The Networking Layer

Routing

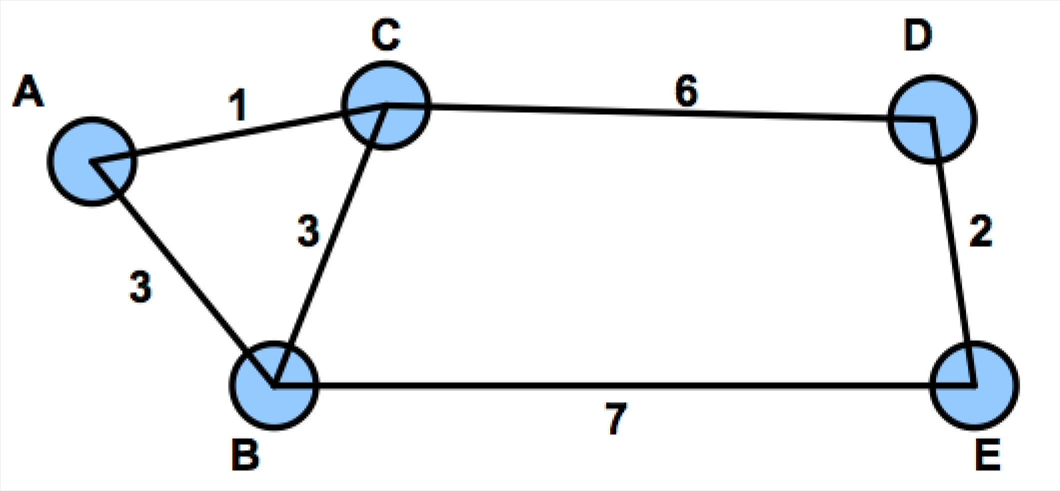
Routing

Routing is path-finding

- Routing creates FIB (that forwarding looks up)
  - Lookup Mechanisms can be:
    - Per-packet
    - Per-connection setup (traffic context)
    - Intermediate (partial path, aggregation) also possible

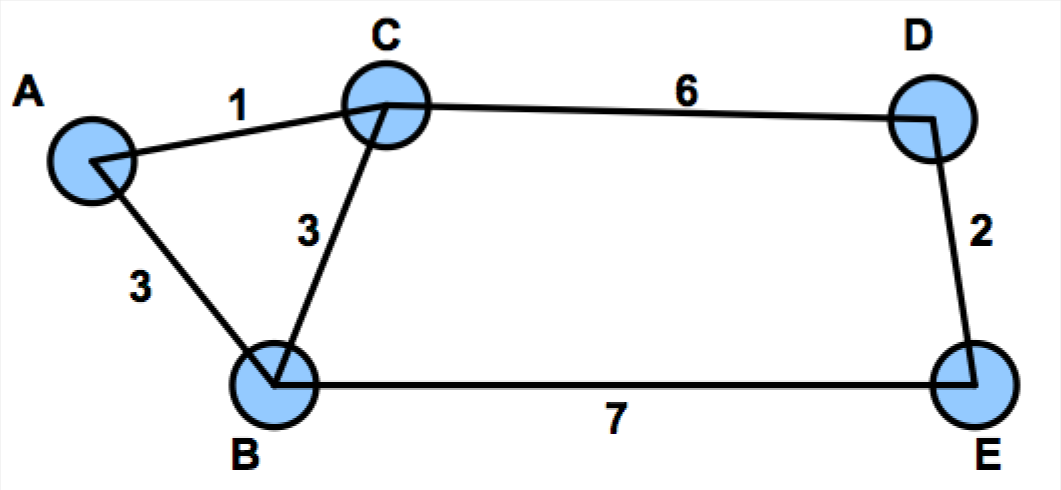
Network Routing Problem

- Select a path for traffic in a network
- Which one is the best path between A and D?



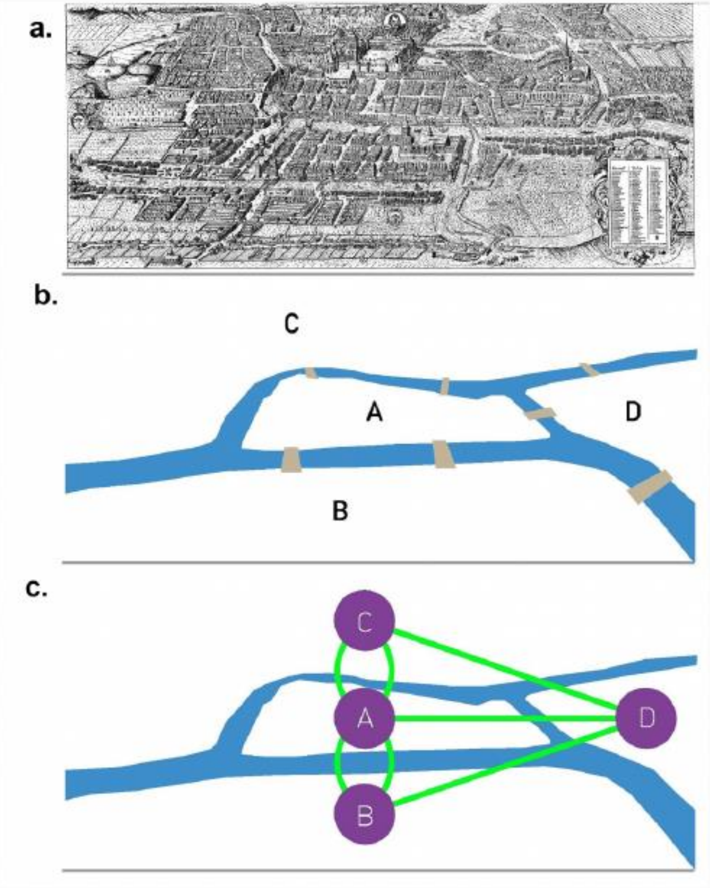
Network Routing Problem

- Criteria: (from A to D)
  - the value indicates the path length?
  - the value indicates the link capacity?
  - the value indicates the network reward?

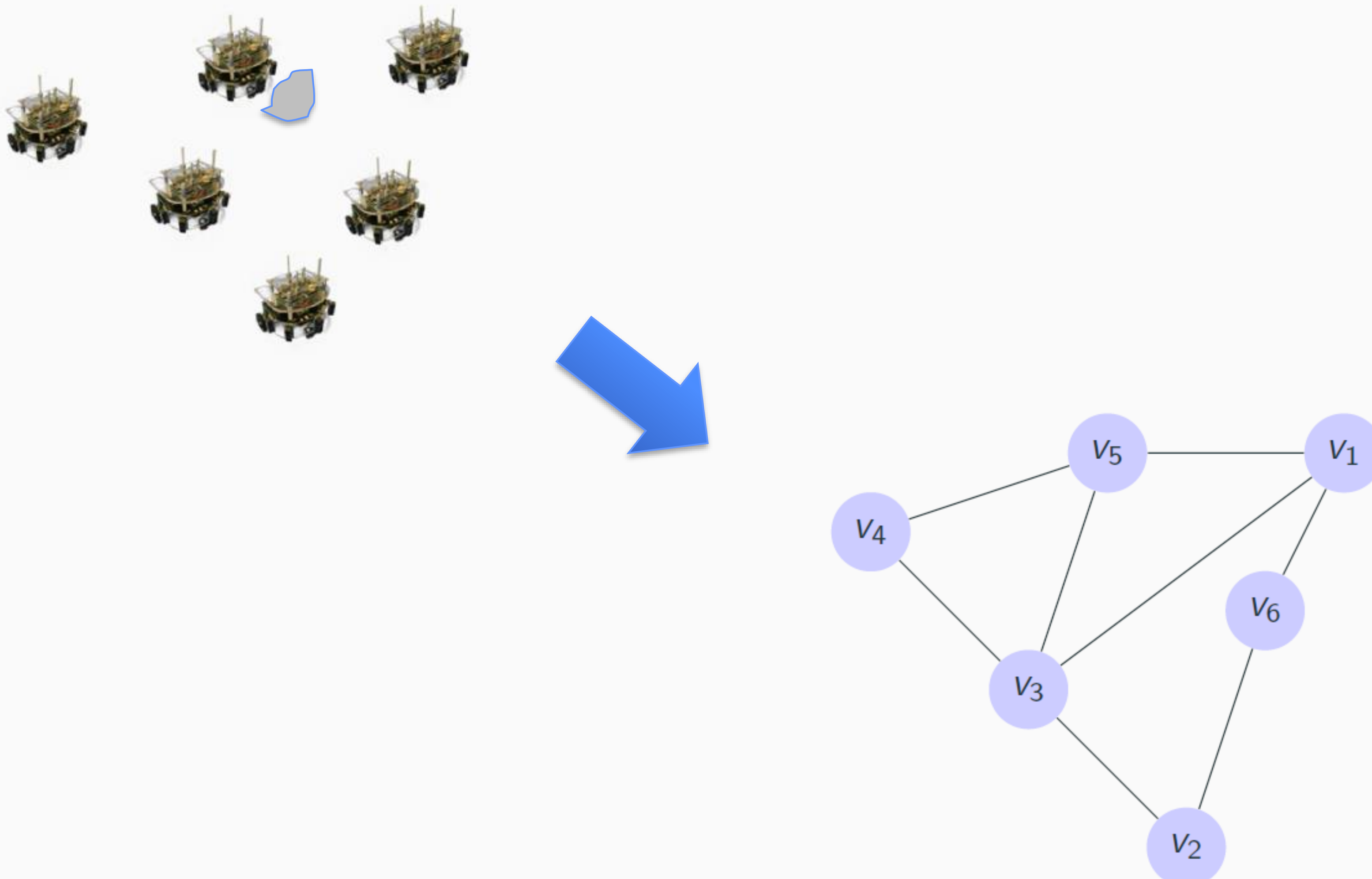


Network Graph Models

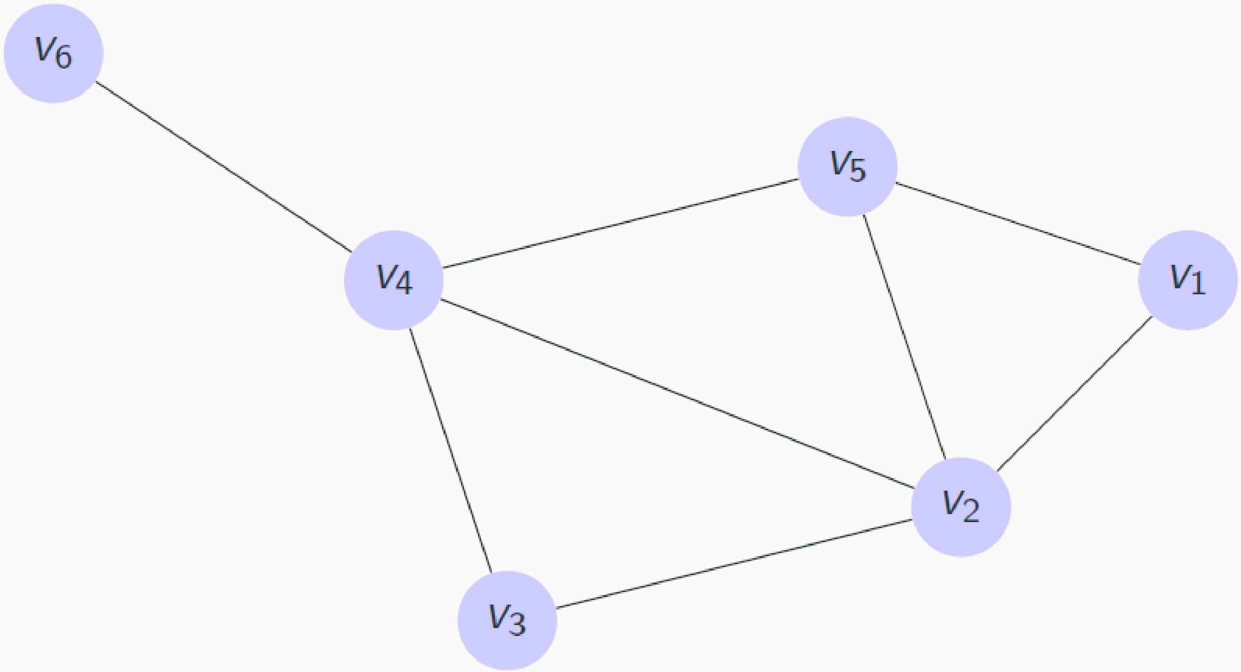
From Networks to Graphs



The Right-level of Abstraction



A Simple Graph



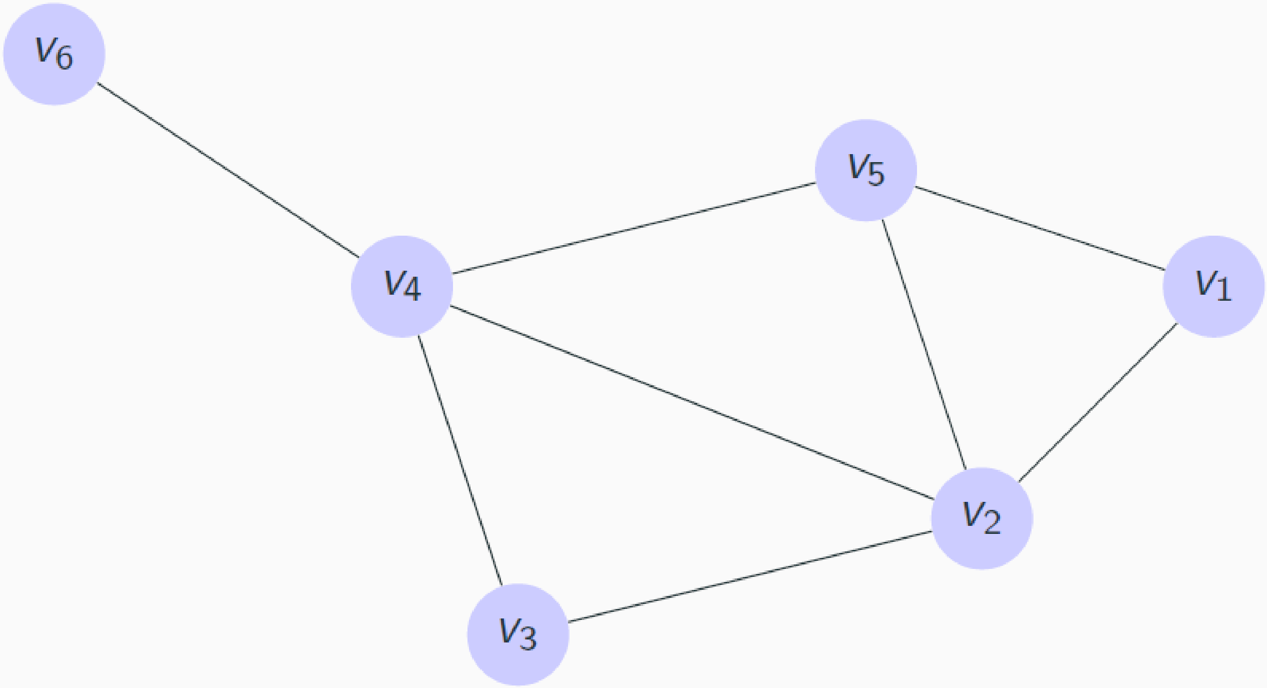
A graph G consists of a set of vertices V and edges E

$$G = (V, E)$$

where

$$\begin{aligned} V &= \{v_1, v_2, v_3, v_4, v_r, v_6\} \\ E &\subseteq V \times V \end{aligned}$$

A Simple Graph

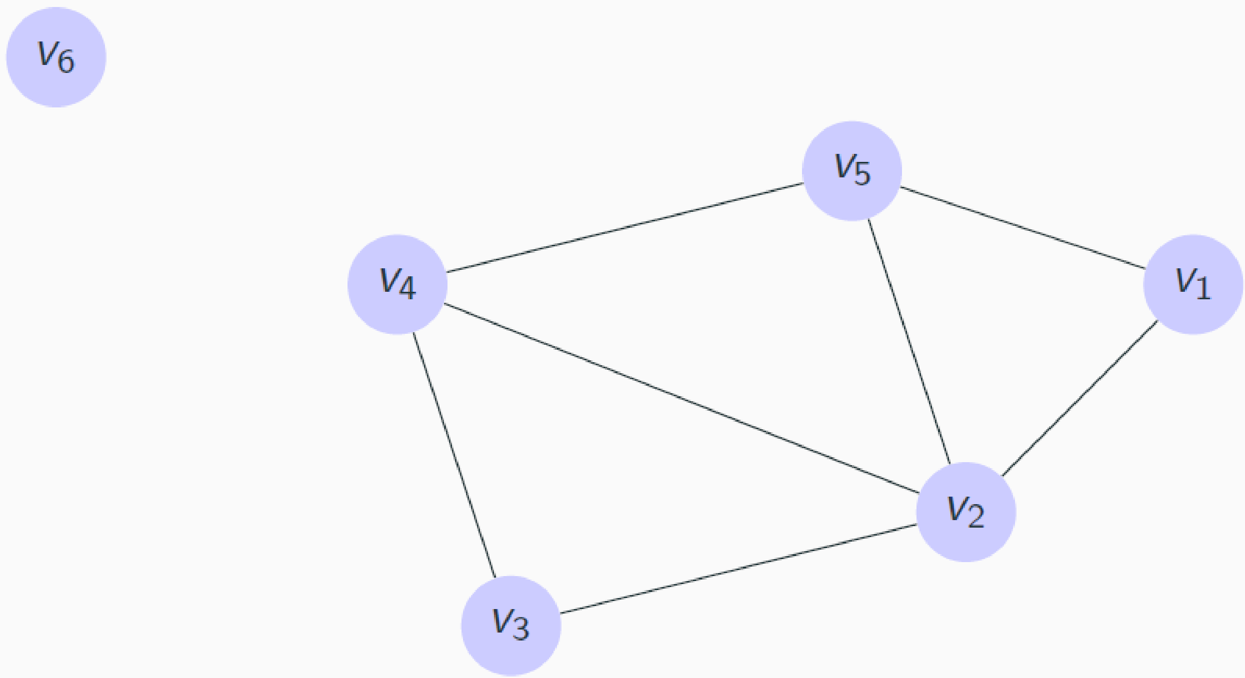


In our case,

$$E = \{(v_1, v_2), (v_1, v_5), (v_2, v_3), (v_2, v_4), (v_2, v_5), (v_3, v_4), (v_4, v_5), (v_4, v_6)\}$$

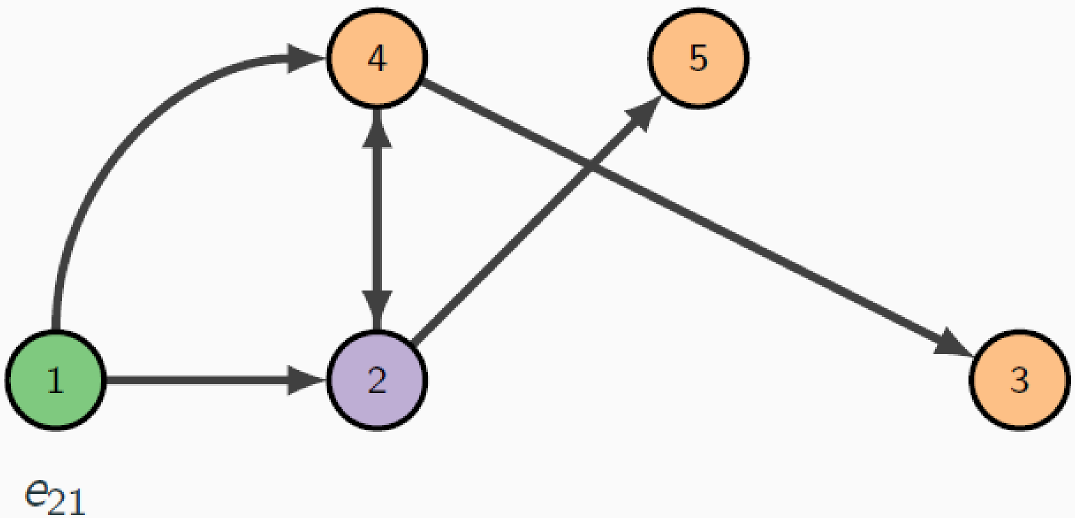
This is an *undirected* graph, i.e. $(v_i, v_j) \in E \iff (v_j, v_i) \in E$

Connected vs Disconnected



An undirected graph is *connected* if there is a path between any two nodes. Otherwise, it is *disconnected*.

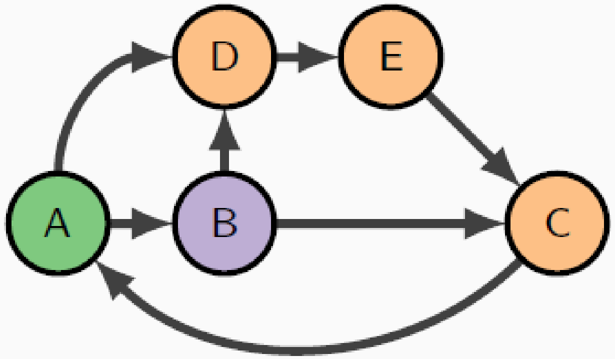
Directed Graph



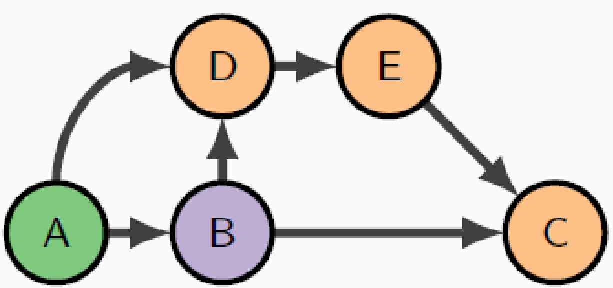
This is a *directed* graph or *digraph*, where each edge has a tail and a head, i.e. $e_{ij} = (v_i, v_j) \in E$ where v_i is the head and v_j is the tail.

Directed Graph

Strongly Connected



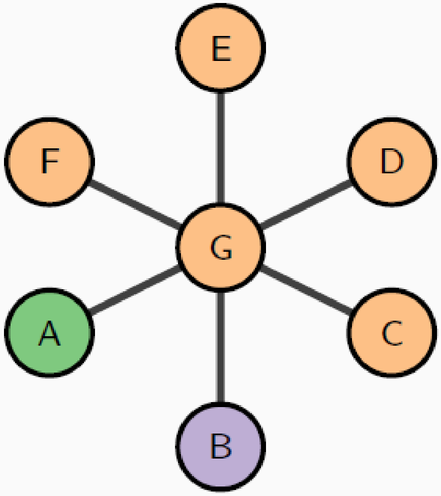
A directed graph is *strongly connected* if a directed path exists between any two nodes. It is *weakly connected* if its undirected version is connected.

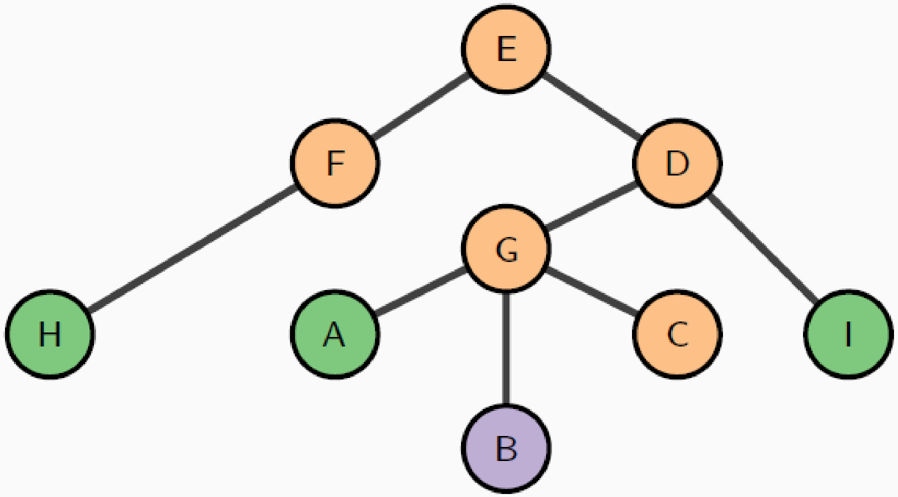


Weakly Connected

Examples

A $Star_N$ graph is a star graph of N nodes.





A *Tree* is a graph without cycles

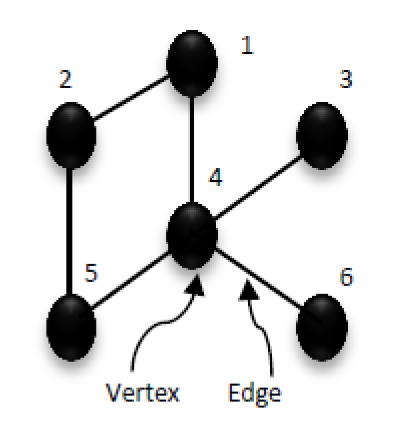
Examples

A mesh graph

Mathematical Representation

- The adjacency matrix A of a graph is a matrix with elements A_{ij} such that:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Mathematical Representation

Two things to notice in this example:

- The diagonal elements of the matrix are zeros -- No self-edges

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Mathematical Representation

Two things to notice in this example:

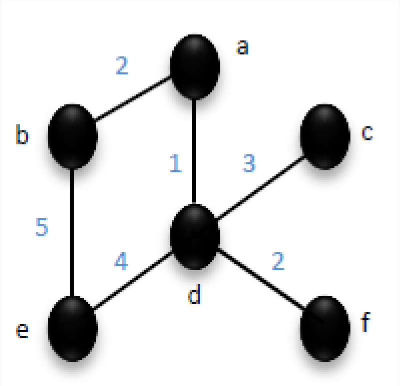
- The matrix is symmetric -- Undirected graph

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Weighted Networks

- Useful to represent the weight or the strength of the connections between the vertices.

- Mathematically the elements of the adjacency matrix are equal to the weight of the edges.



$$B = \begin{pmatrix} 0 & 2 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 1 & 0 & 3 & 0 & 4 & 2 \\ 0 & 5 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \end{pmatrix}$$

Routing - Fundamentals

- Different concerns
  - Correctness and optimality
  - Robustness, stability, …
  - More complicated “policy” concerns
  - Optimality
- Routing may be fixed (a pre-determined path)
- May be adaptive, or dynamic
  - Adapting to network state (processing overhead)

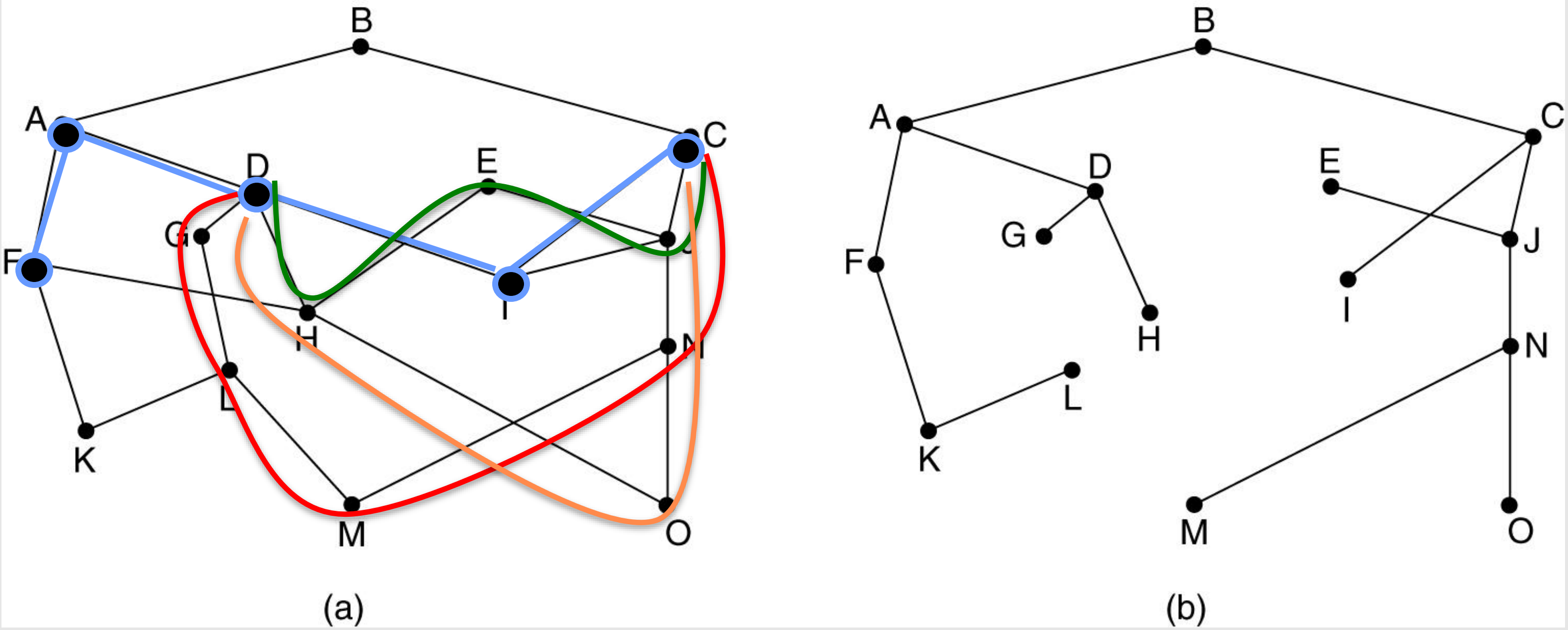
Space of Routing Choices

- Static vs. dynamic or adaptive
- For a node, next-hop determined by
  - Destination only
  - Source and destination
  - Particular flow of traffic
  - Other characteristics
- Next-hop may be unique or multiple
  - Deterministically or randomly picked
- Control and management
  - How is input information for routing collected?
  - How/where is routing algorithm run?
  - How is routing output written to FIBs?

Flooding

- No lookup! Multiple next-hop
  - Simple – forward to everybody else (except neighbor received from), but consume the bandwidth
  - Generates unbounded copies
  - Hop count can be introduced to bound (TTL)
  - Can be more intelligent if packets can be recognized
- Highly robust, simple design
  - Serves as benchmark
- Can introduce “reaction window”
  - See if copies coming from multiple neighbors nearly simultaneously (skip both if so)

"Optimality Condition"

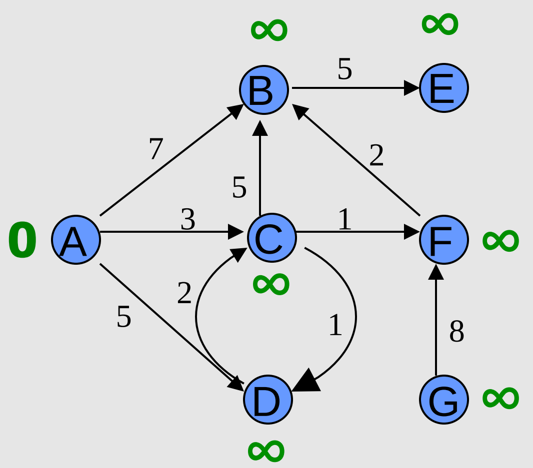


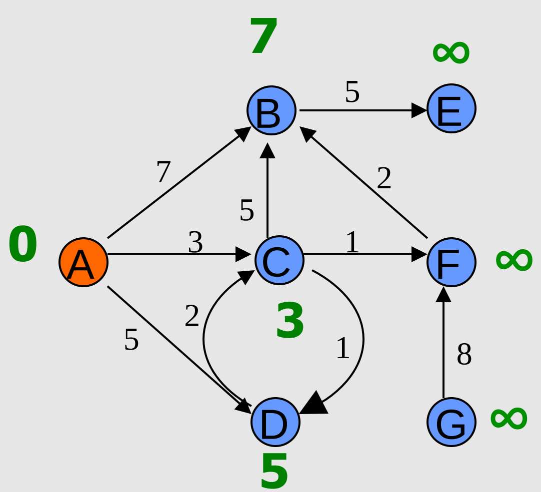
- Nice characteristic, works for "distance" or "cost" related routing goals
  - "Sub-path of optimal path must be optimal"

Finding Shortest Paths

- An example where a graph-theoretic abstraction is useful
  - Abstract networking into vertices and arcs
  - Weighted arcs – least cost approach
- Dijkstra's algorithm
  - Finds least cost path to all vertices from one
  - Based on moving vertices from a set with "tentative" distances to a set with "fixed" distances
  - Desirable computational properties
  - Other algorithms exist
- Useful for additive measures of path cost
  - "Shortest" is better understood as "least additive cost"

Dijkstra's Algorithm



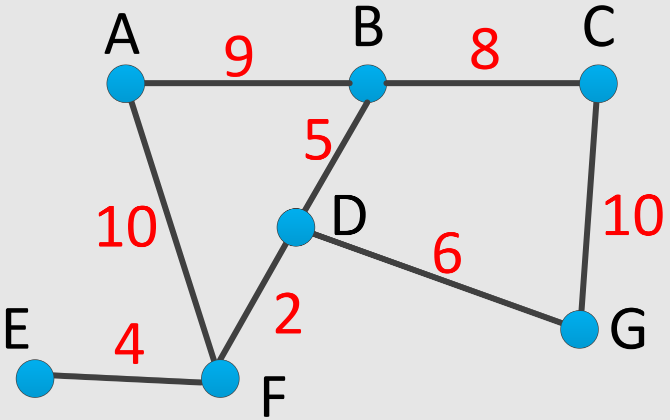


- Finding shortest distances (and corresponding paths) from ONE source to ALL destinations
  - Easily adapted to vice-versa
- First, mark source node with tentative distance zero
- Mark all other nodes tentatively with distance "infinity" (not reachable)
- Do iteratively (until all nodes marked):
  - Pick the node with the least tentative distance, and mark it as final distance – call it node X
  - Pick all outgoing links from this node, for each:
    - Update the tentative distance of the neighbor ***IF***:
    - (Final distance of X + link cost) < Tentative distance of neighbor
    - If updated, remember X as previous node of neighbor

At the end, previous nodes can be read off to find complete paths

Routing Algorithms

- Dijkstra Shortest Path Algorithm:
  - single source to any destinations
  - e.g., values represent accumulated delays (propagation, transmission, queueing, processing), or hop count
  - distance array

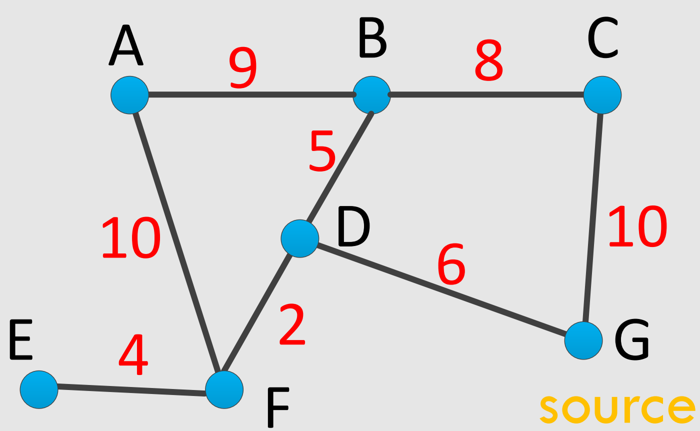


Routing Algorithms

- Dijkstra Shortest Path Algorithm:

  - example

D(X) means the distance from the source to a node X, and p(X) means the parent of a node X in the computed path.

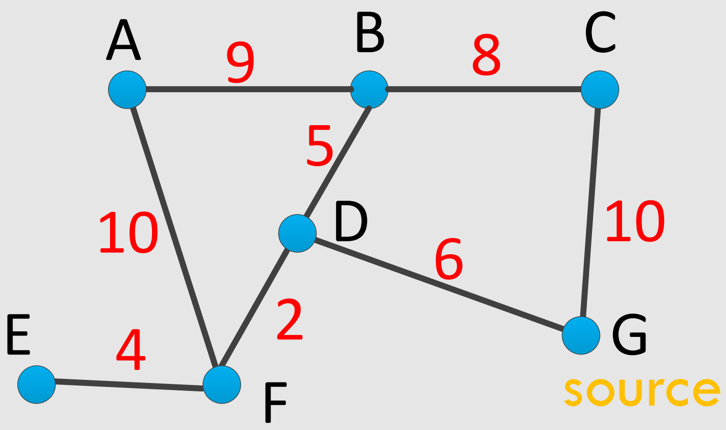


distance array

Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /

Routing Algorithms

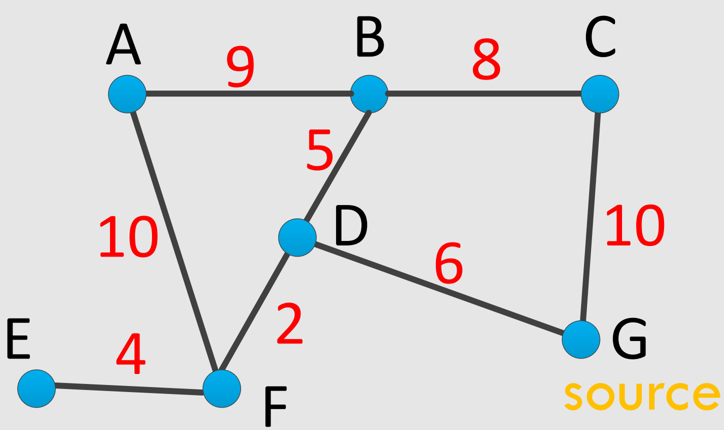
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	

Routing Algorithms

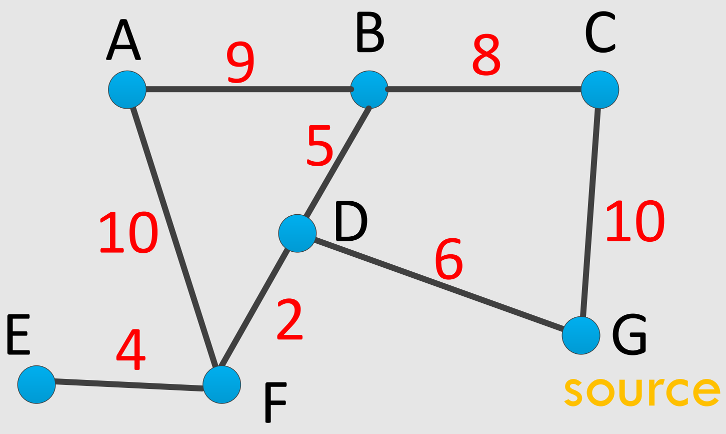
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	

Routing Algorithms

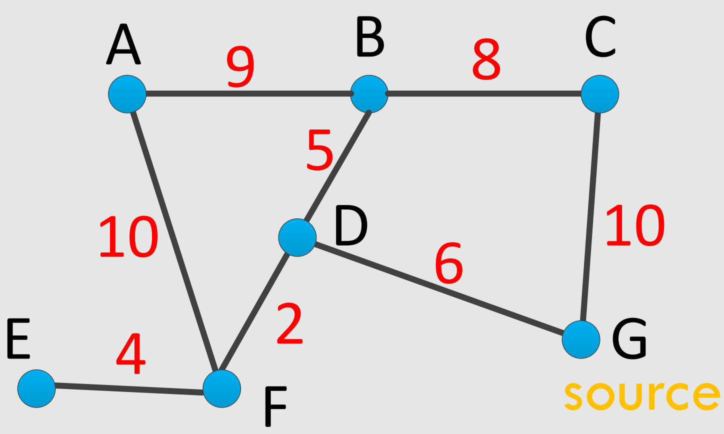
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	
3	GDF	18,F	11,D	10,G		12,F		

Routing Algorithms

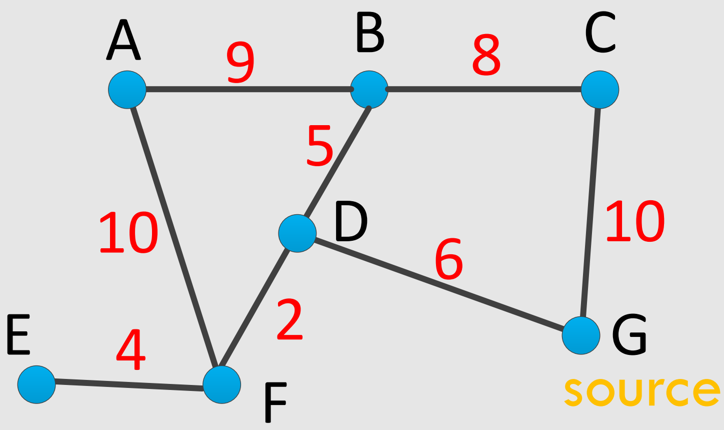
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	
3	GDF	18,F	11,D	10,G		12,F		
4	GDF C	18,F	11,D			12,F		

Routing Algorithms

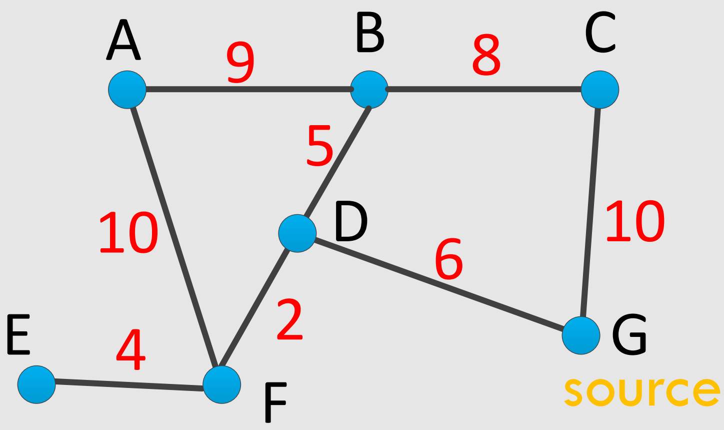
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	
3	GDF	18,F	11,D	10,G		12,F		
4	GDF C	18,F	11,D			12,F		
5	GDF CB	18,F				12,F		

Routing Algorithms

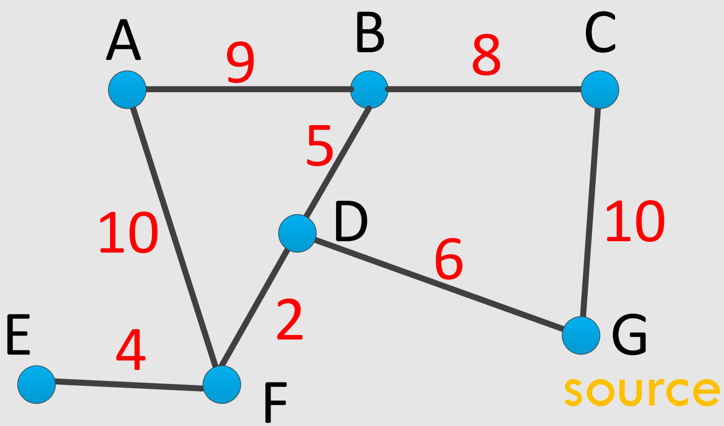
- Dijkstra Shortest Path Algorithm:



Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	
3	GDF	18,F	11,D	10,G		12,F		
4	GDF C	18,F	11,D			12,F		
5	GDF CB	18,F				12,F		
6	GDF CBE	18,F						

Routing Algorithms

- Dijkstra Shortest Path Algorithm:

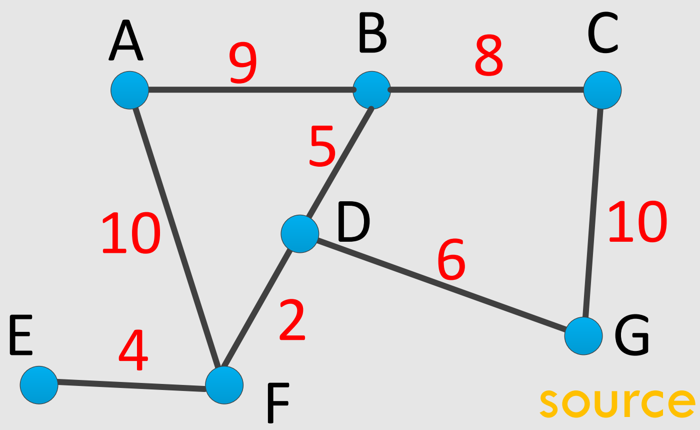


Step		D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)
0	None	∞	∞	∞	∞	∞	∞	0, /
1	G	∞	∞	10,G	6,G	∞	∞	
2	GD	∞	11,D	10,G		∞	8,D	
3	GDF	18,F	11,D	10,G		12,F		
4	GDF C	18,F	11,D			12,F		
5	GDF CB	18,F				12,F		
6	GDF CBE	18,F						
7	GDF CBEA							

Routing Algorithms

- Dijkstra Shortest Path Algorithm:

  - example



distance array

A	B	C	D	E	F
18, F	11, D	10, G	6, G	12, F	8, D

Routing table of G:

Dest.	Next hop
A	D
B	D
C	C
D	D
E	D
F	D

Backtrack by the p(X) field until the source,
then find the next-hop for each destination from G

Routing Protocols

- Each node runs the routing algorithm (e.g., Dijkstra algorithm), then maintain its own routing table
- When packet arrives, forward to next hop …

- How is the routing table constructed?

- Distance vector vs. Link state

  - distributed methods for building routing tables that converge to the shortest path tables

  - 1) collect topology information; 2) run routing algorithm

Embedding in Switch/Router

- Protocols have to be embedded in realization
- Routing protocols must run on router
  - Conventional wisdom: distributed is scalable/robust
- Routers must possess control plane
  - Signaling – exchange of control signals
- Control plane is client of data plane

Routing Protocols

- Each node runs the routing algorithm (e.g., Dijkstra algorithm), then maintain its own routing table
- When packet arrives, forward to next hop …

- How is the routing table constructed?

- Distance vector vs. Link state
  - distributed methods for building routing tables that converge to the shortest path tables
  - 1) collect topology information; 2) run routing algorithm

Link State Protocols

- Each router distributes information it has to every other router
- Information is in the form of *Link State Announcements (LSAs)*
  - (myID, neighborID, link cost)
- Distribution is by controlled flooding
  - Distribute along each link but receiving one
- Now, each router has complete information
- OSPF is a prevalent example
  - SPF with LSA to find arc costs

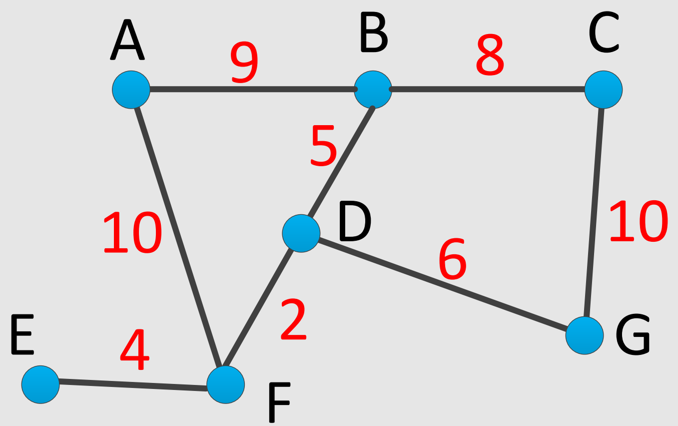
LSA Routing Algorithm

Each router does the following

1. "Meets the (immediately adjacent) neighbors" and learns their IDs
2. Builds an LSA containing IDs and distance to each of its neighbors
3. Transmits the LSA to <u>all</u> other routers
4. Stores the most recent LSA from <u>every other router</u> in the network
   - Not just from its immediate neighbors!
5. Creates a "map" of the network topology from LSAs
6. Computes routes (to store in forwarding table) from its local map of the topology

Link State Routing

- Open Shortest Path First (OSPF)
- Nodes exchange link state advertisements with their neighbors
- Link State Advertisements: info about links connected to a node (LSA) (delay + node ID of other end of link)



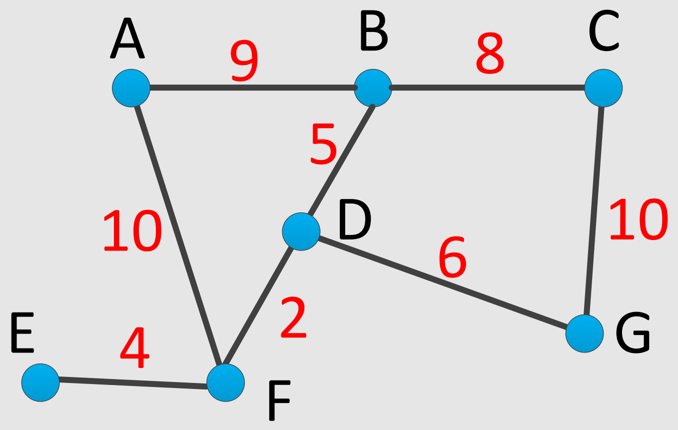
G receives LSA from C and D:

C: [B, 8; G, 10]

D: [B, 5; F, 2; G, 6]

- When a node receives an LSA, it forwards it on all of its links

Link State Routing



What is the next after G knows everything?

Run Dijkstra algorithm directly

Round 1:

G receives LSA from C and D:

C: [B, 8; G, 10]

D: [B, 5; F, 2; G, 6]

Round 2:

G receives LSA from B and F:

B: [A, 9; C, 8; D, 5]

F: [A, 10; D, 2; E, 4]

Round 3:

G receives LSA from A and E:

A: [B, 9; F, 10]

E: [F, 4]

Generating LSAs

- A router generates LSAs *periodically* → background refresh rate
- A router also generates LSAs when its local environment changes
  - it has a new neighbor (router comes online)
  - a link goes down (indicated by absence of "Hello" packets)
  - the cost of a link to an existing neighbor has changed
- Limiting the overhead (network bandwidth) consumed by routing messages, particularly LSAs
  - set a minimum interval between successive updates

Forwarding LSAs – Control Overhead

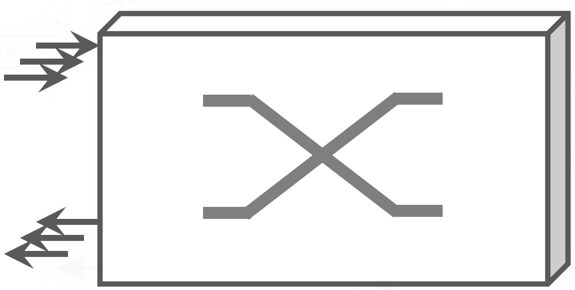
- LSA flooding can use information in LSA
  - Each router stores the most recent copy of LSAs from all routers → easy to recognize duplicate LSAs
  - Each router floods an LSA only once
    - An LSA will travel over each link of the network exactly once

Link State Routing (Summary)

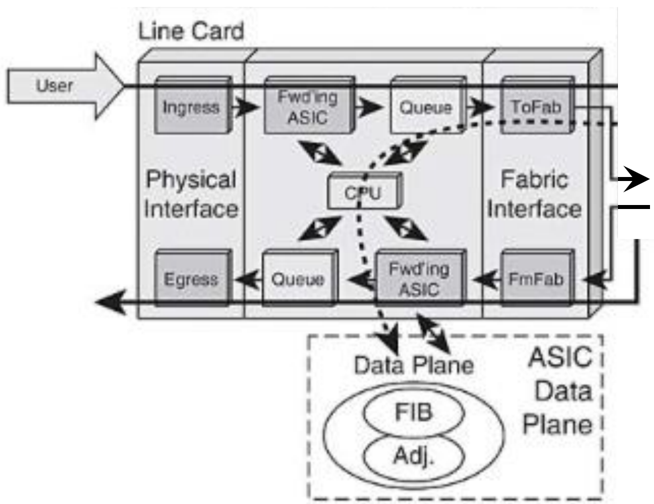
- Based on global knowledge
- Converges Faster
- No count to infinity problem

- But …
- Require more network resources (bandwidth)
- Heavy traffic due to flooding of packets
- Flooding may result in infinite looping which can be solved by using the Time to live (TTL) field

Switch/Router Architecture



Switch/Router Architecture



Switch/Router Architecture