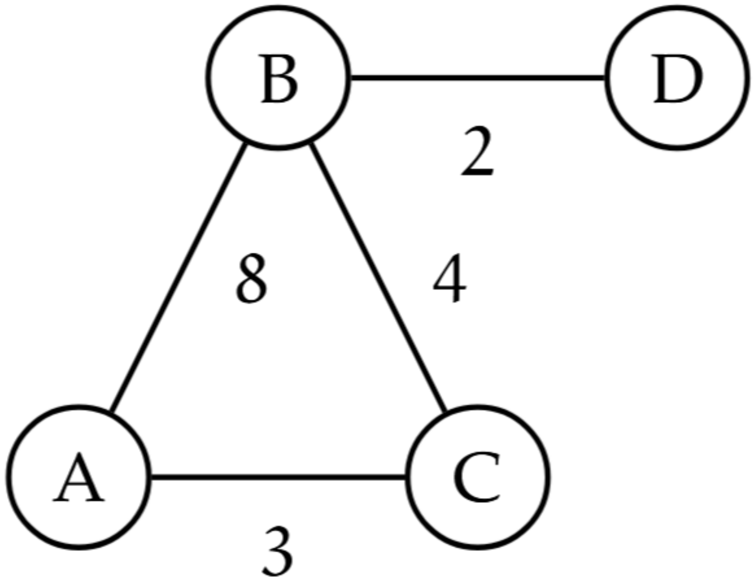
Node A Table (Init)		
To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)		
To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)		
To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

- Initialization
  - Distance is edge cost for each neighbor
  - Next is each neighbor
  - Remaining are set to infinity



Distance Vector Algorithm Example

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

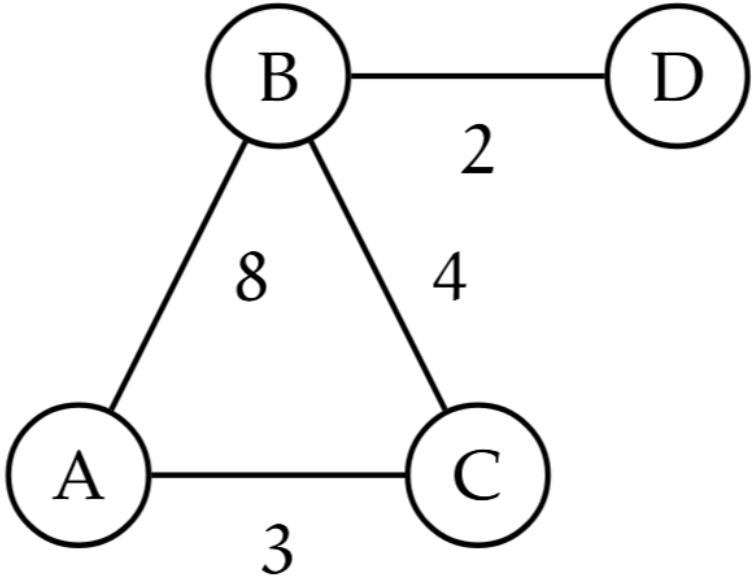
Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	10	B

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- B sends to A
  - Entry D update



Node A Table (Init)		
To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)		
To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

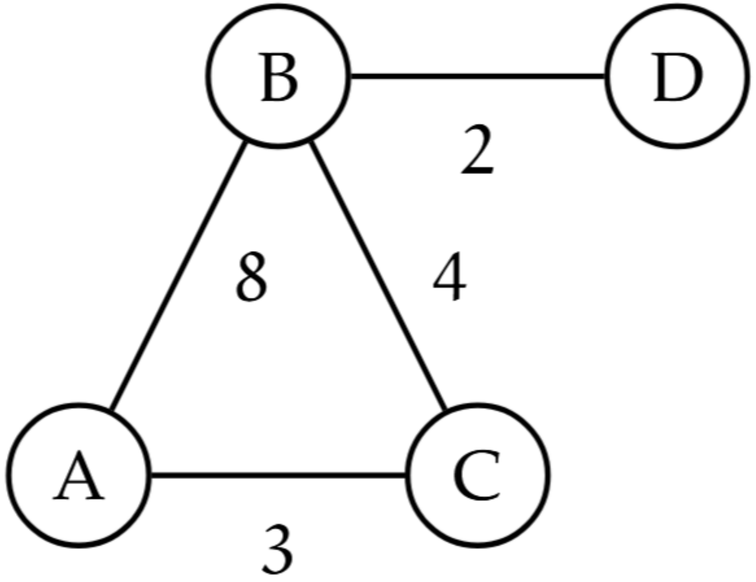
Node C Table (Init)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)		
To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

Node A Table (Iter 0)		
To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Distance Vector Algorithm Example

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- C sends to A
  - Entry B update



Distance Vector Algorithm Example

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

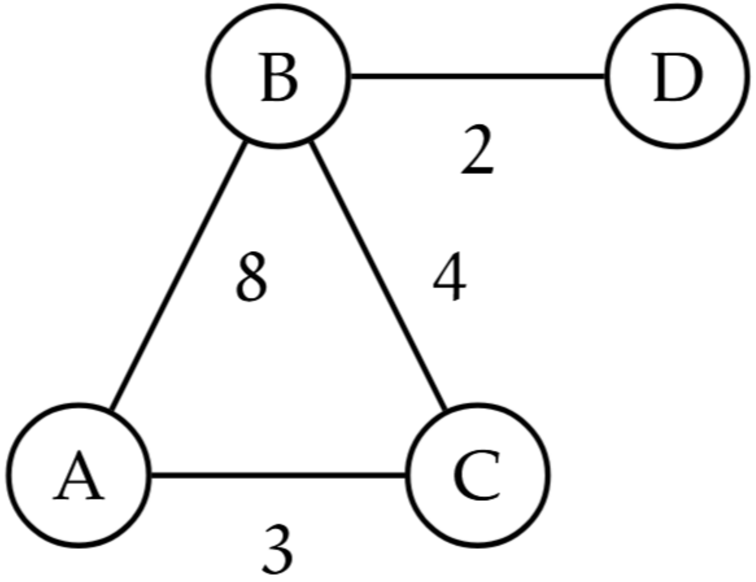
Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra’s
    - Update distance and next if better
- A sends to B
  - No update



Distance Vector Algorithm Example

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

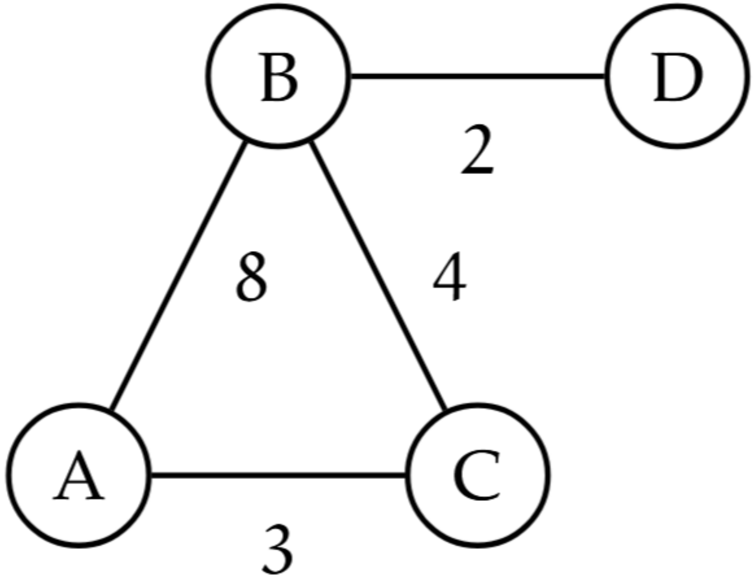
Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- C sends to B
  - Entry A update



Distance Vector Algorithm Example

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

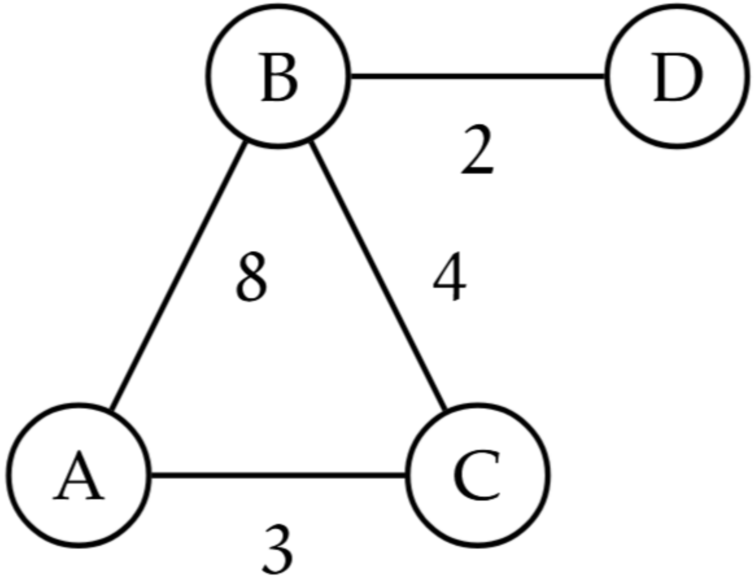
Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- D sends to B
  - No update



Distance Vector Algorithm Example

Node A Table (Init)		
To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)		
To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

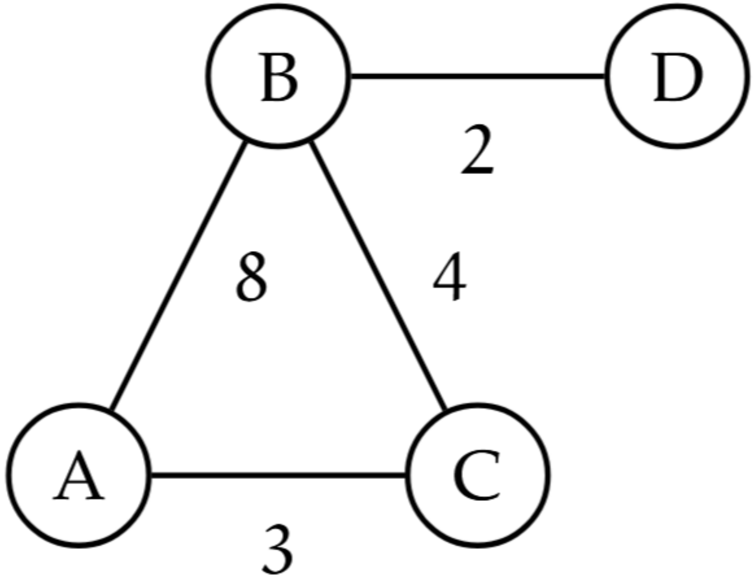
Node D Table (Init)		
To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

Node A Table (Iter 0)		
To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)		
To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 0)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- A sends to C
  - No update



Distance Vector Algorithm Example

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- B sends to C
  - Entry D update

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

Node A Table (Iter 0)

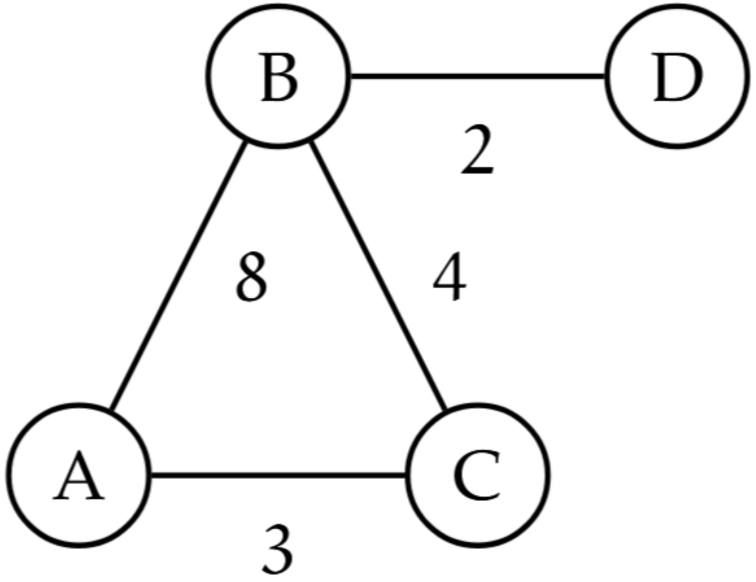
To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 0)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B



Distance Vector Algorithm Example

- Iteration 0
  - Each node sends their DV to each neighbor
  - Compare distances like in Djikstra's
    - Update distance and next if better
- B sends to D
  - Entry A and C update

Node A Table (Init)

To	Distance	Next
A	0	-
B	8	B
C	3	C
D	∞	∞

Node B Table (Init)

To	Distance	Next
A	8	A
B	0	-
C	4	C
D	2	D

Node C Table (Init)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	∞	∞

Node D Table (Init)

To	Distance	Next
A	∞	∞
B	2	B
C	∞	∞
D	0	-

Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

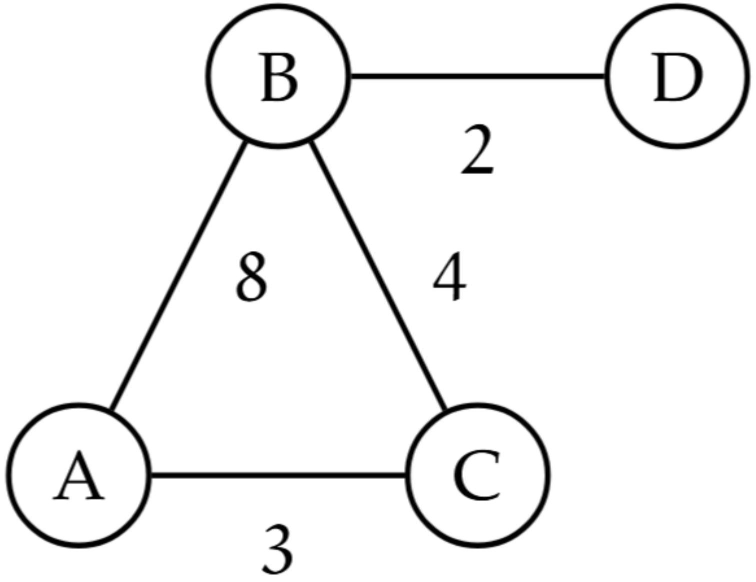
To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 0)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 0)

To	Distance	Next
A	10	B
B	2	B
C	6	B
D	0	-



Distance Vector Algorithm Example

Node A Table (Iter 0)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	10	B

Node B Table (Iter 0)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 0)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 0)

To	Distance	Next
A	10	B
B	2	B
C	6	B
D	0	-

Node A Table (Iter 1)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	9	C

Node B Table (Iter 1)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

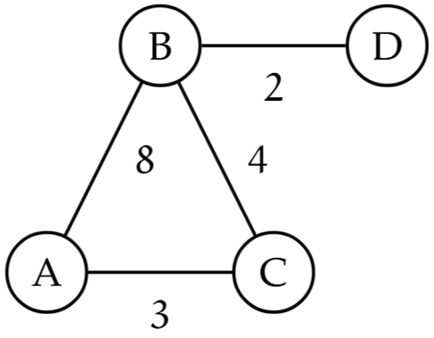
Node C Table (Iter 1)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 1)

To	Distance	Next
A	9	B
B	2	B
C	6	B
D	0	-

- Iteration 1
  - Just like iteration 0
  - I will skip some steps and do everything at once
- Everyone sends to their neighbors!
  - A gets from B and C
    - Entry D update from C
  - B gets from A, C and D
    - No update
  - C gets from A and B
    - No update
  - D gets from B
    - Entry A update from B



Distance Vector Algorithm Example

Node A Table (Iter 1)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	9	C

Node B Table (Iter 1)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 1)

To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 1)

To	Distance	Next
A	9	B
B	2	B
C	6	B
D	0	-

Node A Table (Iter 2)

To	Distance	Next
A	0	-
B	7	C
C	3	C
D	9	C

Node B Table (Iter 2)

To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 2)

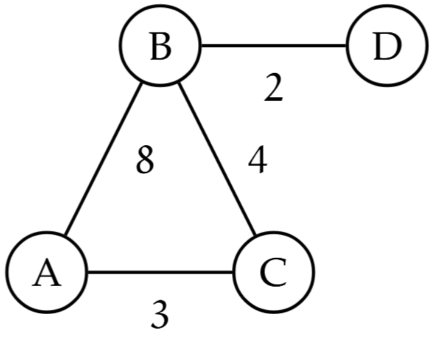
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 2)

To	Distance	Next
A	9	B
B	2	B
C	6	B
D	0	-

- Iteration 2
  - Just like iteration 1
  - I will skip some steps and do everything at once
- Everyone sends to their neighbors!
  - A gets from B and C
    - No update
  - B gets from A, C and D
    - No update
  - C gets from A and B
    - No update
  - D gets from B
    - No update

No more updates
We are done!



Distance Vector Algorithm Example

Node A Table (Iter 1)		
To	Distance	Next
A	0	-
B	7	C
C	3	C
D	9	C

Node B Table (Iter 1)		
To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 1)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

Node D Table (Iter 1)		
To	Distance	Next
A	9	B
B	2	B
C	6	B
D	0	-

Node A Table (Iter 2)		
To	Distance	Next
A	0	-
B	7	C
C	3	C
D	9	C

Node B Table (Iter 2)		
To	Distance	Next
A	7	C
B	0	-
C	4	C
D	2	D

Node C Table (Iter 2)		
To	Distance	Next
A	3	A
B	4	B
C	0	-
D	6	B

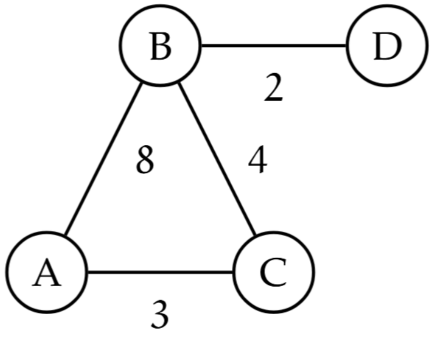
Node D Table (Iter 2)		
To	Distance	Next
A	9	B
B	2	B
C	6	B
D	0	-

Forwarding Table:

Destination Node	Edge from A
B	(A, C)
C	(A, C)
D	(A, C)

Distances:

From	To			
	A	B	C	D
A	0	7	3	9
B	7	0	4	2
C	3	4	0	6
D	9	2	6	0



Questions & comments?