CH. 3
PART 3: IP ROUTING PROTOCOLS

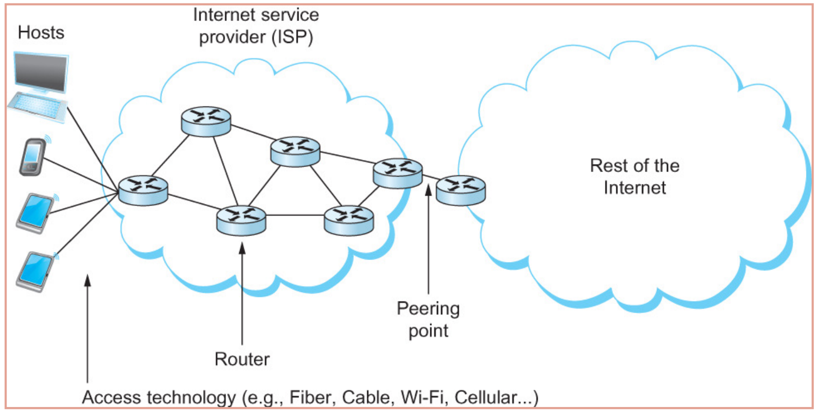
V1.3

Lecture on how routers communicate over the control plane for sharing routing information

Computer Networks Course, Universidad de León, 2018

Chapter Outline

- The extended LAN
- Bridges and switches
  - ST algorithm
- Routers have two communication planes:
  - Data plane: Forwarding with Longest Prefix Matching
  - Control plane: Routing
    - Bellman-Ford Algorithm (A Distance Vector Algorithm)
    - Dijkstra Algorithm (A Link State Protocol)
- IP addressing (Already done, Lab)
- Performance of switches and routers

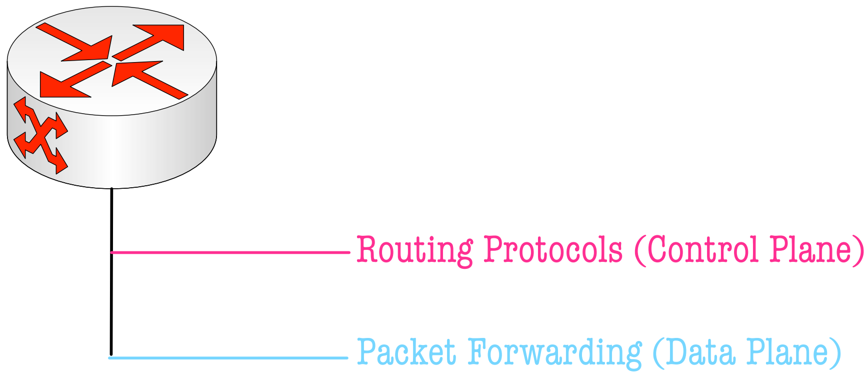


What is Routing?

Forwarding vs. Routing

- Forwarding:
  - To select an output port based on destination address and routing table
  - Longest Prefix Matching
- Routing:
  - Process whereby the routing table is built

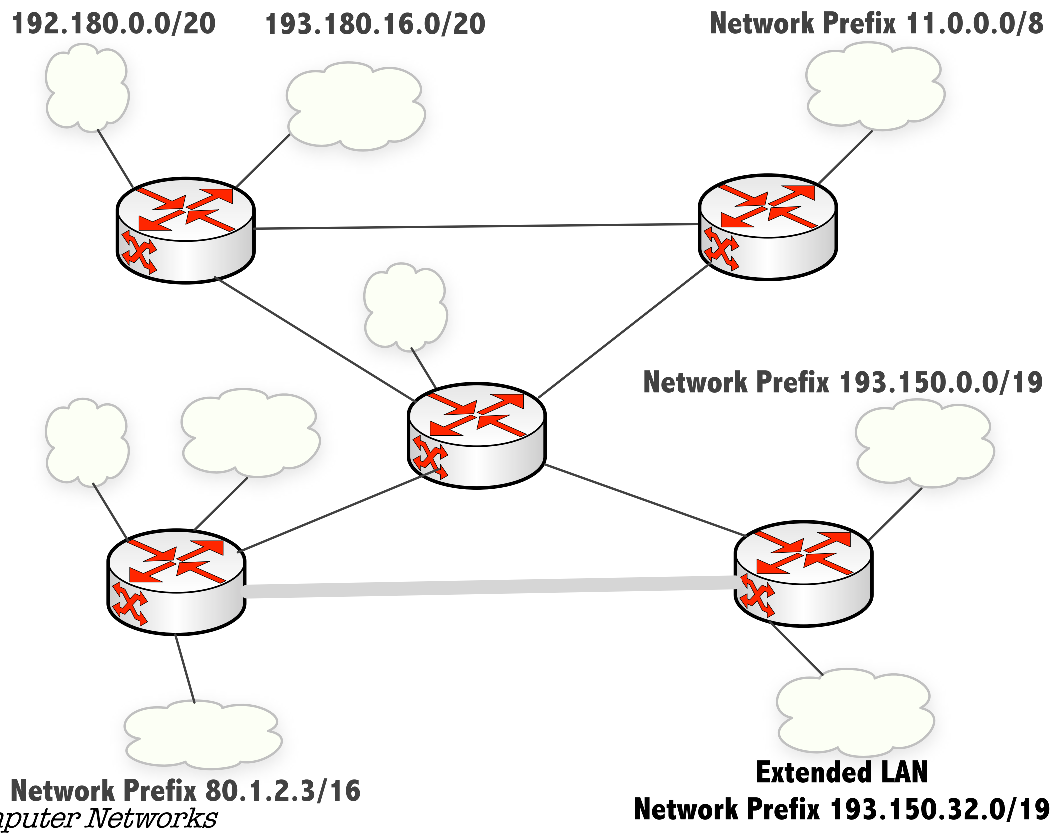
The two basic functions of an IP Router



Based on textbook Conceptual Computer Networks
© 2013-2018 by José María Foces Morán
& José María Foces Vivancos

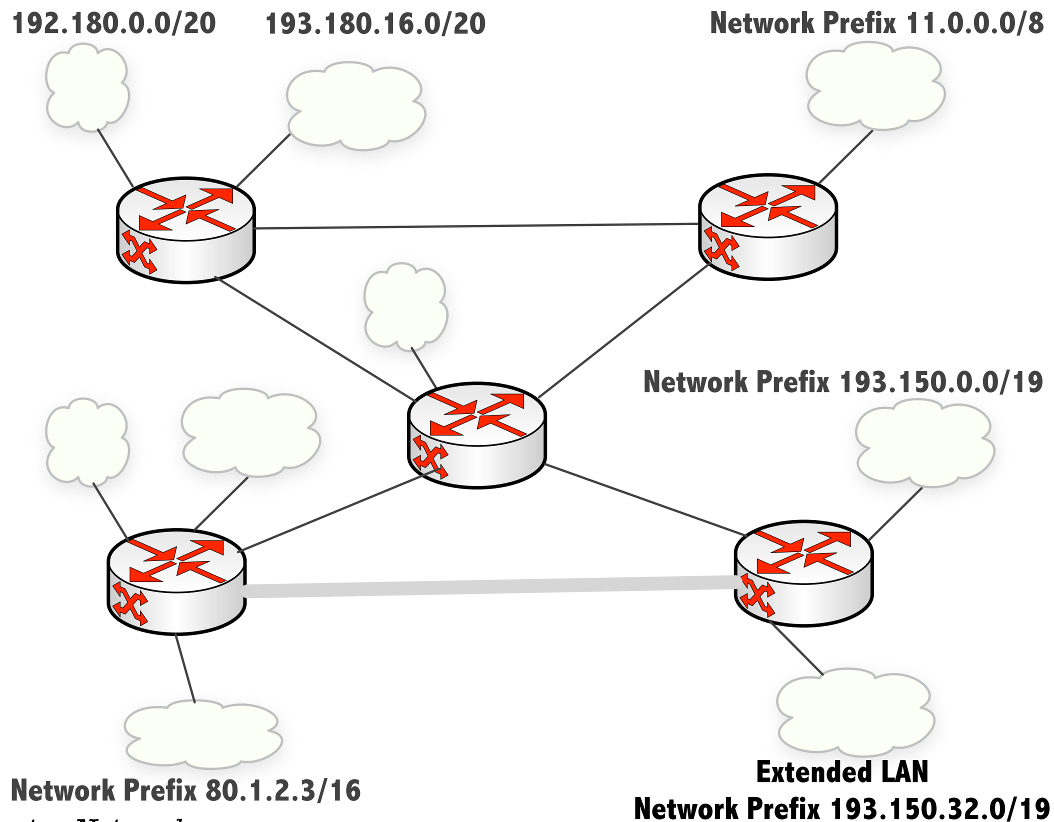
What is Routing?

In an internetwork, managing the routing tables of each router is difficult and error prone. How can this problem be solved?



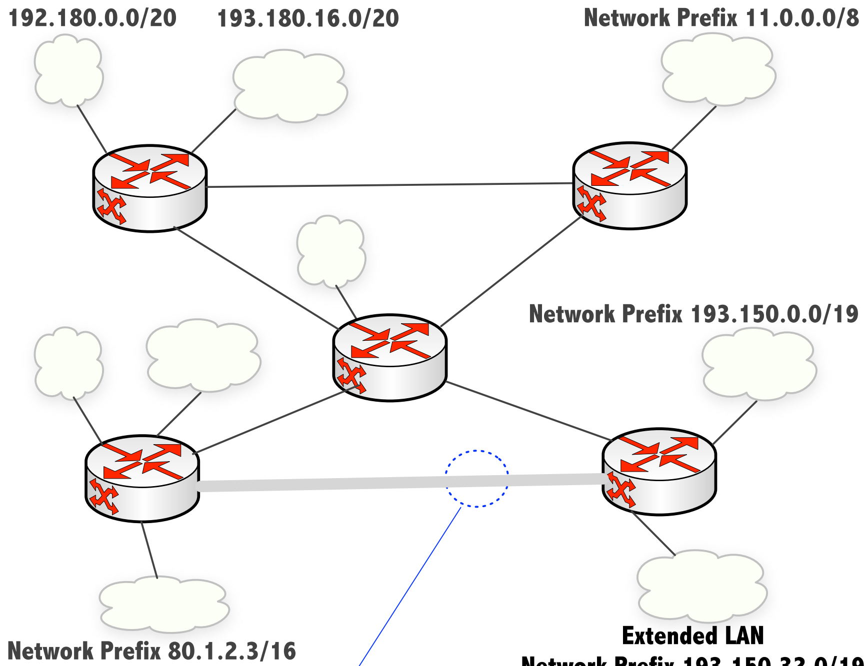
What is Routing?

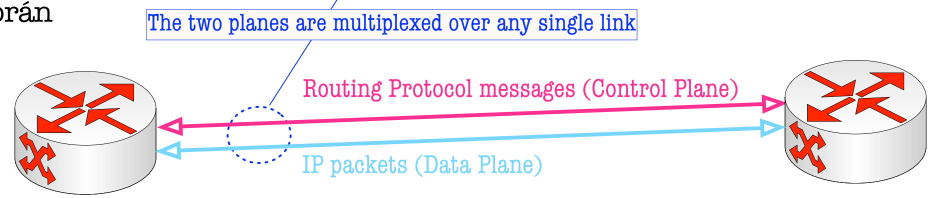
How can this problem be solved? Not by having an administrator enter the routing tables statically



What is Routing?

Solution: Have routers share routing information that enables them to build the routing tables automatically





Forwarding *vs.* routing tables

(a)	
Prefix/Length	Next Hop
18/8	171.69.245.10

(b)		
Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

- Forwarding table (b)
  - Used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function
  - A row in the forwarding table contains the mapping from a **network number to an outgoing interface** and some MAC information, such as Ethernet Address of the next hop

- Routing table (a)
  - Built by the routing algorithm as a precursor to build the forwarding table
  - Generally contains mapping from **network numbers to next hops**

Forwarding *vs.* routing tables: An example

- Example rows from (a) routing and (b) forwarding tables
  - Prefix/length = network number/cidr
  - Next hop: IP address of next router in a directly connected network

(a)

Prefix/Length	Next Hop
18/8	171.69.245.10

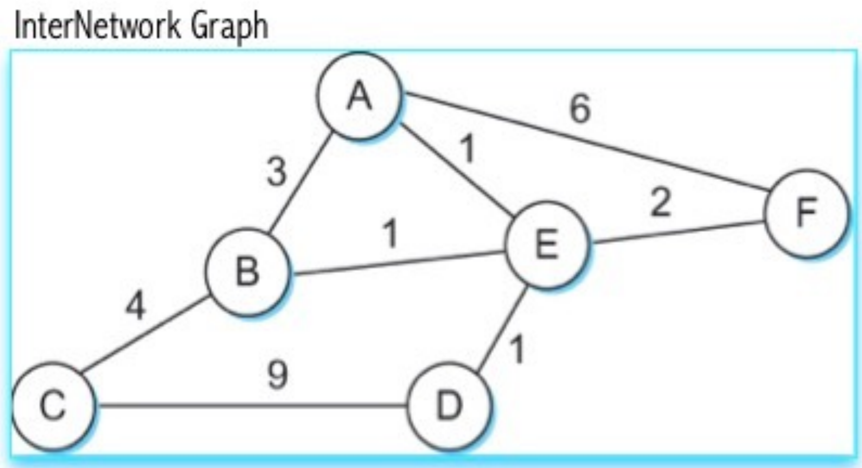
(b)

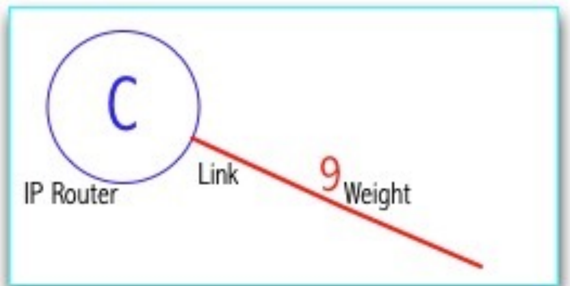
Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

Least cost Routing

- Network (*Internetwork*) as a Graph

- The basic problem of routing is to find the lowest-cost path between any two nodes
  - Where the cost of a path equals the sum of the costs of all the edges that make up the path





Based on textbook Conceptual Computer Networks © 2013-2018 by José María Foces Morán & José María Foces Vivancos

Routing

- For a simple network, we can calculate all shortest paths and load them into some nonvolatile storage on each node.

- Such a **static** approach has several shortcomings
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change

- What is the **solution**?
  - Need a distributed and dynamic protocol
  - Two main classes of protocols
    - Distance Vector
    - Link State

Link State Routing

- **Strategy**: Each node sends the costs of its directly connected links to all the nodes
  - Complementary to DV

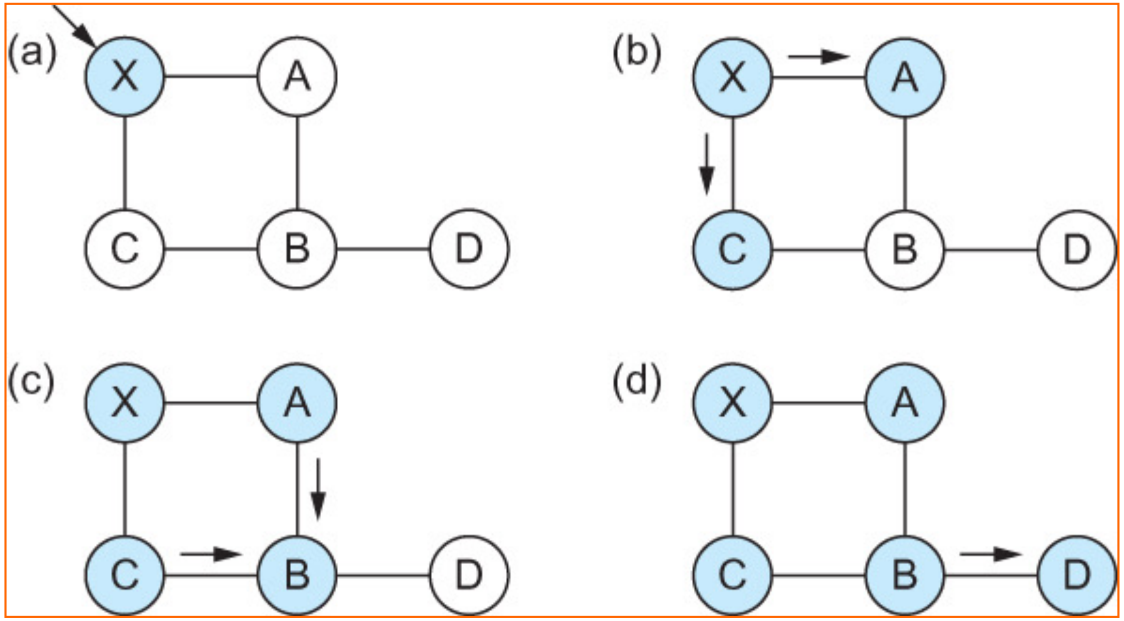
- **Link State Packet (LSP)**
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet

- **Reliable Flooding**
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - Start SEQNO at 0; generate new LSP periodically; SEQNO++
  - TTL-- of each stored LSP; discard when TTL=0
  - From hop-to-hop, reliability is provided by acknowledgements and retransmissions

Reliable Flooding

LSP = Link State Packet

a. LSP arrives at node X
b. X floods LSP to A and C
c. A and C flood LSP to B (not X)
d. Flooding is complete



Shortest Path Routing

- Dijkstra's Algorithm - Assume non-negative link weights
  - N: set of **nodes** in the graph
  - l((i, j): the non-negative **cost** associated with the edge between nodes i, j ∈ N and l(i, j) = ∝ if no edge connects i and j
  - Let s ∈ N be the **starting node (root)** which executes the algorithm to find shortest paths to all other nodes in N
  - Two variables used by the algorithm
    - M: set of nodes incorporated so far by the algorithm = {P}
    - C(n) : the cost of the path from s to each node n = {T}

    - The algorithm:

```
M = {s}
For each n in N - {s}
    C(n) = l(s, n)
while (N ≠ M)
  M = M ∪ {w} such that C(w) is the minimum
                        for all w in (N-M)
  For each n in (N-M)
          C(n) = MIN (C(n), C(w) + l(w, n))
```

Shortest Path Routing

- In practice, each *router* computes its routing table directly from the LSP's it has collected using a realization of Dijkstra's algorithm called the forward search algorithm

- Specifically each switch maintains two lists of nodes, known as **Temporary** and **Permanent**
  - Permanent {P} nodes that do belong to the shortest path from the root
  - Temporary {T} nodes: those that have not been added to the shortest path from the root, yet

- Each of these lists contains a set of entries of the form:

Next-hop(Current node, partial cost, total cost)

Based on textbook Conceptual Computer Networks © 2013-2018 by José María Foces Morán & José María Foces Vivancos

L(K, 3, 17)

L can be reached from K at a cost of 3, the total least-cost path, so far is 17 hops

- This is the notation that we are going to use in this course (CN/ADG)
- **Beware**: it is not the same one used in the textbook!

Shortest Path Routing

- The algorithm

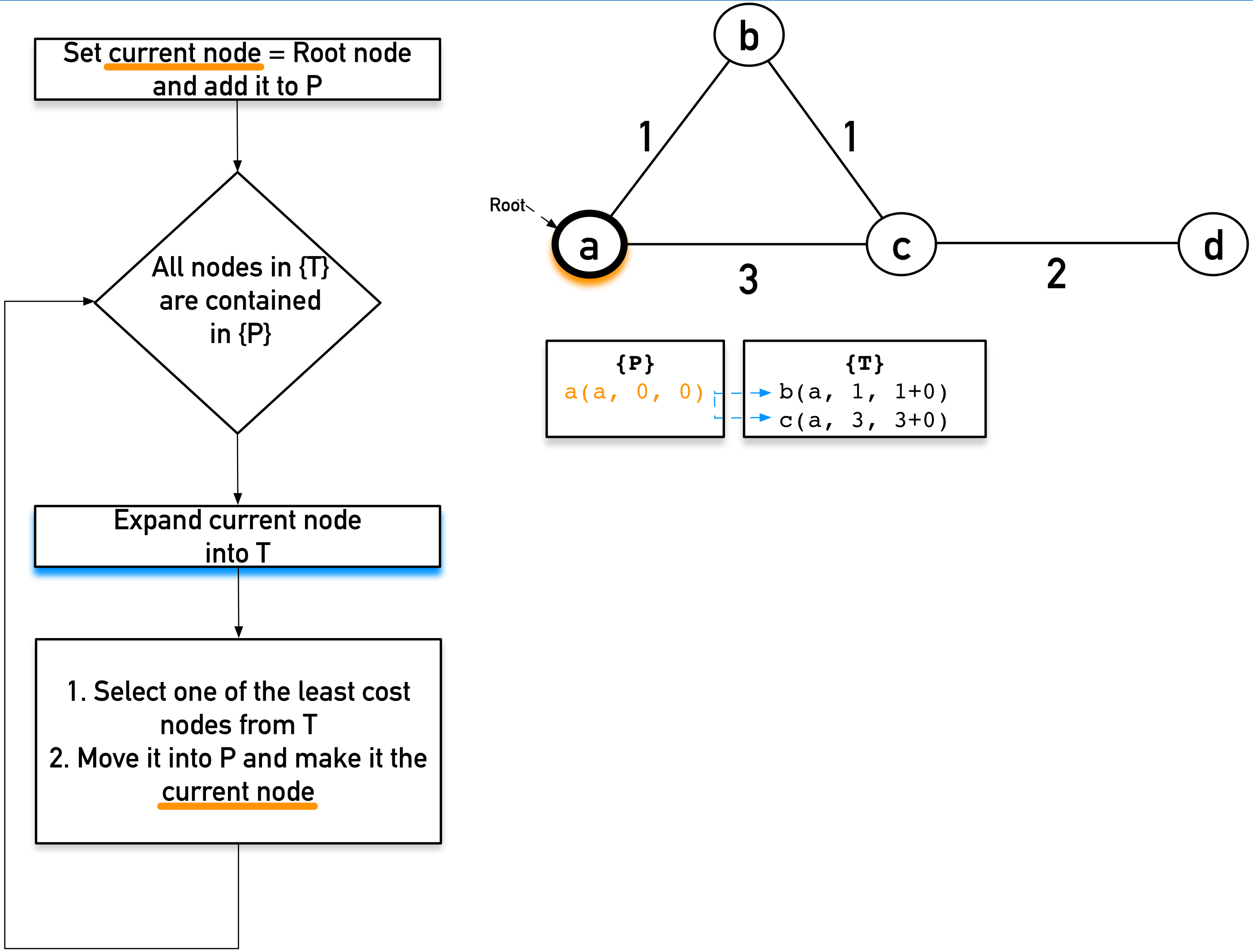
  - Initialize the **Permanent** list with an entry for myself; this entry has a cost of 0
    - The root of resulting shortest-path tree

  - For the node just added to the **Permanent** list in the previous step, call it node **Next**, select its LSP

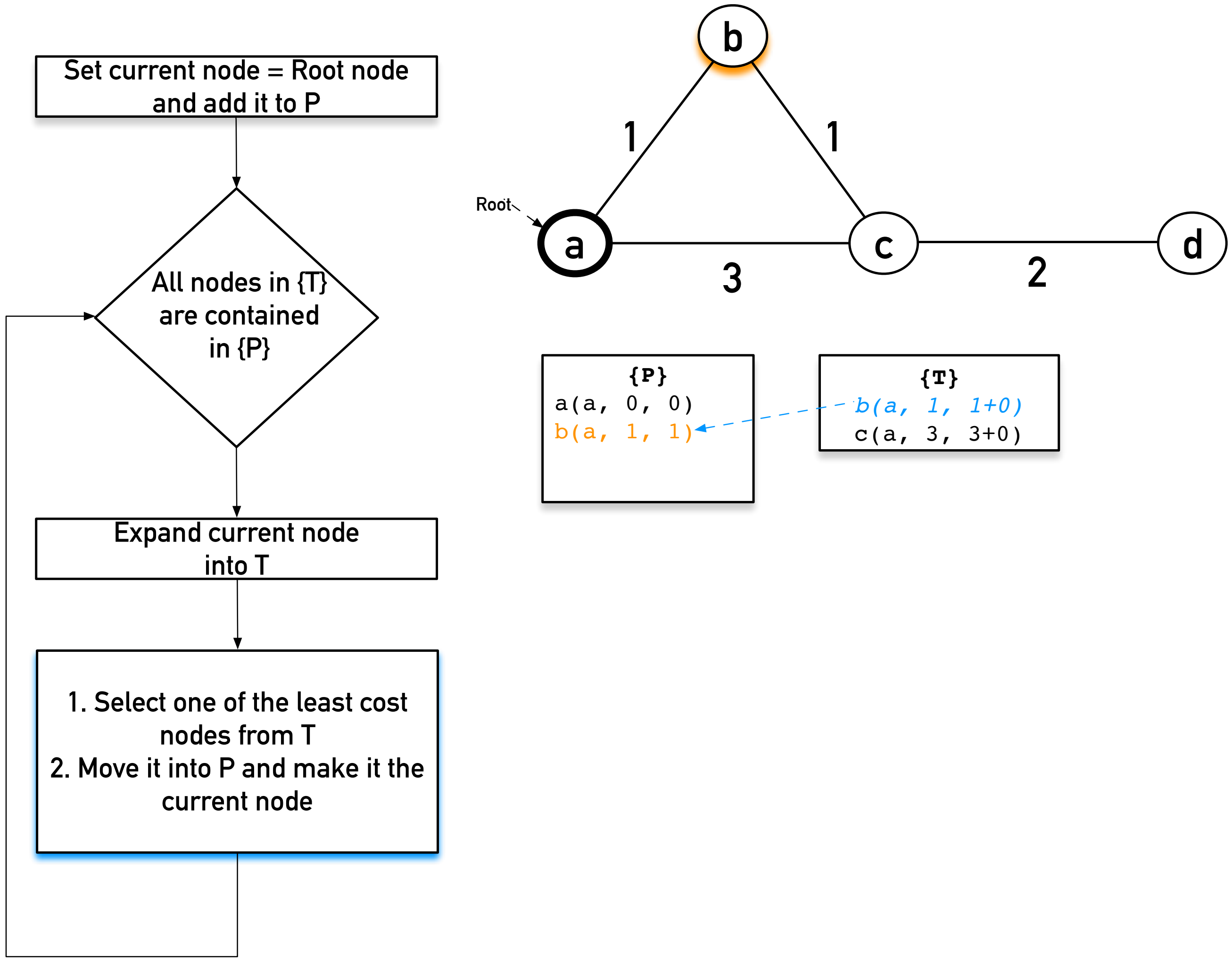
  - For each neighbor (Neighbor) of **Next**, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor
    - If Neighbor is currently on neither the **Permanent** nor the **Temporary** list, then add (Neighbor, Cost, Nexthop) to the **Tentative** list, where Nexthop is the direction I go to reach Next
    - If Neighbor is currently on the **Temporary** list, and the Cost is less than the currently listed cost for the Neighbor, then replace the current entry with (Neighbor, Cost, Nexthop) where Nexthop is the direction I go to reach Next

  - If the **Temporary** list is empty, stop. Otherwise, pick the entry from the **Temporary** list with the lowest cost, move it to the **Permanent** list, and return to Step 2.

Dijkstra's Algorithm (Fwd Search)



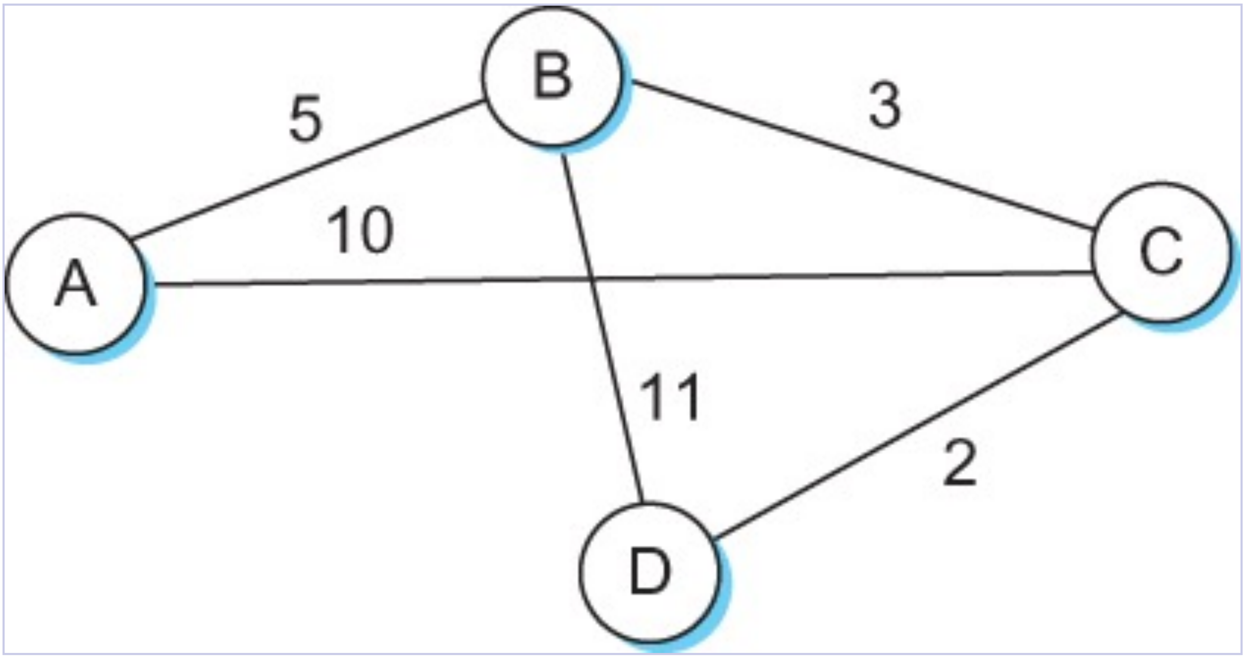
Dijkstra's Algorithm (Fwd Search)



Shortest Path Routing

Example about Link-State routing with the **Forward Search Algorithm (Dijkstra)**

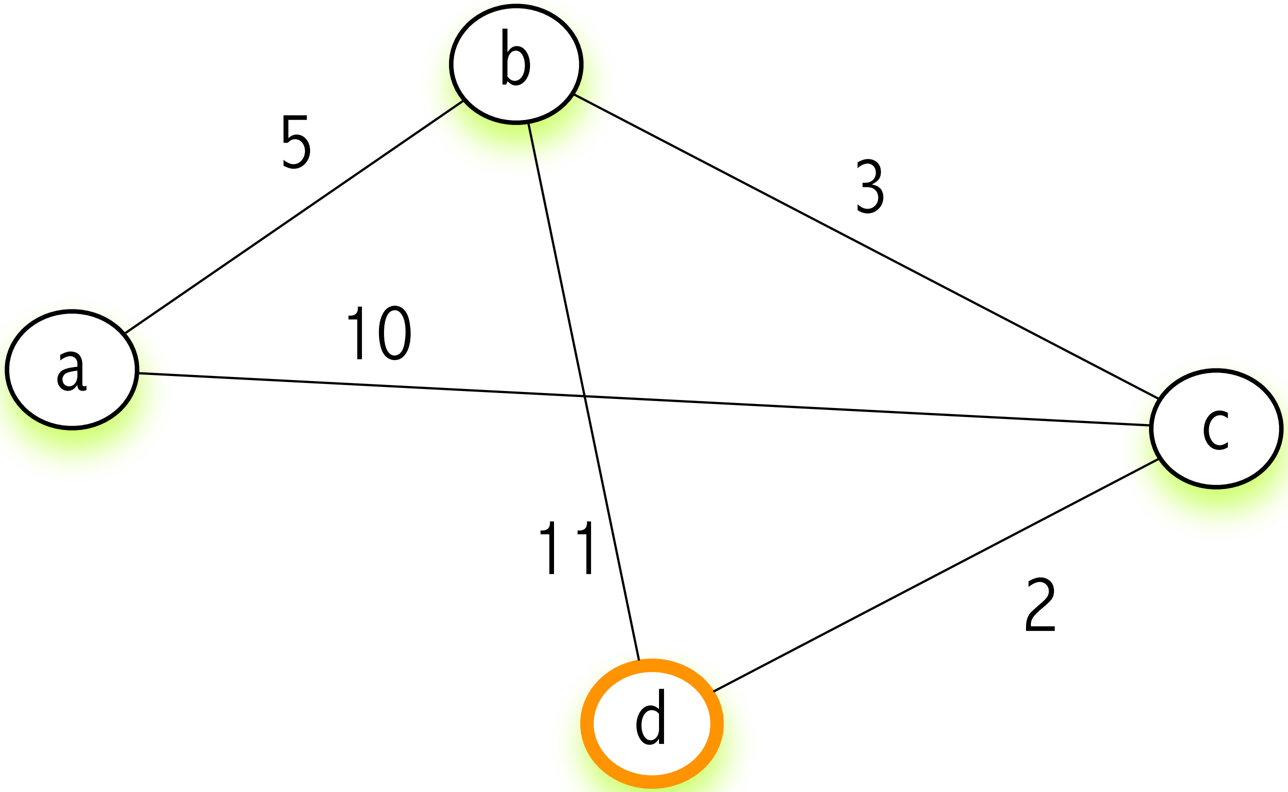
· Calculate the Shortest Path Tree of **node D**



Shortest Path Routing

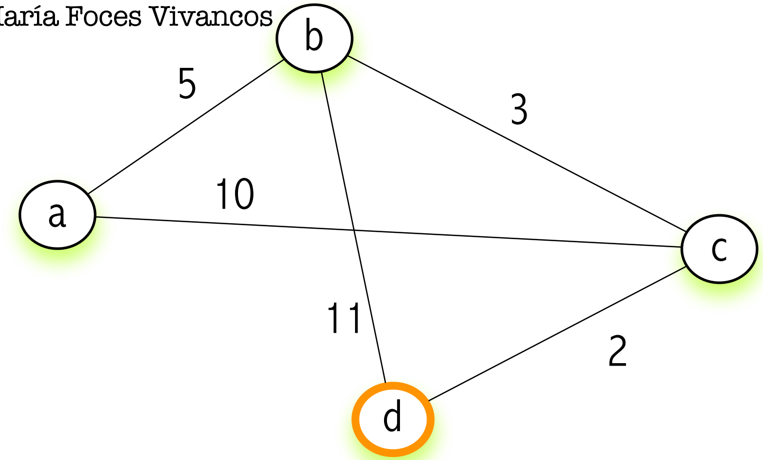
- In this example, node d is the root node
  - Recall *our* notation:

    Next-hop(Current node, partial cost, total cost)



Forward Search Algorithm

- This algorithm keeps two lists of edges
  - The permanent list {P}
  - The Temporary list {T}



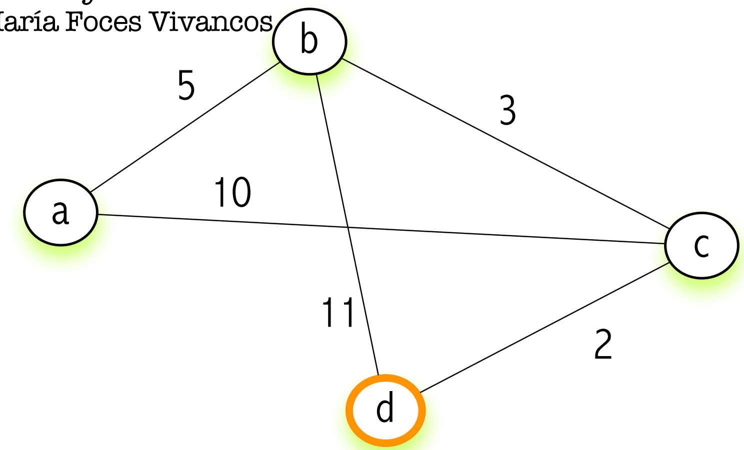
- The algorithm adds edges to {T}
- From {T} it selects the shortest edge and adds it to {P} which became the current edge
- The **current** edge at {P} is **expanded**, which adds more edges to {T}
- Finishes when all the nodes have an edge in {P}, which must be a tree: The Shortest Path Tree

Shortest Path Routing

Notation: Next-hop(Current node, partial cost, total cost)

- Since d is the root node, add it directly to {P}
  - Now, d belongs to the Least Cost Path tree, mark it on the graph

{P}	{T}
d(d, 0, 0)	

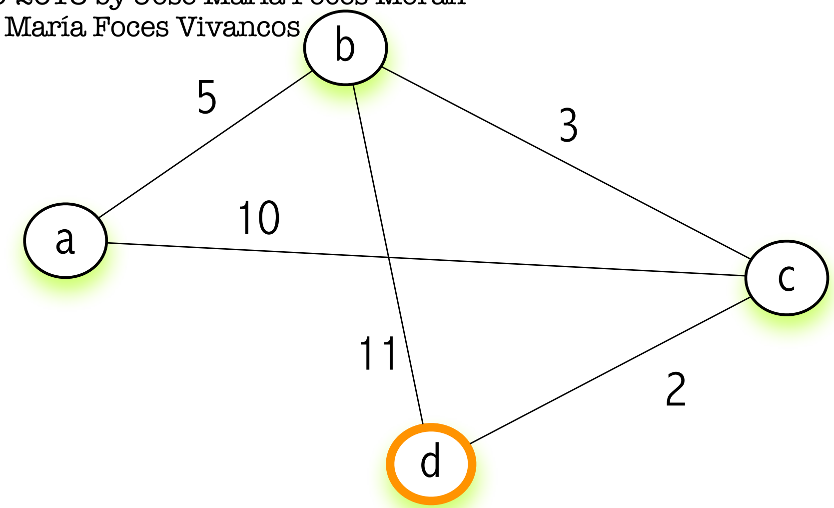


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, <u>total cost</u>)

- Expand the current node at {P} d(d, 0, 0)
  - The <u>forward neighbors</u> of d(d, 0, 0) are b(d, 11, 11+ 0) and c(d, 2, 2 + 0)
  - Add them to {T}

{P}	{T}
d(d, 0, 0)	b(d, 11, 11+0) c(d, 2, 2+0)

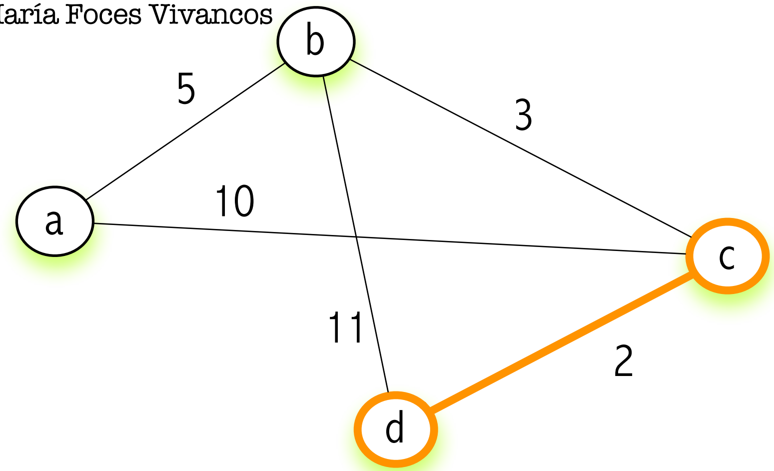


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, total cost)

- Select the least cost edge from {T}, in this case it's c(d, 2, 2)
- Add that least cost edge c(d, 2, 2) to {P}

{P}	{T}
d(d, 0, 0)	
c(d, 2, 2)	b(d, 11, 11+0) ~~c(d, 2, 2+0)~~

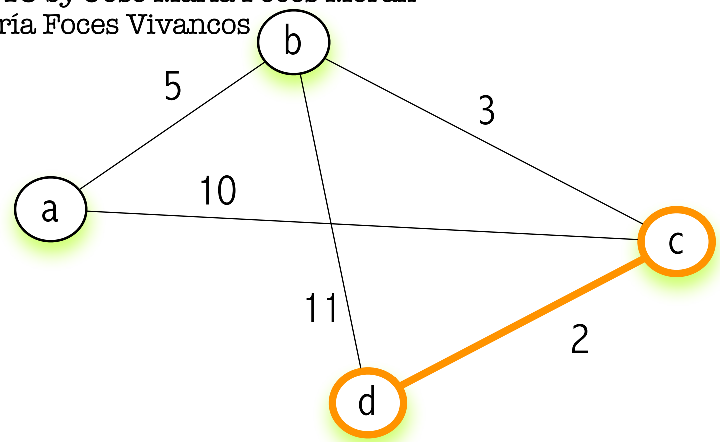


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, <u>total cost</u>)

- Expand the current edge at {P} c(d, 2, 2)
  - The expansion of c(d, 2, 2) is comprised of neighbors a and b since in this case c is reached from c, this is the <u>forward expansion</u> of c which must not contain d since it is on the backwards path
  - Add edges b(c, 3, 3+2) and a(c, 10, 10+2) to {T}

{P}	{T}
d(d, 0, 0)	~~b(d, 11, 11+0)~~ ~~c(d, 2, 2+0)~~
c(d, 2, 2)	b(c, 3, 5) a(c, 10, 12)

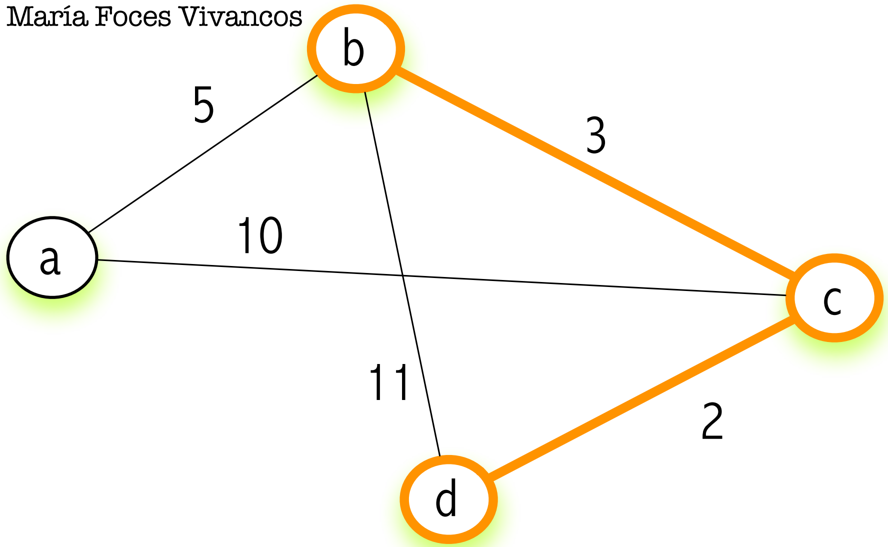


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, total cost)

- Expand b(c, 3, 5):
  - d(b, 11, 11 + 5)
  - a(b, 5, 5 + 5)

{P}	{T}
d(d, 0, 0)	~~b(d, 11, 11+0)~~ ~~c(d, 2, 2+0)~~
c(d, 2, 2)	~~b(c, 3, 5)~~ a(c, 10, 12) a(b, 5, 5) d(b, 11, 11 + 5)
b(c, 3, 5)	

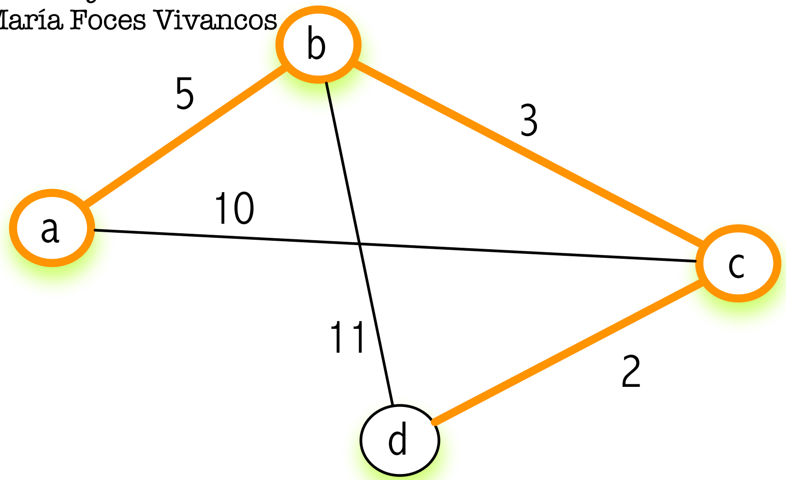


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, total cost)

- Discard d(b, 11, 11 + 5) since we have a shorter d(d, 0, 0)
- Discard a(c, 10, 12) a since a(b, 5, 5 + 5) is shorter
- Select the shortest edge a(b, 5, 5) and move it into {P}, then discard it from {T}

{P}	{T}
d(d, 0, 0)	~~b(d, 11, 11+0)~~ ~~c(d, 2, 2+0)~~
c(d, 2, 2)	~~b(c, 3, 5)~~ ~~a(c, 10, 12)~~ ~~d(b, 11, 11 + 5)~~ ~~a(b, 5, 5+5)~~
b(c, 3, 5)	
a(b, 5, 10)	

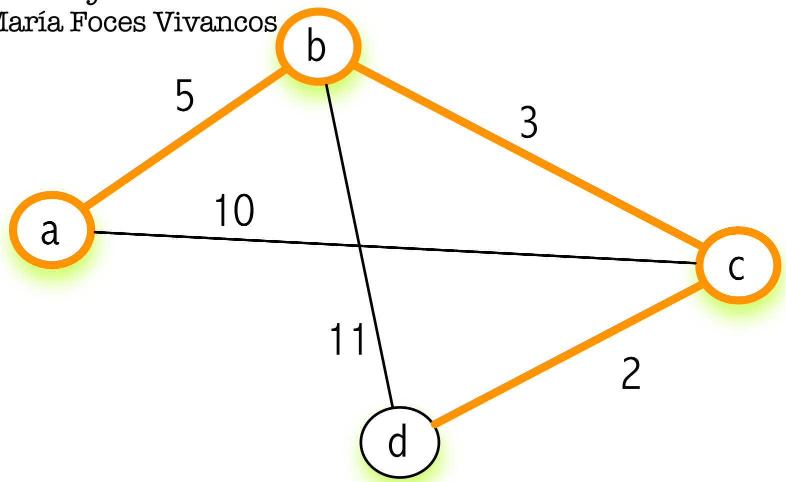


Shortest Path Routing

Notation: Next-hop(Current node, partial cost, total cost)

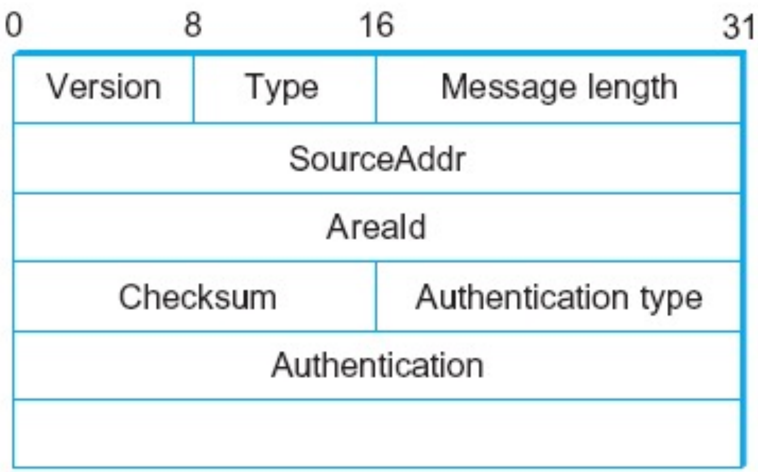
- Expand a(b, 5, 10) which yields c(a, 10, 10 + 5)
  - Discard c(a, 10 , 15) since another, shorter edge to c exists in {P}: c(d, 2, 2)
- Now, {T} is empty: finish.

{P}	{T}
d(d, 0, 0)	~~b(d, 11, 11+0)~~ ~~c(d, 2, 2+0)~~
c(d, 2, 2)	~~b(c, 3, 5)~~ ~~a(c, 10, 12)~~ ~~d(b, 11, 11 + 5)~~ ~~a(b, 5, 5)~~
b(c, 3, 5)	~~c(a, 10, 10 + 5)~~
a(b, 5, 10)	

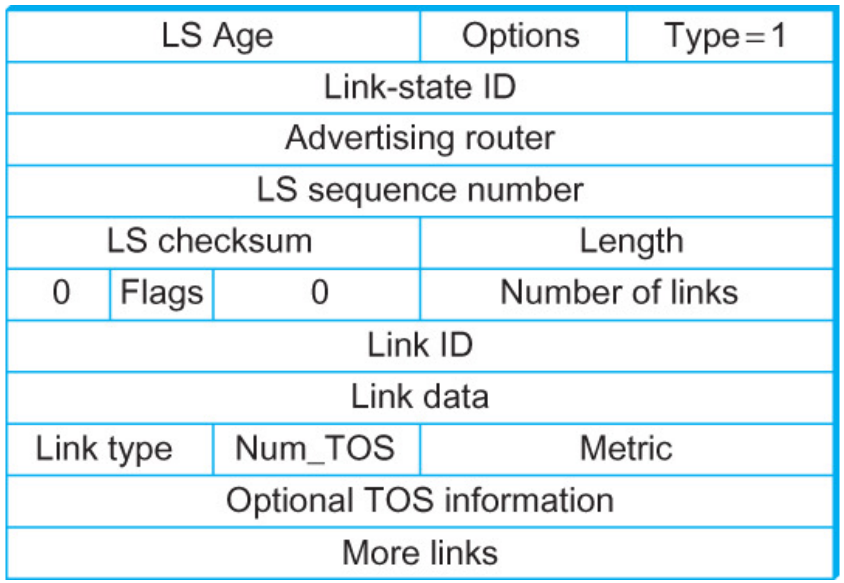


Open Shortest Path First (OSPF)

- OSPF is an interior (Intra-autonomous system), link-state routing protocol
  - Implements the Dijkstra Shortest Path algorithm (Forward Search Algorithm)
  - OSPF data units: header format and LSE



OSPF Header Format



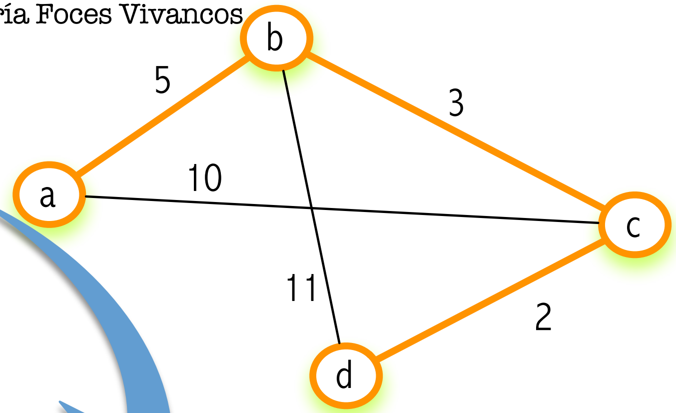
OSPF Link State Advertisement

Obtain the Forwarding Table of d

Exercises

1. Obtain the routing table of node d
2. Obtain the routing table of node a
3. Obtain the routing table of node b

Based on textbook Conceptual Computer Networks



Shortest path tree of node d

{P}	{T}
d(d, 0, 0)	~~b(d, 11, 11)~~ ~~c(d, 2, 2)~~
c(d, 2, 2)	~~a(c, 10, 12)~~ ~~b(c, 3, 5)~~
b(c, 3, 5)	~~a(b, 5, 10)~~
a(b, 5, 10)	{T} is empty: finish

Routing table of node d

Destination (Net number)	Next-hop	Cost
d	d	0
c	c	2
b	c	5
a	c	10

Recommended exercises

- Exams from past terms
- Textbook exercises (Computer Networks, P&D Ch. 3) 46, 48, 49, 62
- Review IP addressing and IP Forwarding Algorithm
- Review the examples and exercises included in this presentation

Summary

- We have looked at some of the issues involved in building scalable and heterogeneous networks by using switches and routers to interconnect links and networks.
- To deal with heterogeneous networks, we have discussed in details the service model of Internetworking Protocol (IP) which forms the basis of today's routers.
- We have discussed in details two major classes of *interior* routing algorithms
  - Distance Vector
  - Link State

The end