Introduction to Computer Networks

CS640

# Link State Routing

https://pages.cs.wisc.edu/~mgliu/CS640/F22/

**Ming Liu**
**mgliu@cs.wisc.edu**

# Today

## Last lecture

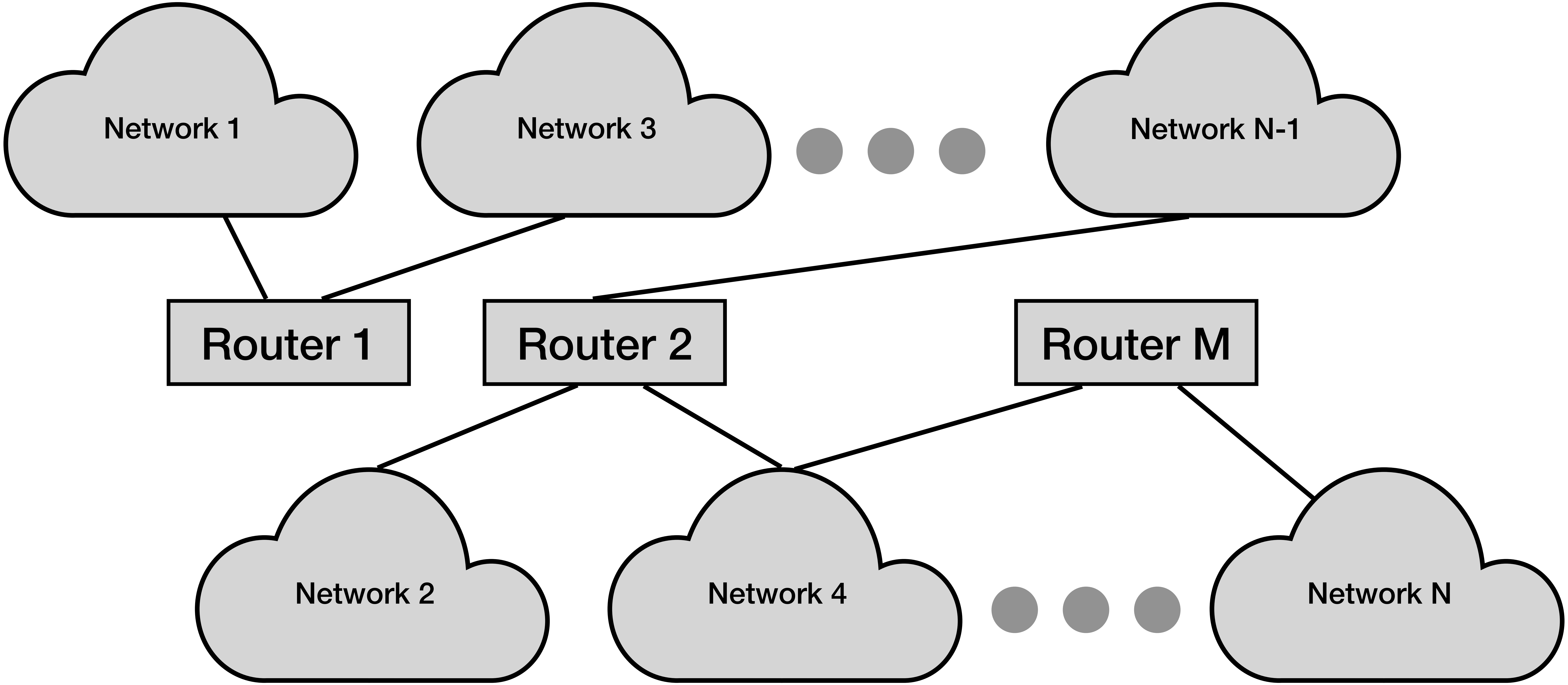
- How to decide the forwarding path among routers?

## Today

- How to decide the forwarding path among routers?

## Announcements

- Lab3 is due 11/04/2022, 11:59 PM


Network 1
Network 3
Network N-1
Router 1
Router 2
Router M
Network 2
Network 4
Network N

**Q: How to decide the forwarding path among routers?**

**OR**

**Q: How to build the routing table?**

## Q: How to decide the forwarding path among routers?

## OR

## Q: How to build the routing table?

## A: Routing Algorithm/Protocol.

- Represent connected networks as a graph
- Vertices in the graph are routers
- Edges in the graph are links
- Links have communication cost, which can be quantized!

# Techniques

## #1: Static configuration

## #2: Distance vector routing

## #3: Link state routing

# Link State Routing Overview

**Key idea: Send all nodes (not just neighbors) information about the communication cost of direct-connected links (not the entire routing table)**

**Find the shortest path between two nodes of the entire network**

- Each node has complete information about the network
- Known to converge quickly under static conditions

# Q: How does the link state routing work?

# Q: How does the link state routing work?

# A: Two steps:

- #1: Reliable flooding
- #2: Route calculation

## Q: How does the link state routing work?

## A: Two steps:

- #1: Reliable flooding
- #2: Route calculation

**Assumption: Each node can find out the state of the link to its neighbors and the cost of each link**

# Step 1: Reliable Flooding

**A node sends its link-state information out on all of its directly connected links; each node that receives this information then forwards it out on all its links**

# Step 1: Reliable Flooding

**A node sends its link-state information out on all of its directly connected links; each node that receives this information then forwards it out on all its links**

## Link state packet (LSP)

- The ID of the node that created the LSP
- The cost of the link to each directly connected neighbor
- The sequence number (SEQ#)
- The time-to-live (TTL) of this packet

# Step 1: Reliable Flooding

**A node sends its link-state information out on all of its directly connected links; each node that receives this information then forwards it out on all its links**

**Link state packet (LSP)**

LSP is generated when there is a topology change event or timeout event happening

# Q: Why do we need a sequence number?

**Q: Why do we need a sequence number?**

**A: Identify the latest link cost**

# Sequence Number

## Sender logic:

- Generate a new LSP periodically
- Start SEQ# at 0 when rebooted and increment SEQ# after each LSP

## Receiver logic:

- Upon receiving a copy of LSP (A)
  - Check if it has already received a copy (A') before
  - If A' == NULL, then accept
  - If A' != NULL
    - If A'.SEQ# > A.SEQ#, then accept; Otherwise, ignore
  - Forward A to all its neighbors except the neighbor from which the LSP was just received

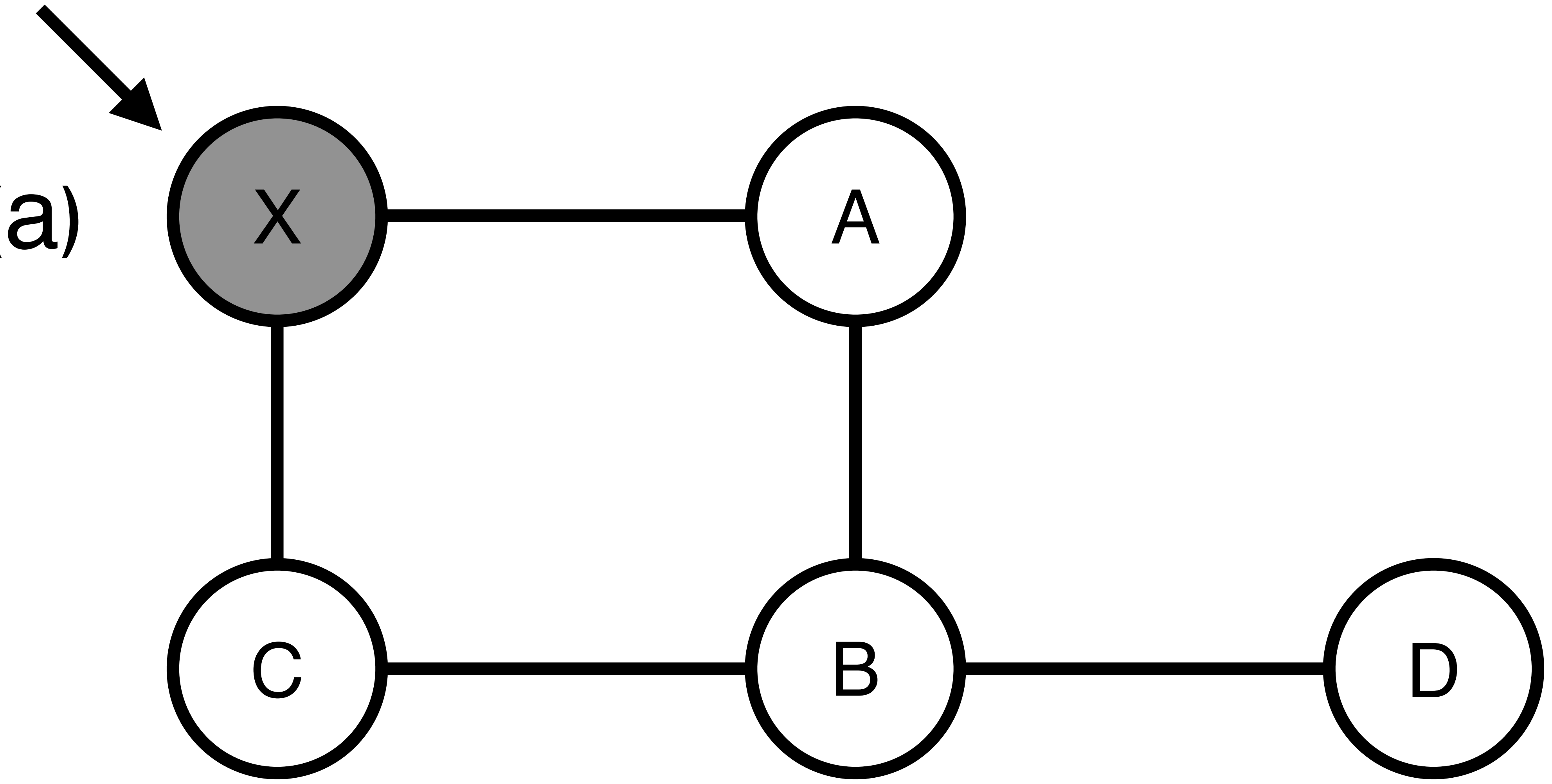
# Time-to-live (TTL)

**Decrement the TTL field when storing the LSP**

**Discard the LSP when its TTL = 0**

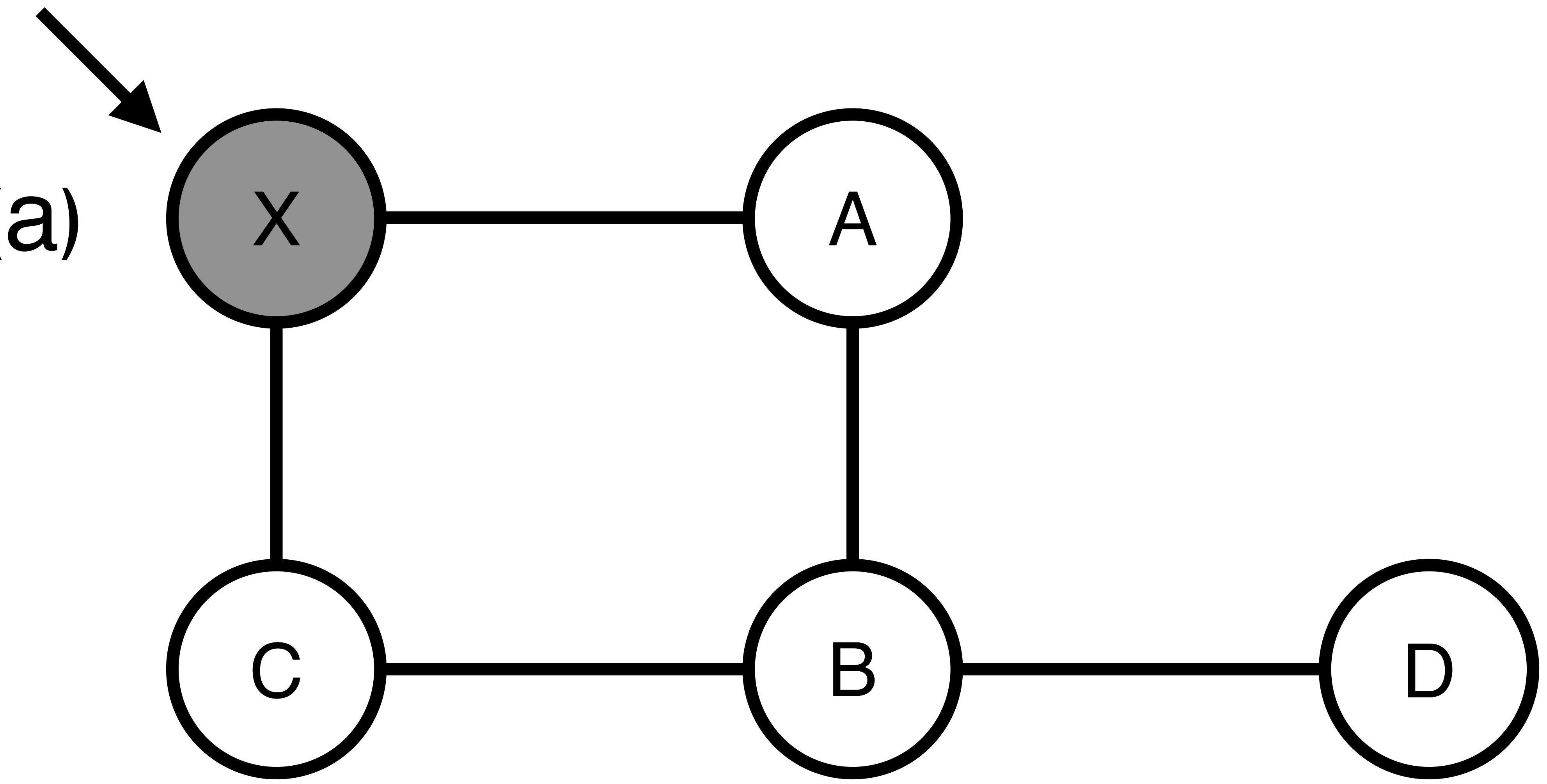
# A Flooding Example

(a)

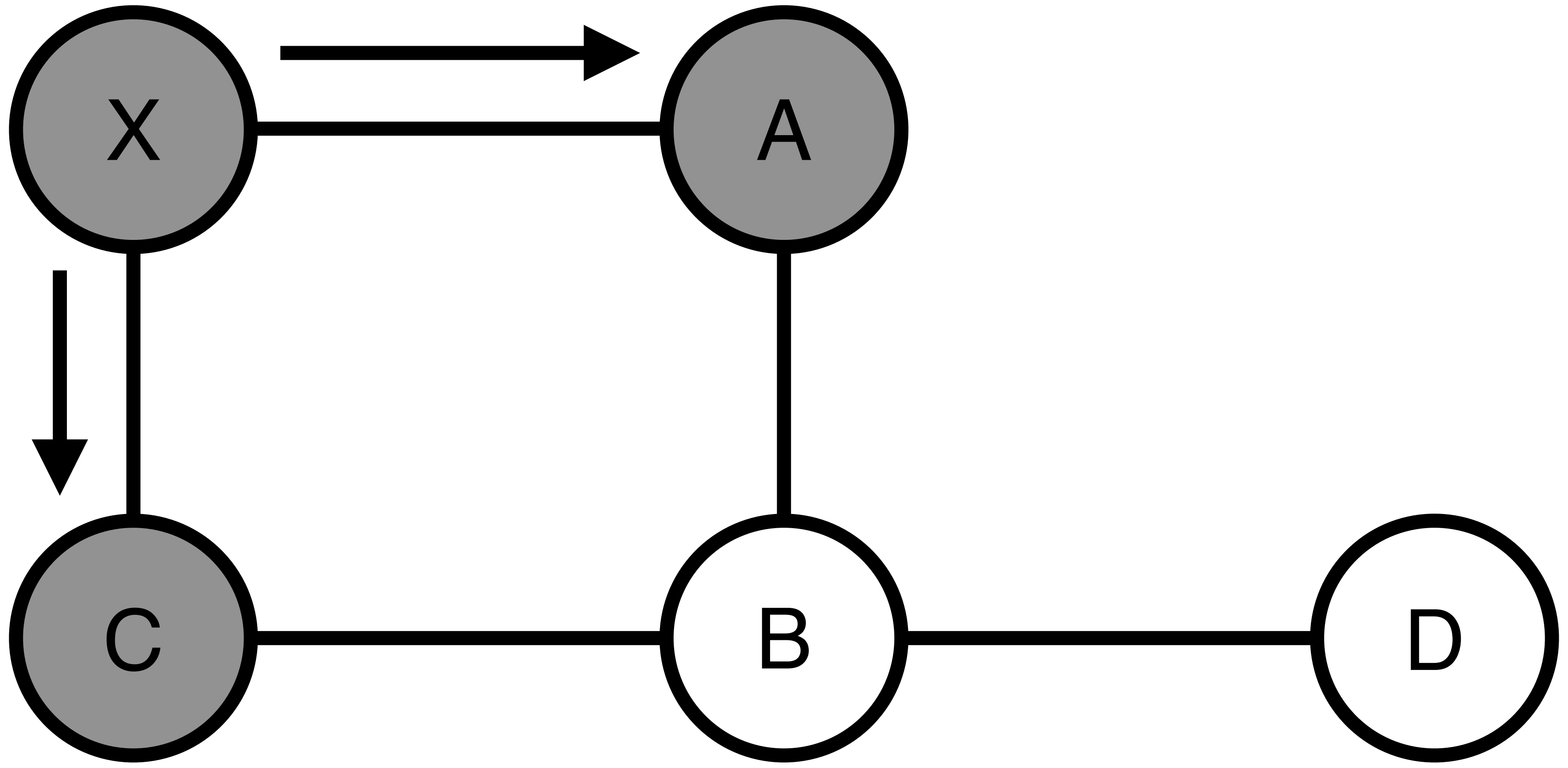


# A Flooding Example

(a)

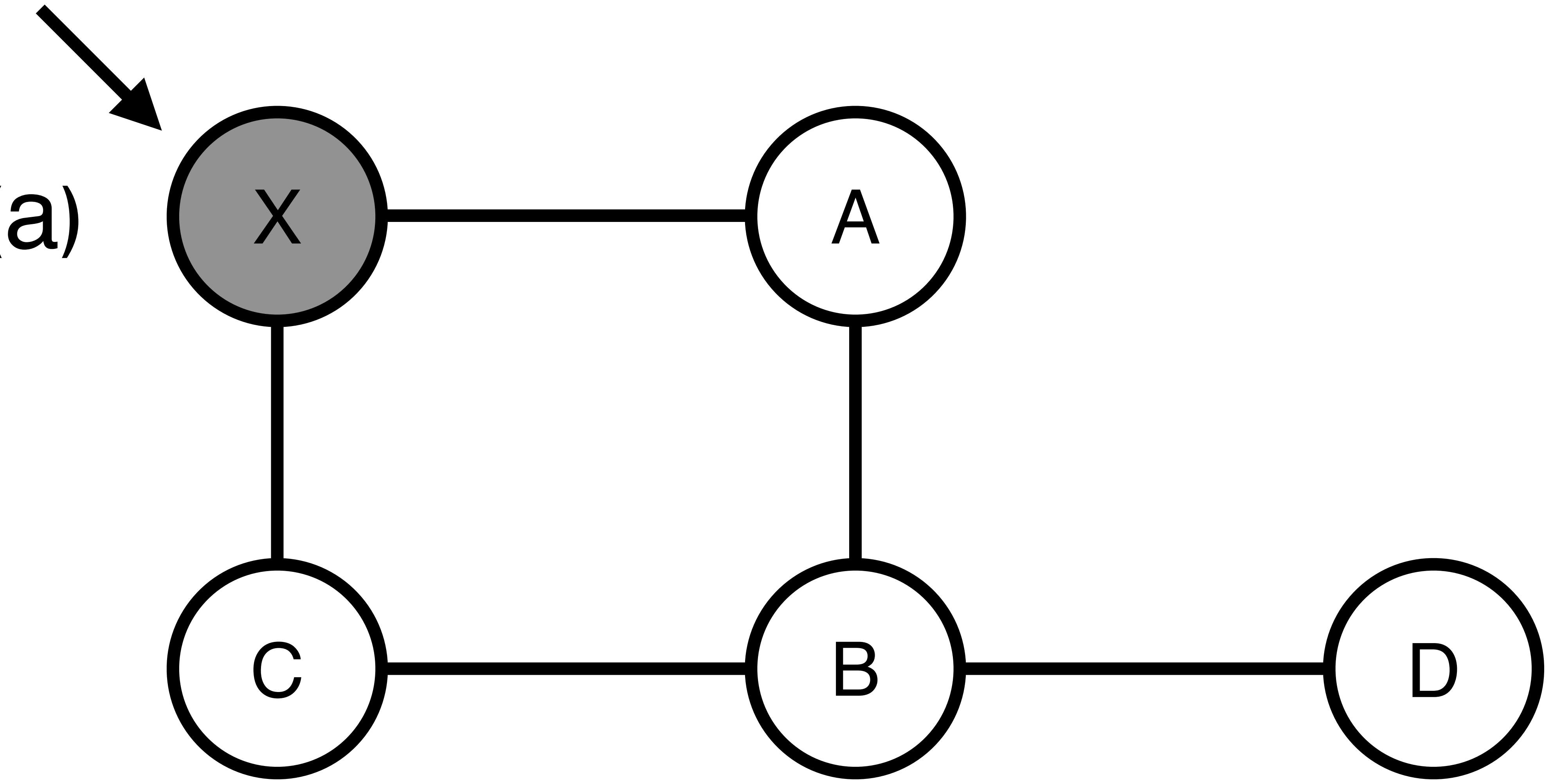



(b)

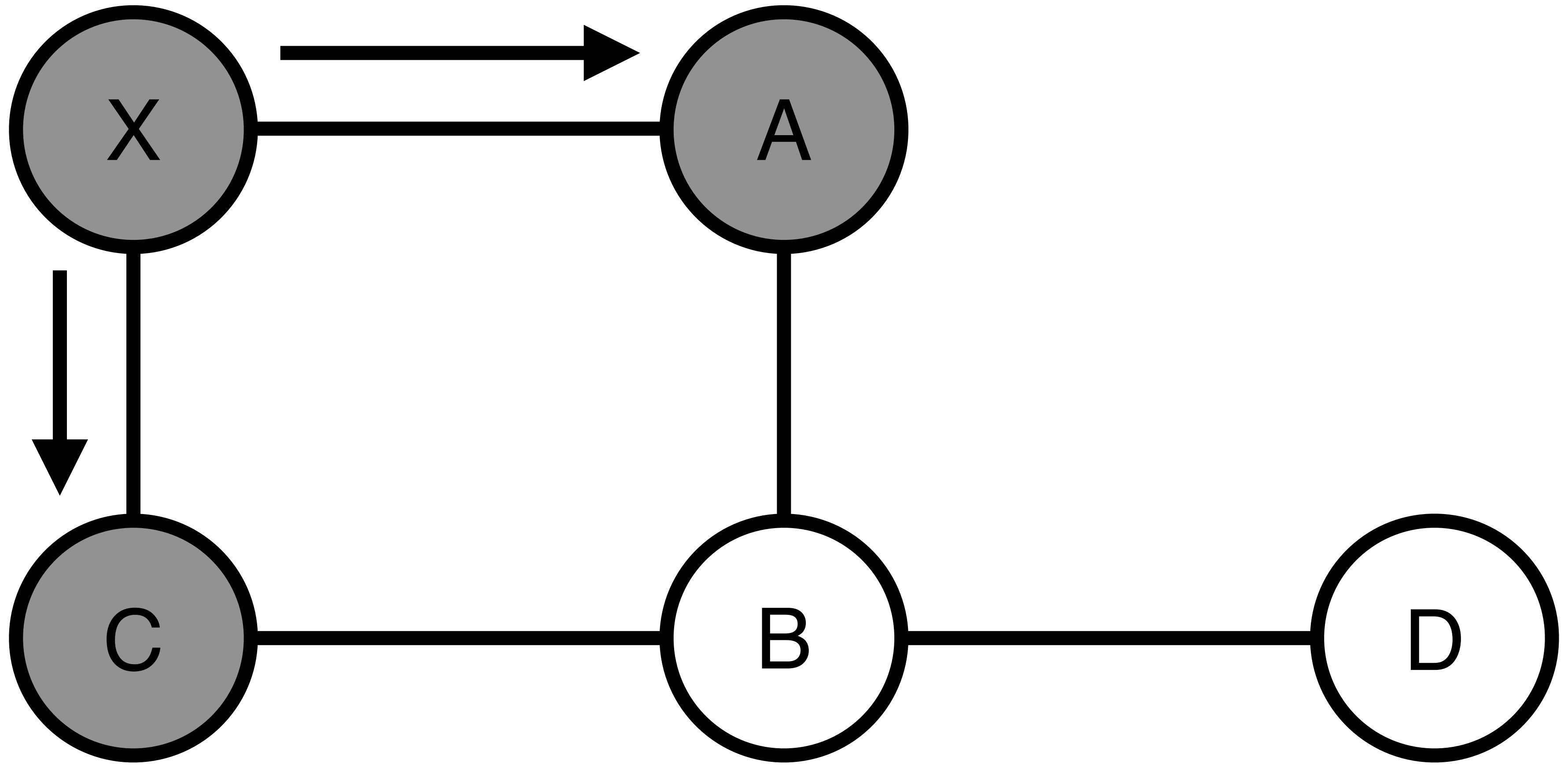


# A Flooding Example

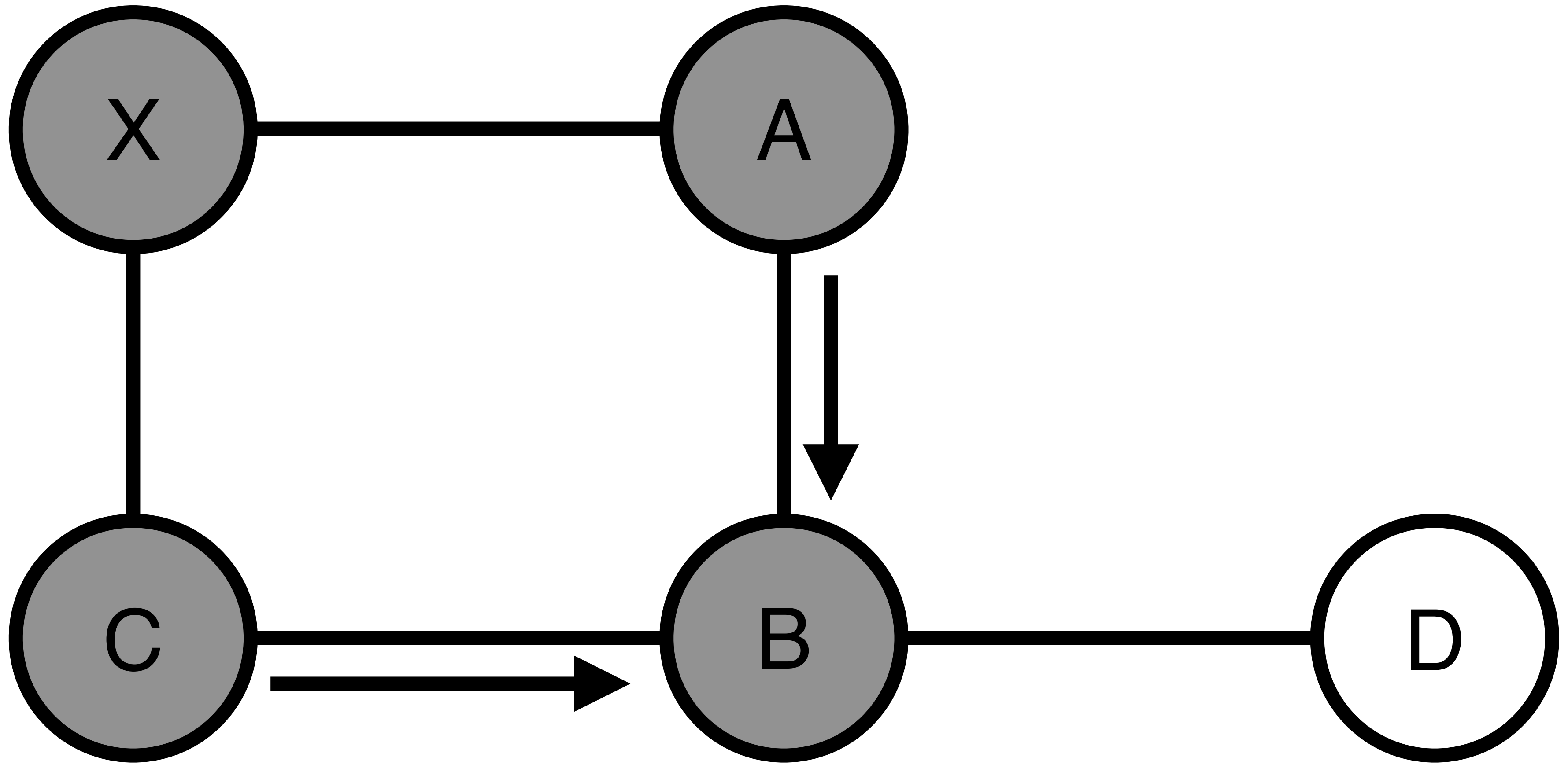
(a)



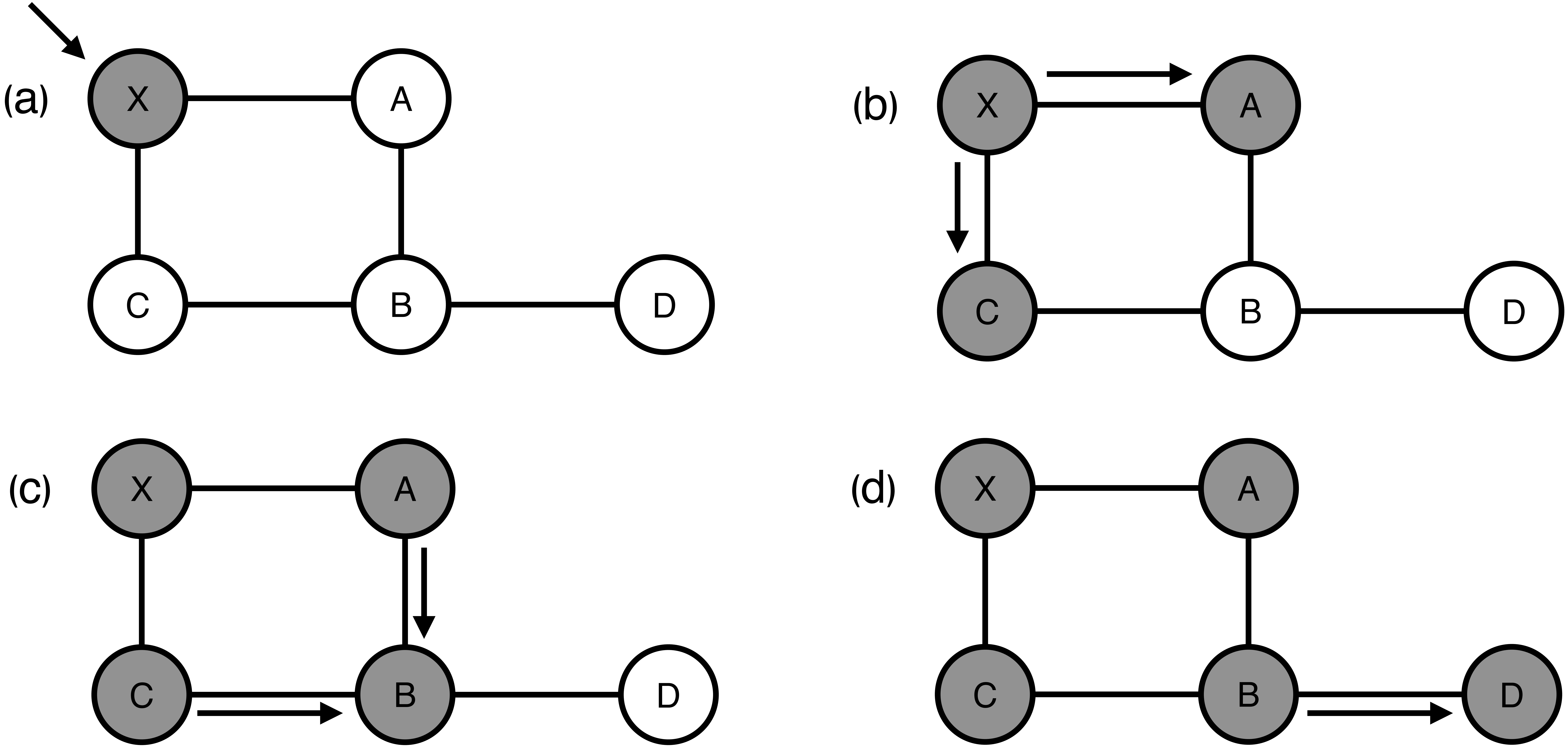

(b)




(c)



# A Flooding Example

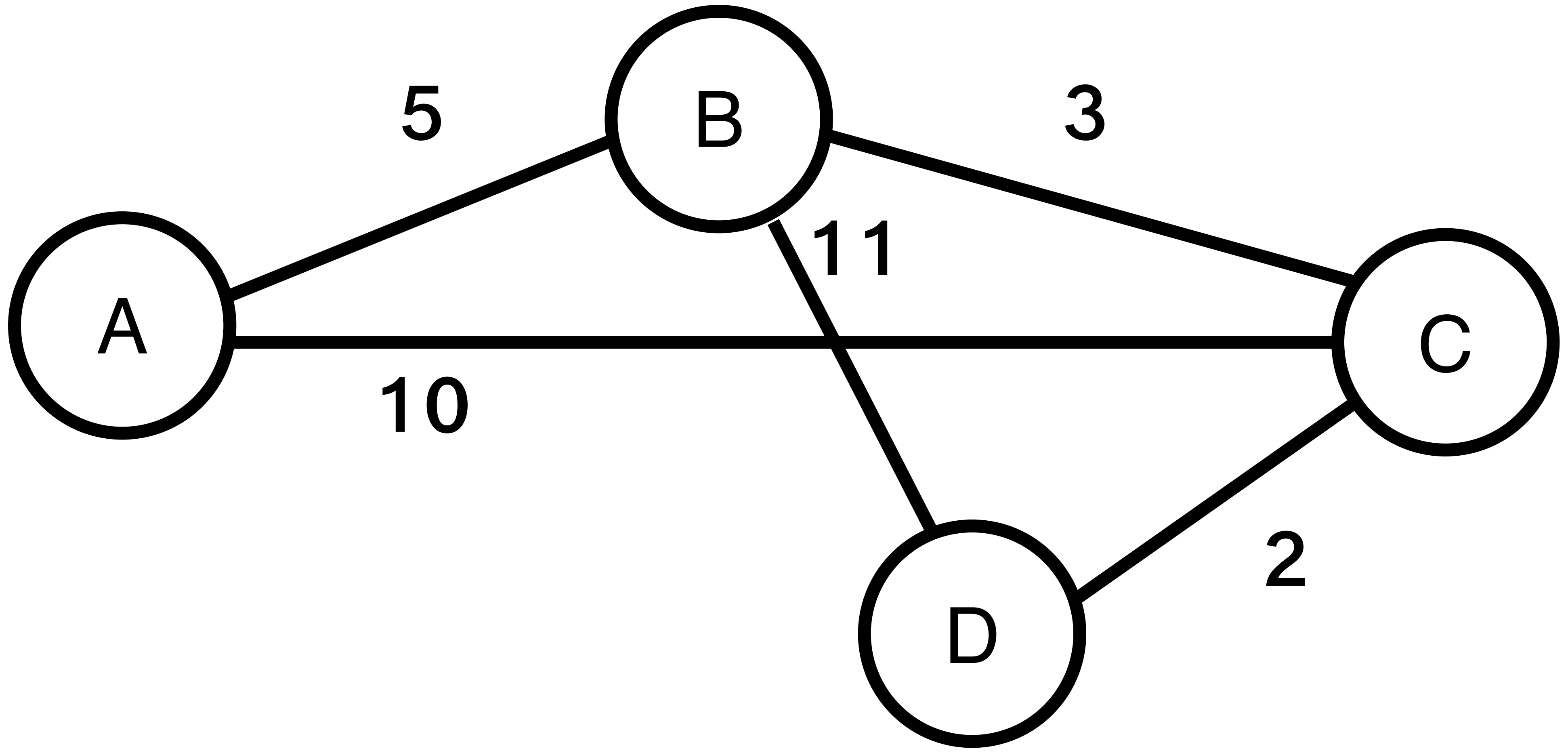


# Q: How does the link state routing work?

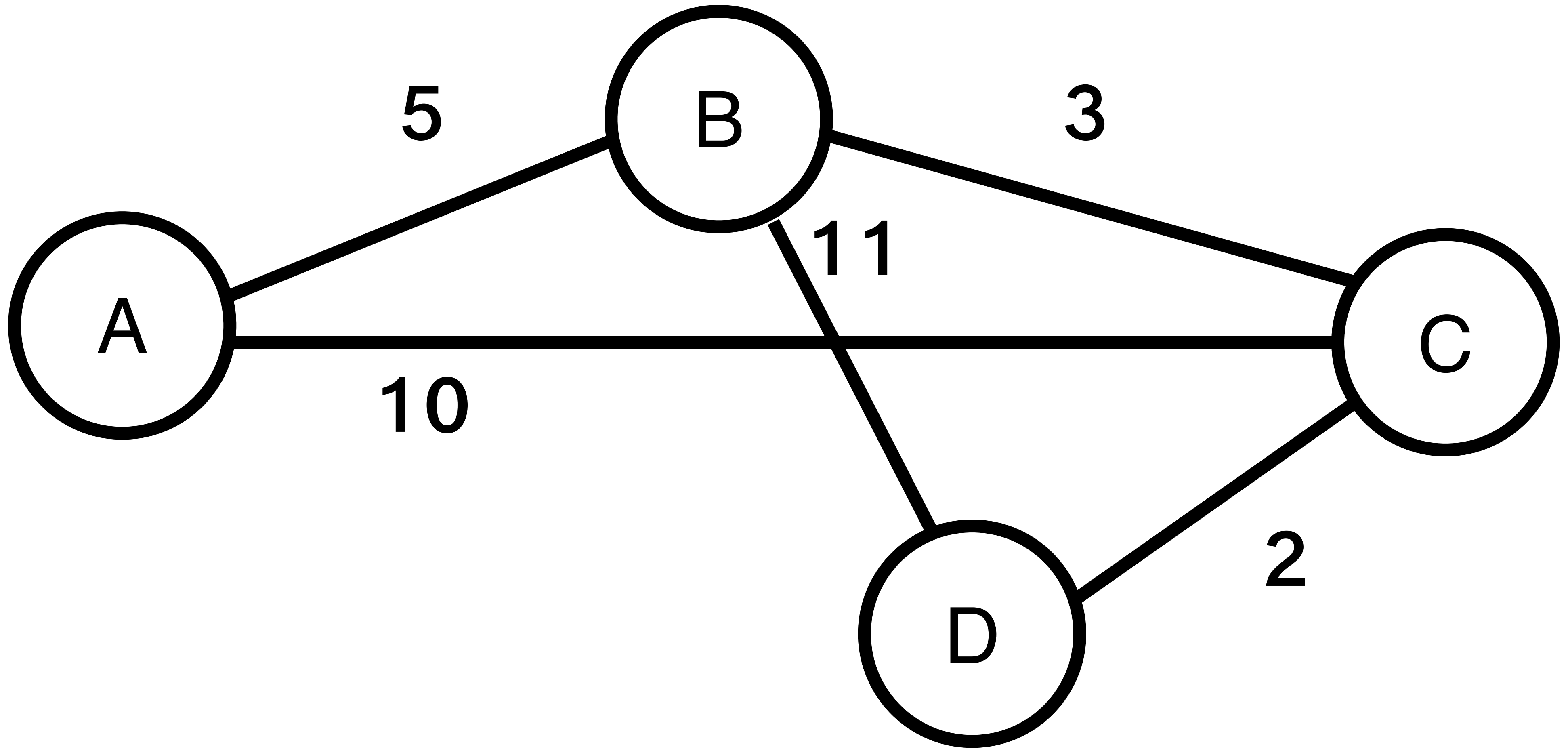
# A: Two steps:

- `#1:` Reliable flooding ✔
- `#2:` Route calculation

# Step 2: Route Calculation

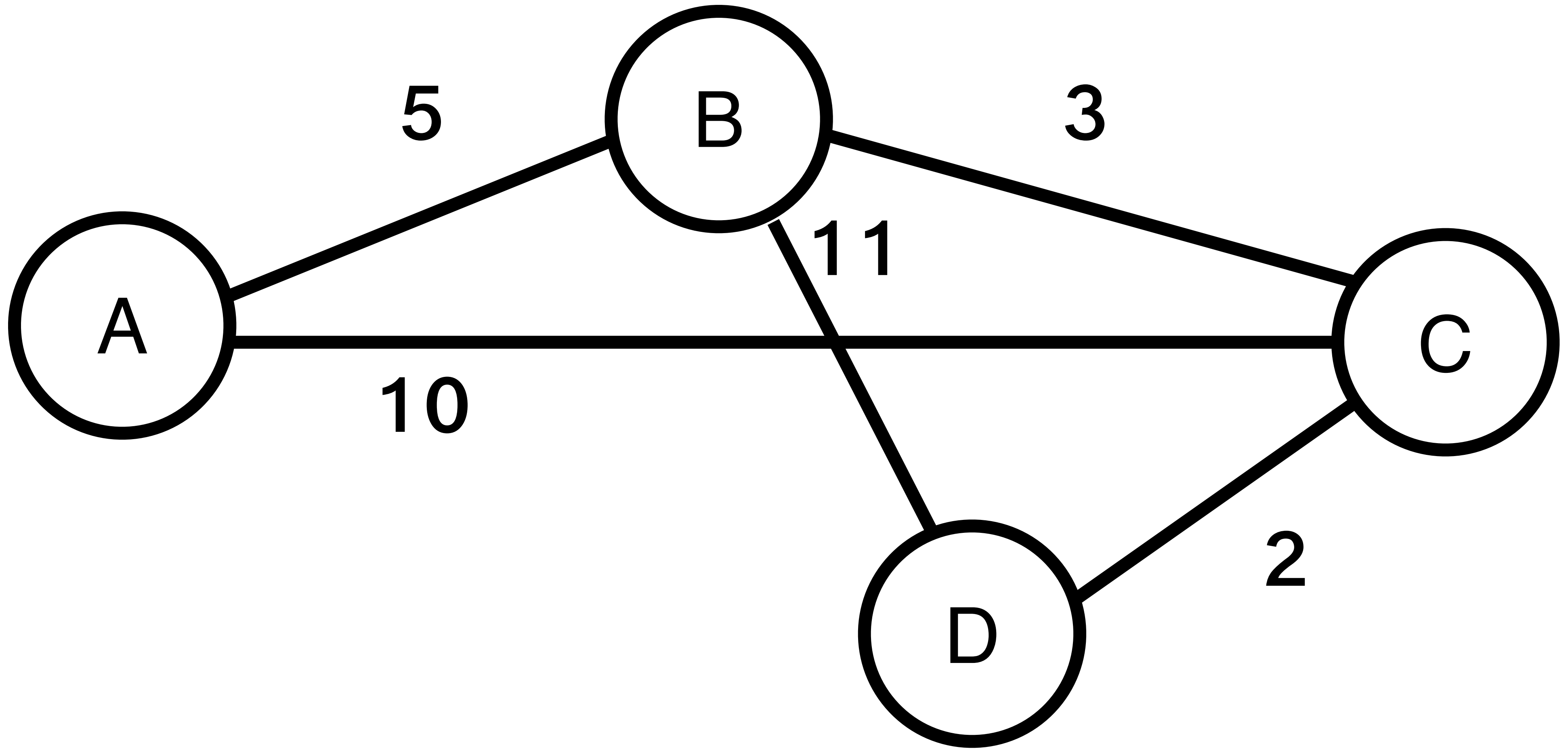


# Step 2: Route Calculation



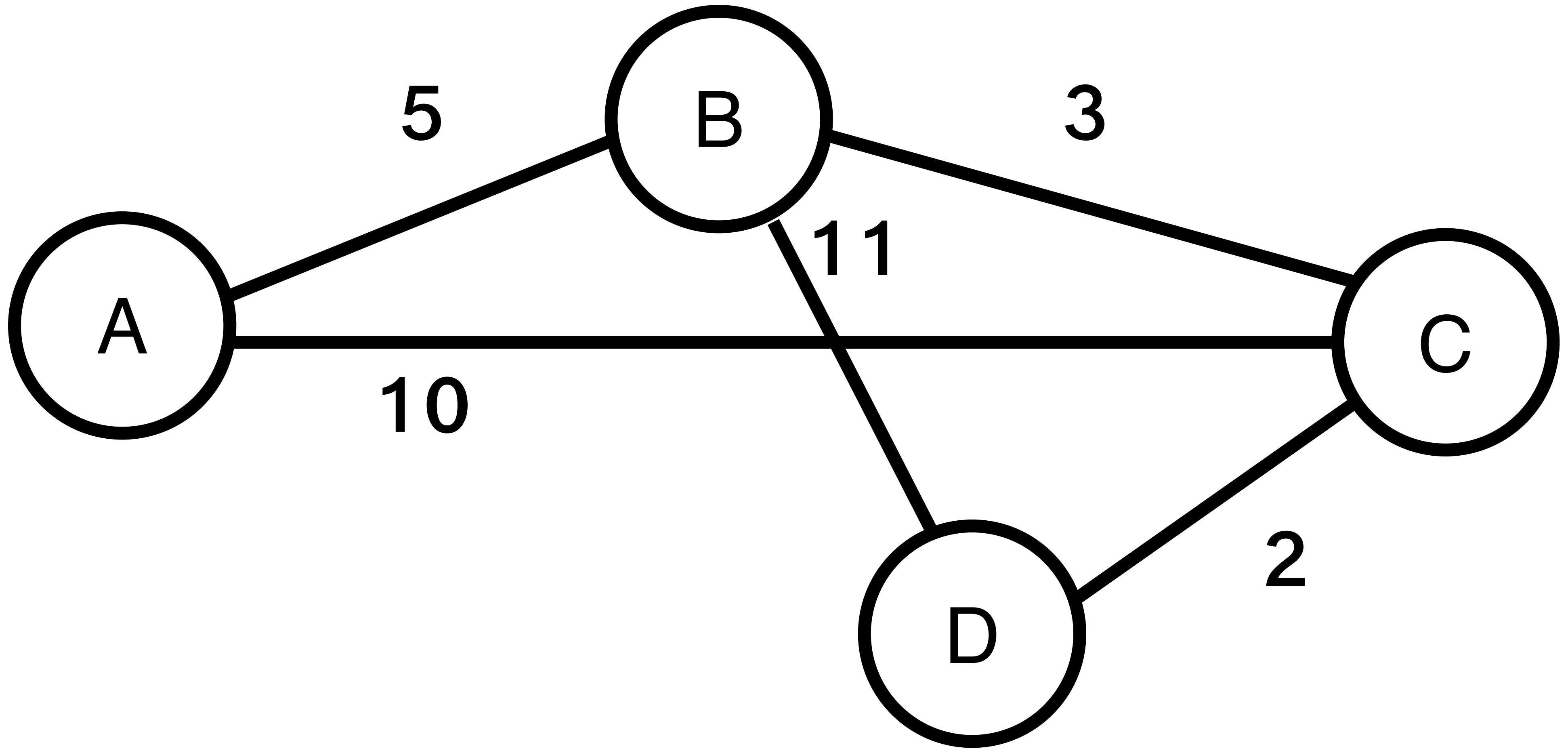

| Router A Info. | ID | Link Costs | SEQ# | TTL |
|---|---|---|---|---|
| **A LSP** | A | [A, B] = 5, [A, C] = 10 | 1 | 64 |
| **B LSP** | B | [B, A] = 5, [B, C] = 3, [B, D] = 11 | 1 | 63 |
| **C LSP** | C | [C, A] = 10, [C, B] = 3, [C, D] = 2 | 1 | 63 |
| **D LSP** | D | [D, B] = 11, [D, C] = 2 | 1 | 62 |

# Step 2: Route Calculation



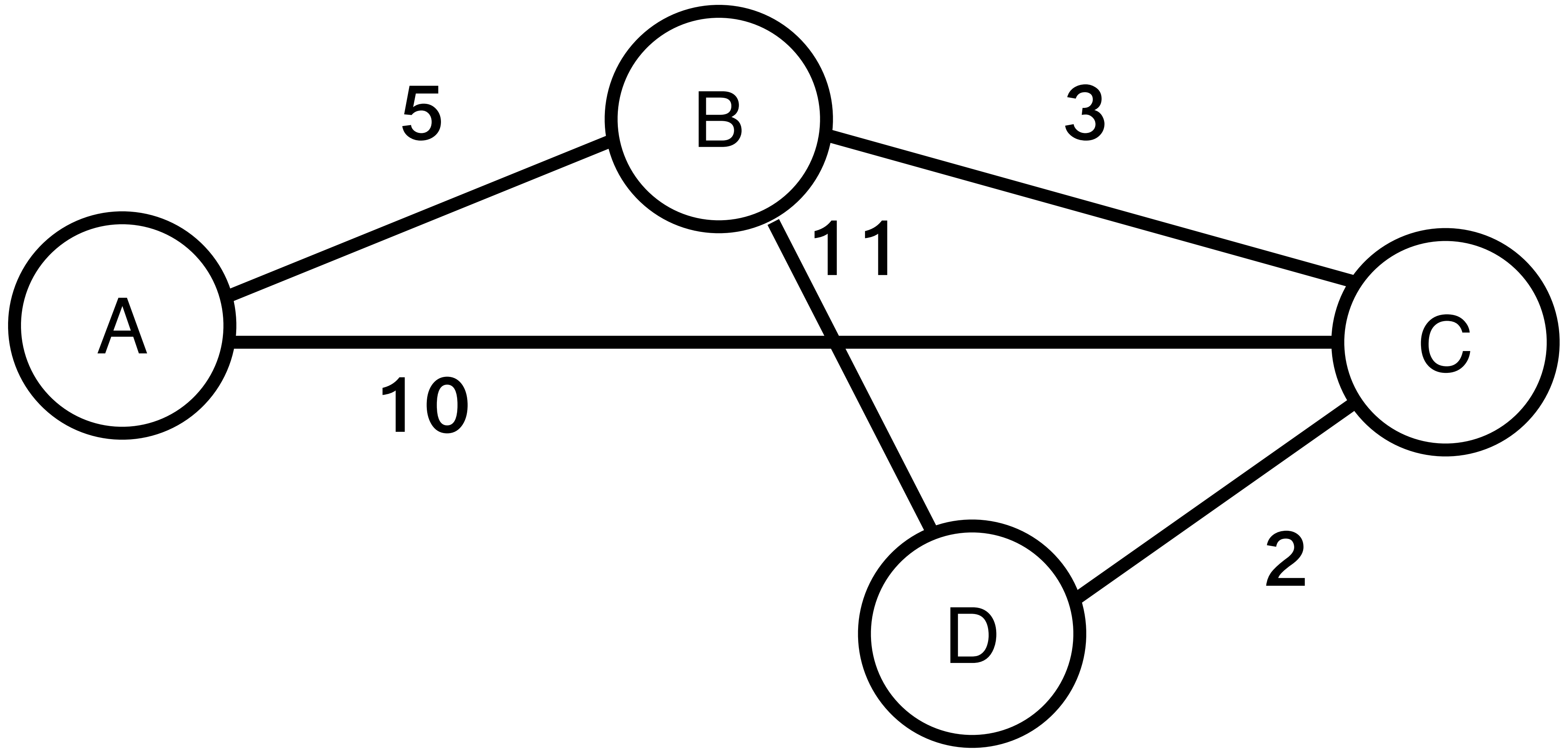

| Router B Info. | ID | Link Costs | SEQ# | TTL |
|---|---|---|---|---|
| A LSP | | | | |
| B LSP | | | | |
| C LSP | | | | |
| D LSP | | | | |

# Step 2: Route Calculation




| Router B Info. | ID | Link Costs | SEQ# | TTL |
|---|---|---|---|---|
| **A LSP** | A | [A, B] = 5, [A, C] = 10 | 1 | 63 |
| **B LSP** | B | [B, A] = 5, [B, C] = 3, [B, D] = 11 | 1 | 64 |
| **C LSP** | C | [C, A] = 10, [C, B] = 3, [C, D] = 2 | 1 | 63 |
| **D LSP** | D | [D, B] = 11, [D, C] = 2 | 1 | 63 |

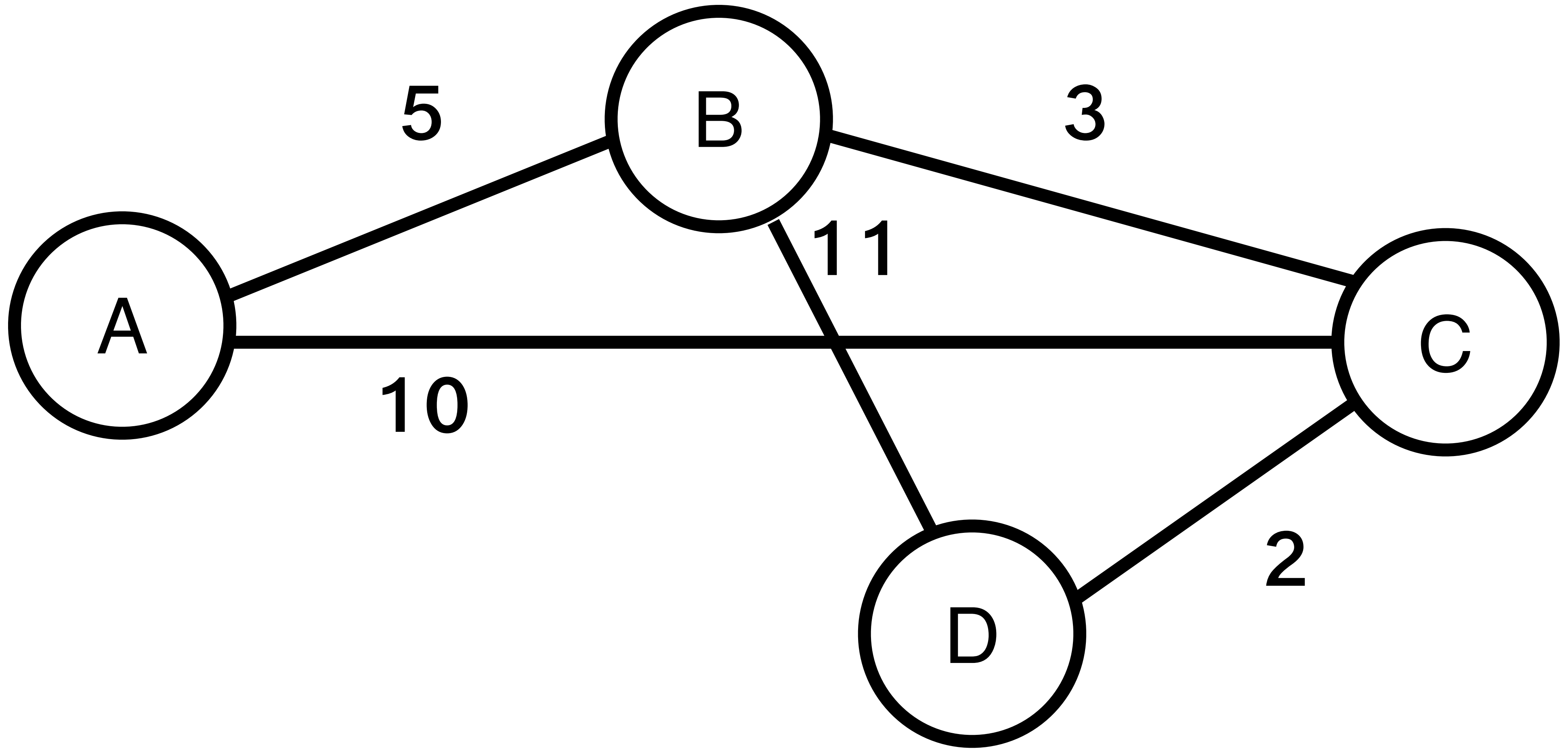
# Step 2: Route Calculation




**Problem formulation: compute the shortest path between any two nodes i and j, given**

- N: the set of nodes in the graph
- l(i,j): the non-negative cost associated with the edge between nodes i, j$\in$N and l(i,j) = $\infty$ if no edge connects i and j

# Step 2: Route Calculation — Dijkstra Algorithm




**Problem formulation: compute the shortest path between any two nodes i and j, given**

- N: the 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

# Dijkstra's Shortest-Path Routing

## Input

- N: the set of nodes in the graph
- $l(i,j)$: the non-negative cost associated with the edge between nodes $i, j \in N$ and $l(i,j) = \infty$ if no edge connects i and j

**Let $s \in N$ be the starting node which executes the algorithm to find shortest paths to all other nodes in N**

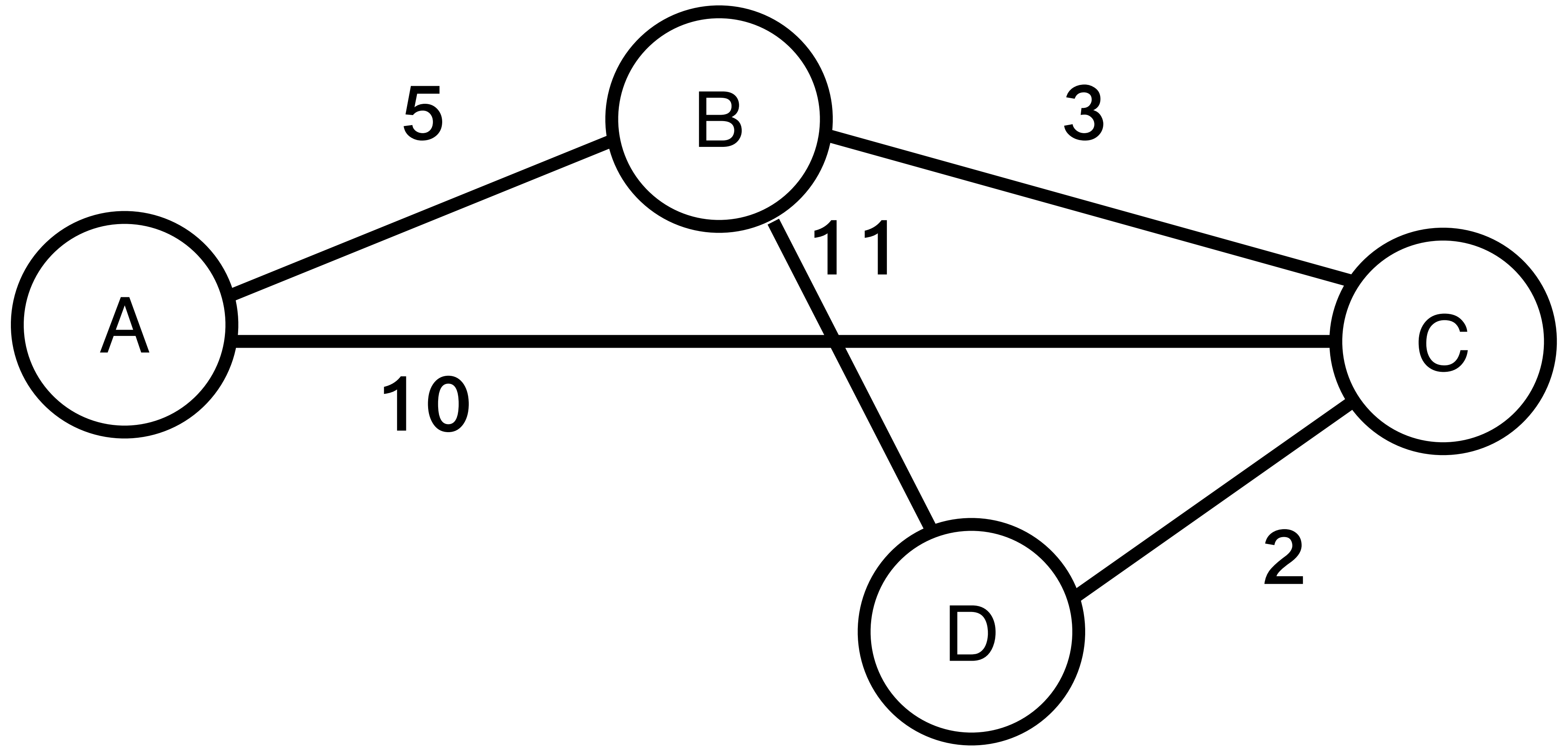
# Dijkstra’s Shortest-Path Routing Algorithm

## Algorithm:

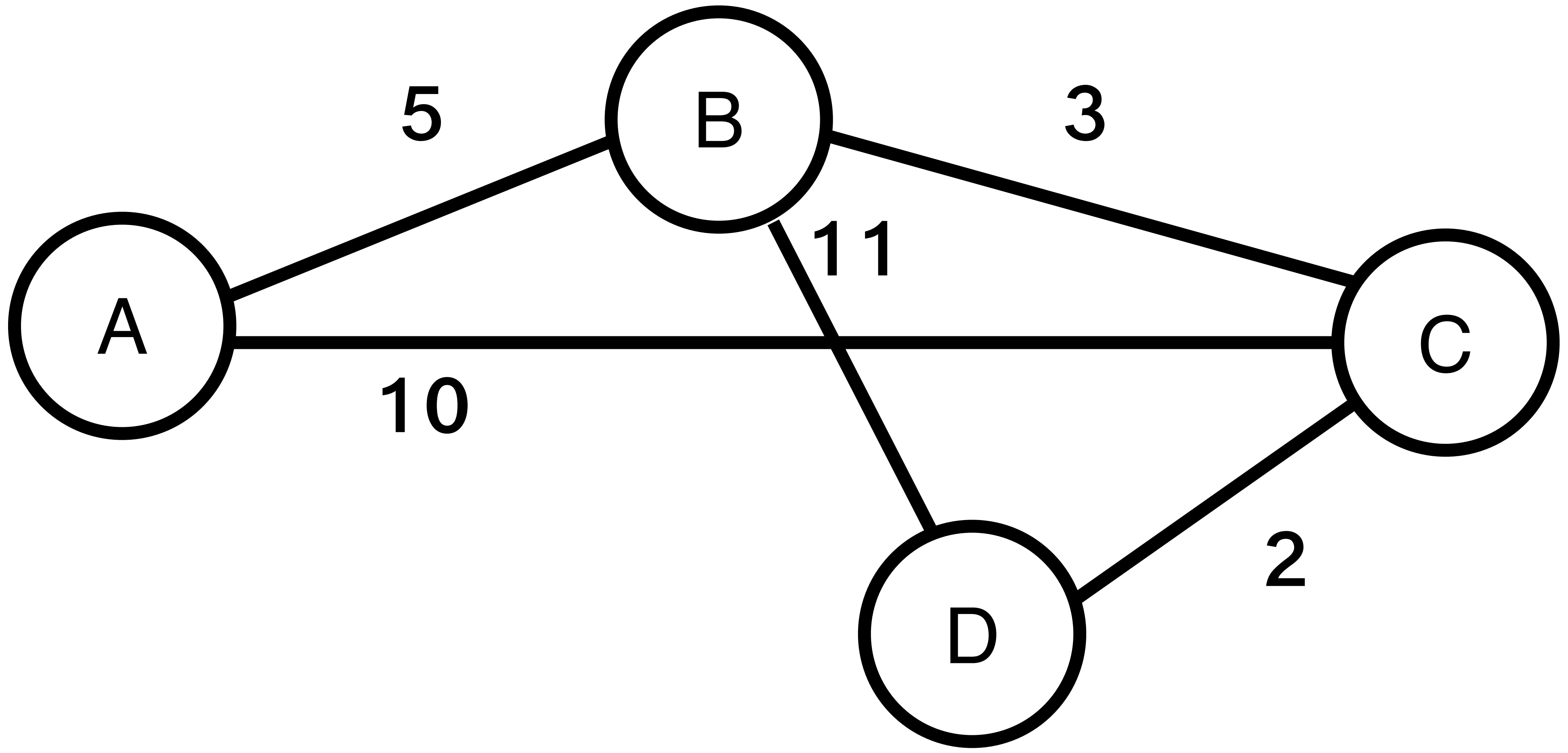
- M: set of nodes incorporated so far by the algorithm
- C(n): the cost the path from s to each node n

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)   /* costs of directly connected nodes */
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)    /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

# Building Routing Table for Node D

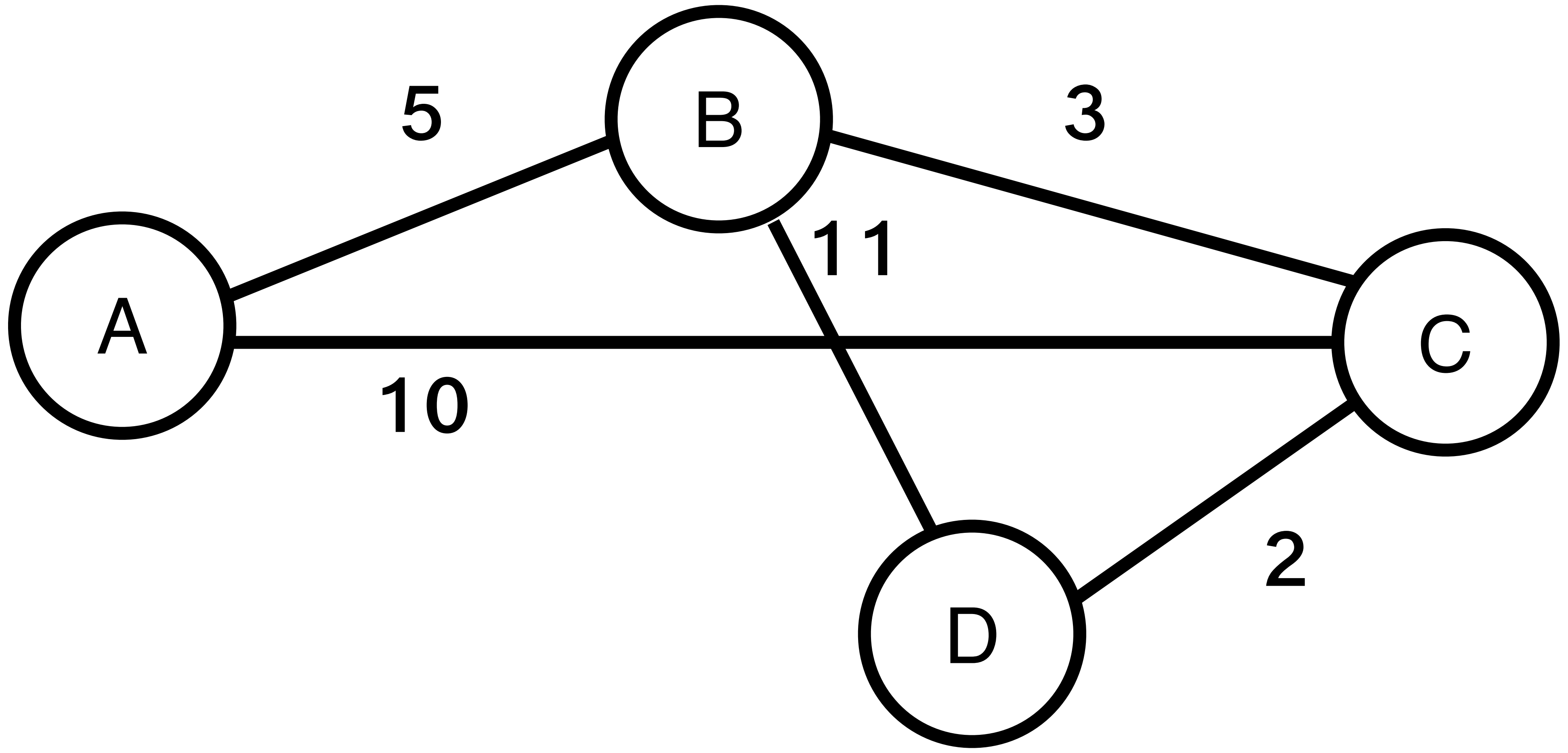


# Building Routing Table for Node D




| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

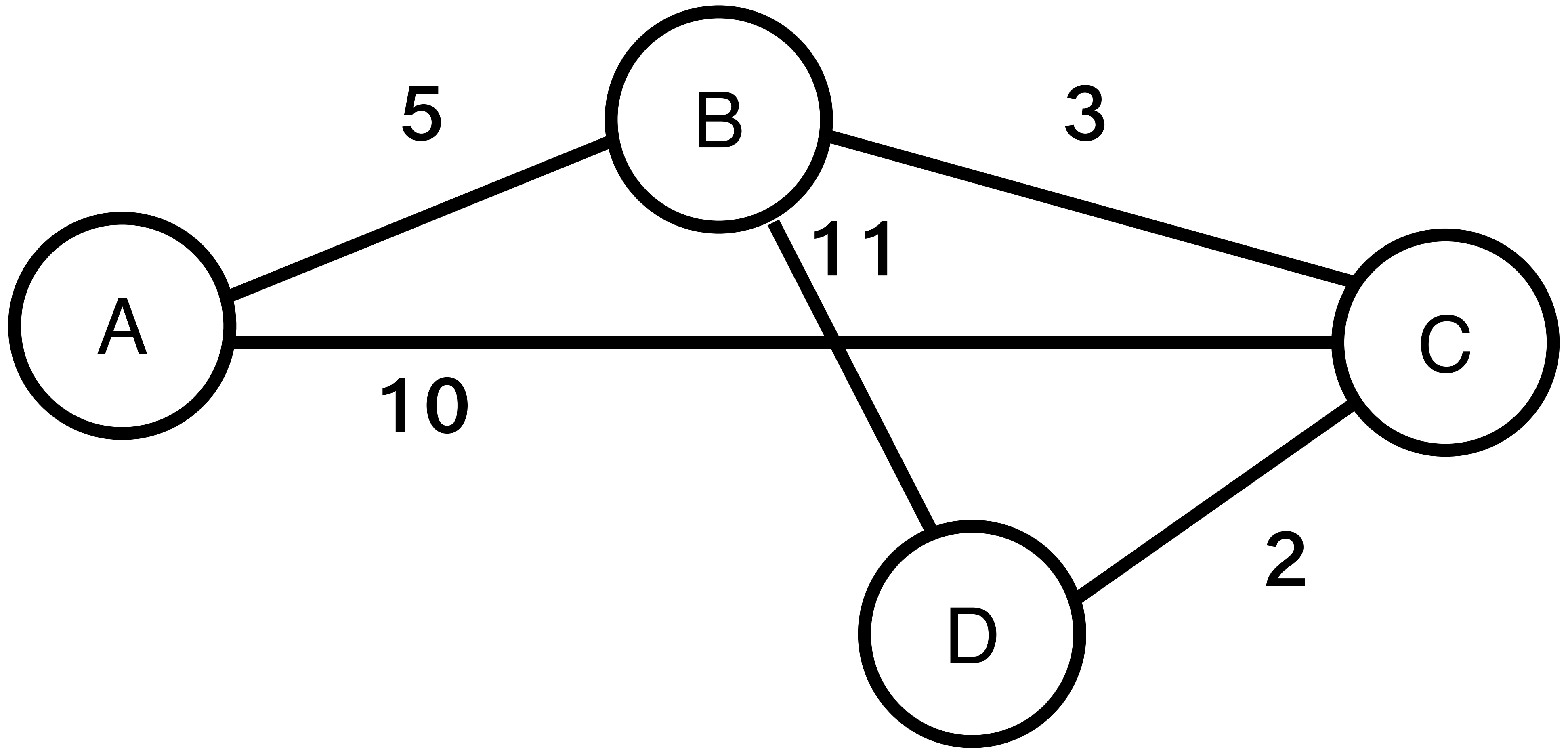
# Building Routing Table for Node D




| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

M from the above algorithm

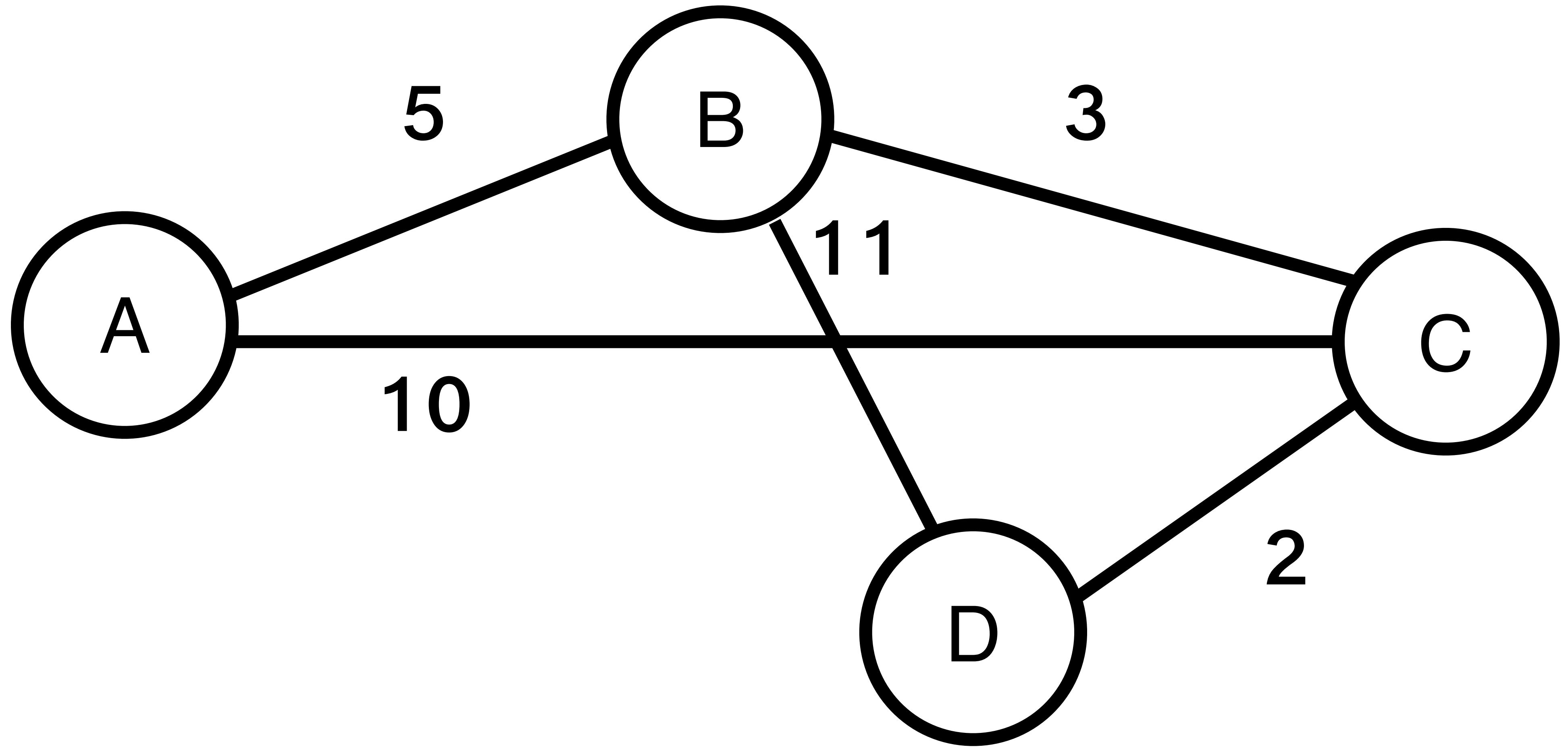
# Building Routing Table for Node D




| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

(N-M) from the above algorithm

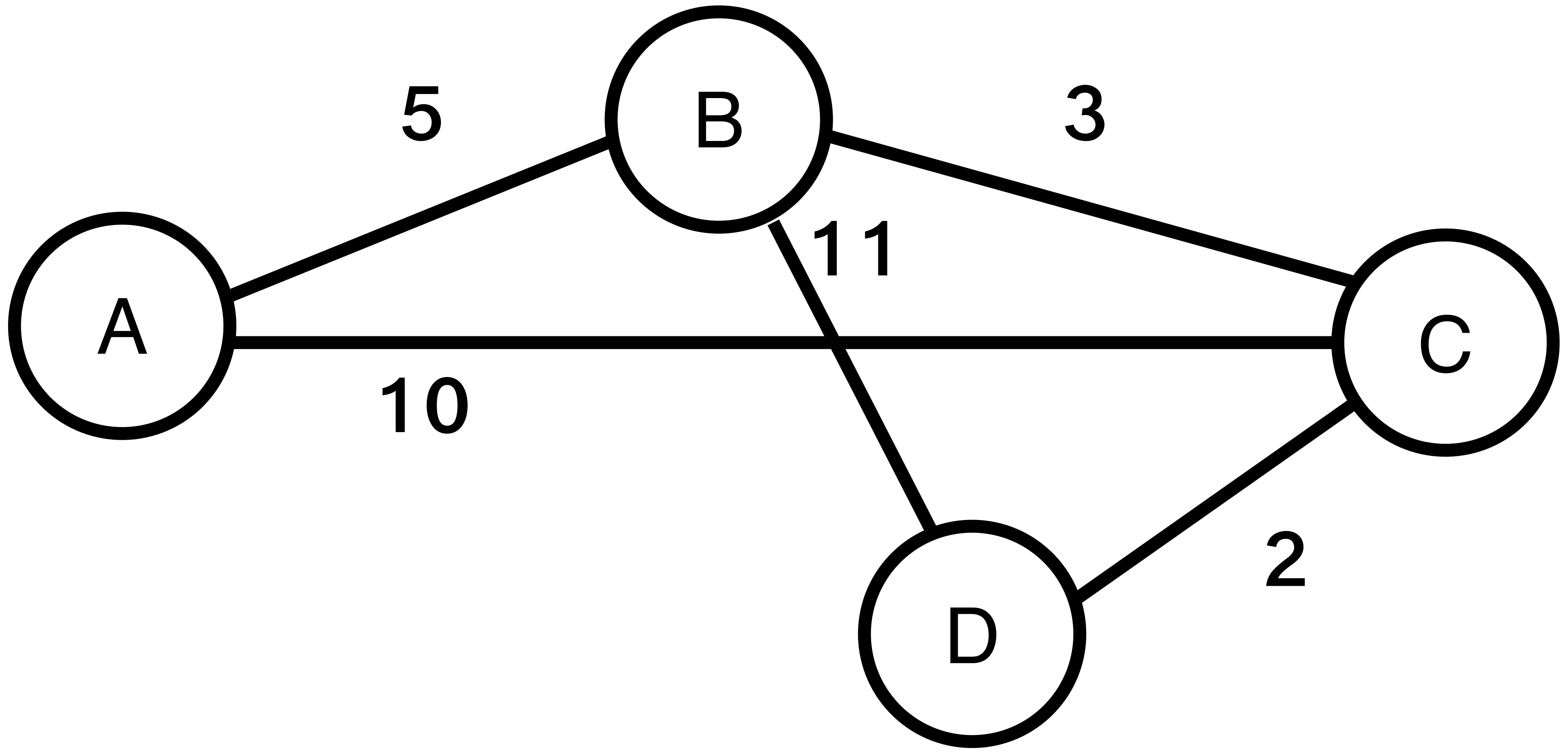
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Building Routing Table for Node D

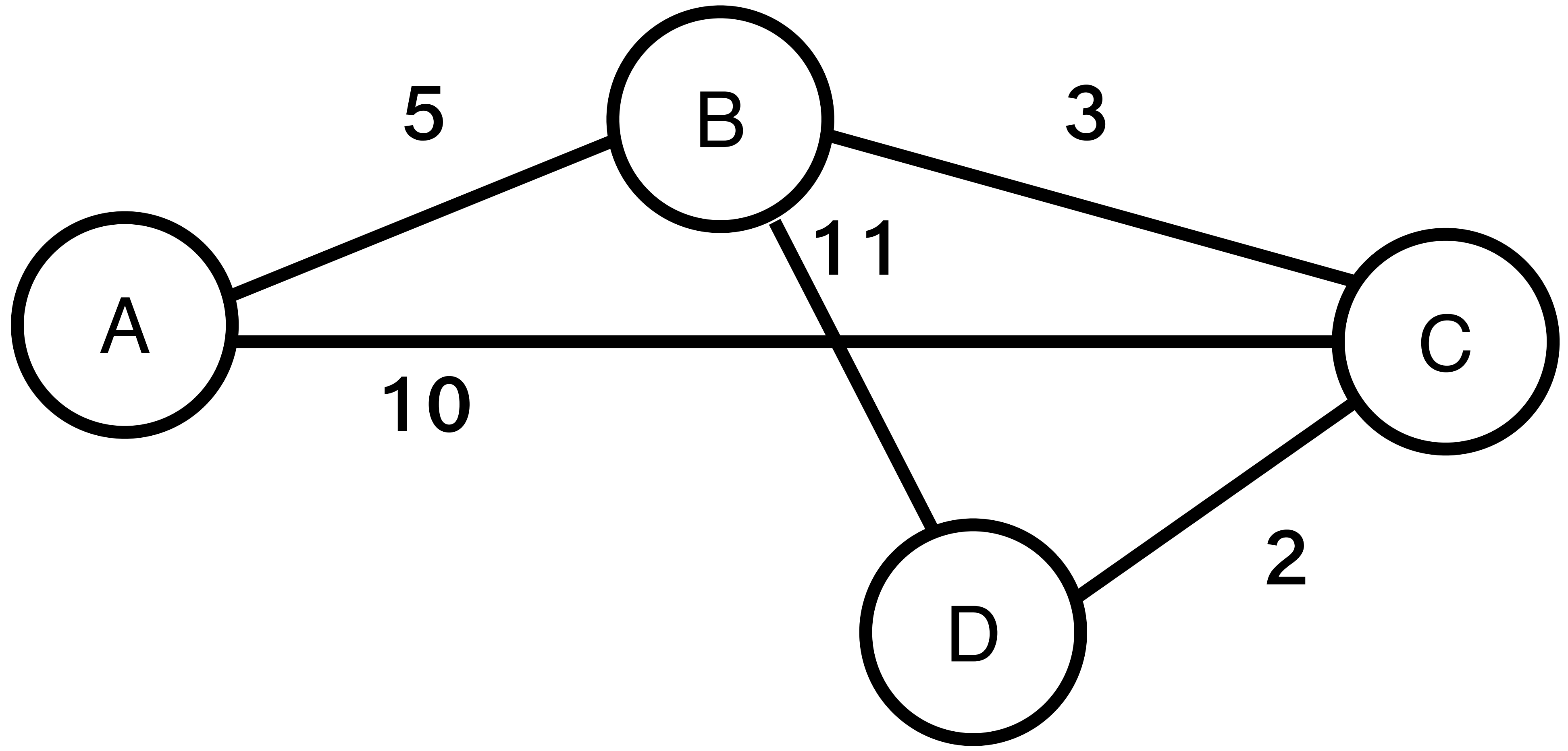



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)    /* costs of directly connected nodes */
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)     /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

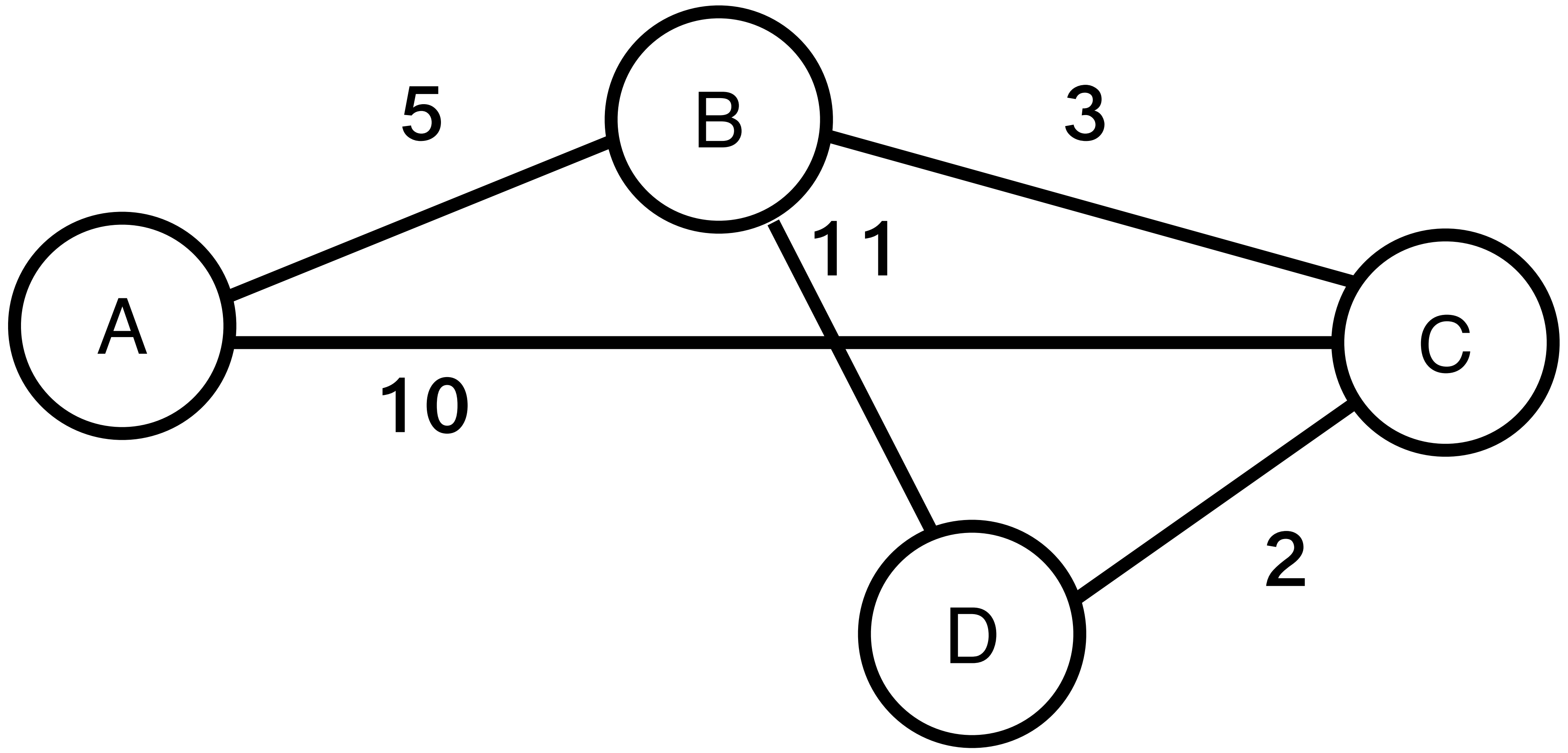
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| | | | |
| | | | |
| | | | |
| | | | |

# Building Routing Table for Node D

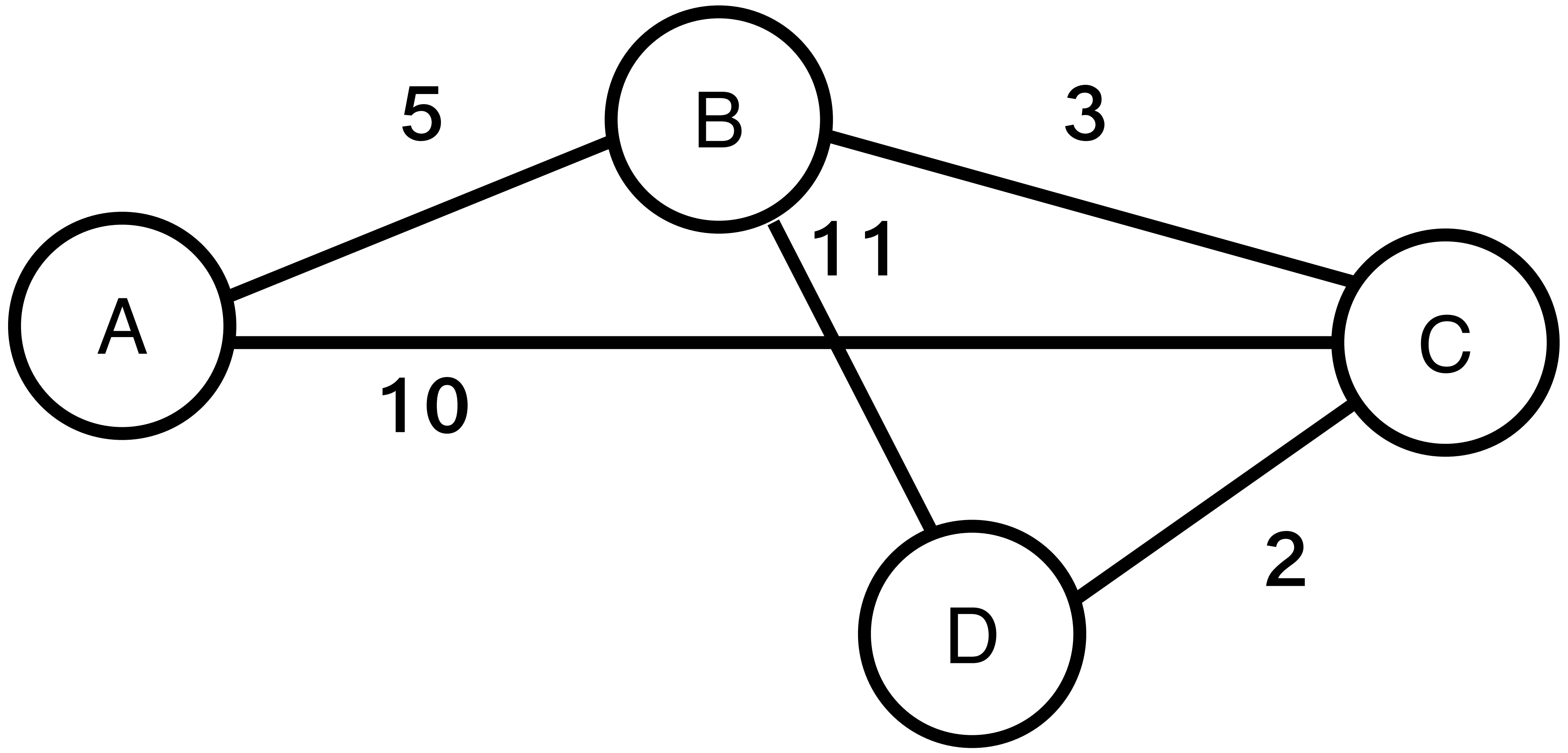



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)    /* costs of directly connected nodes */
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)     /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

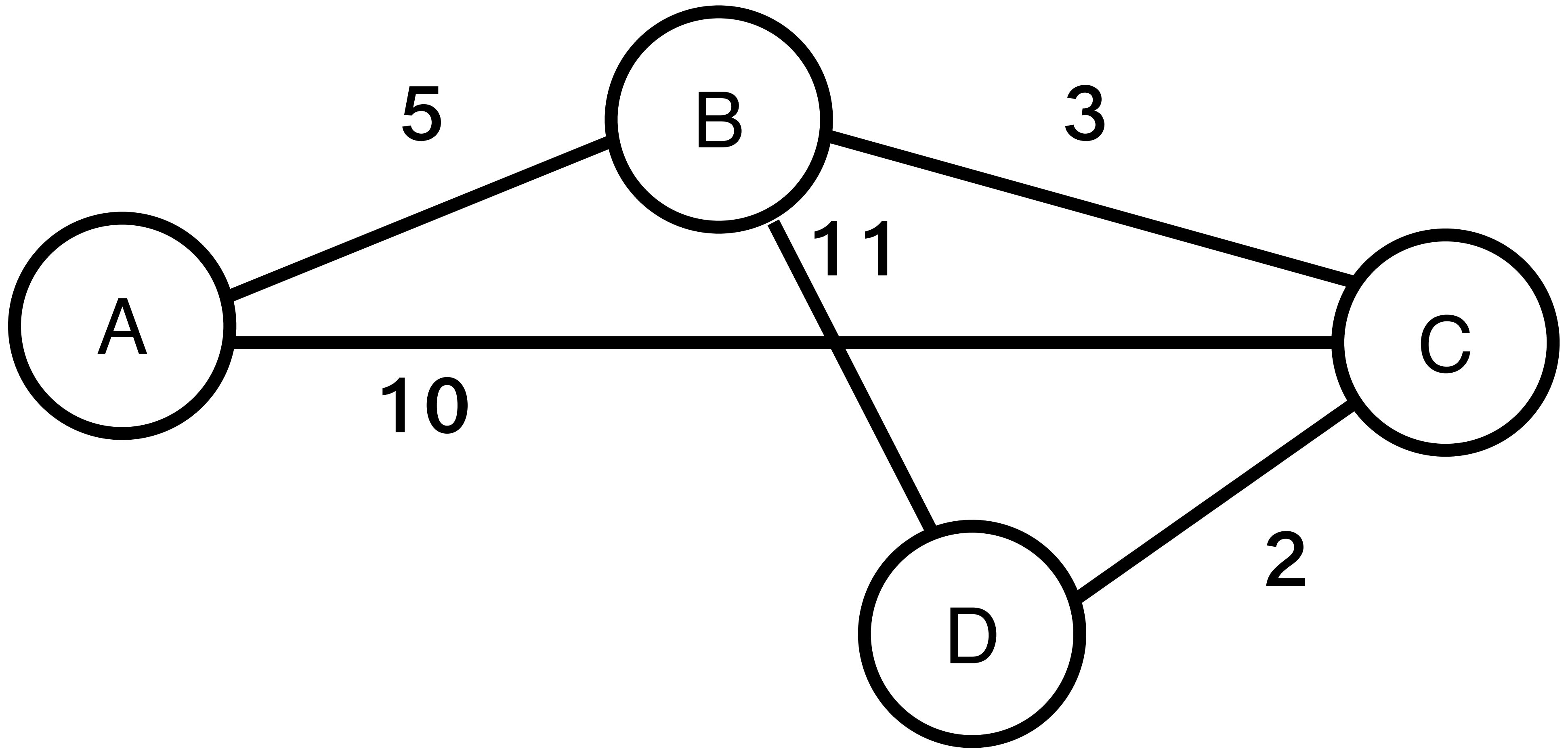
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| 3 | (D, 0, -), (C, 2, C) | (B, 11, B) | Integrate lowest-cost member of tentative list |
| | | | |
| | | | |
| | | | |

# Building Routing Table for Node D

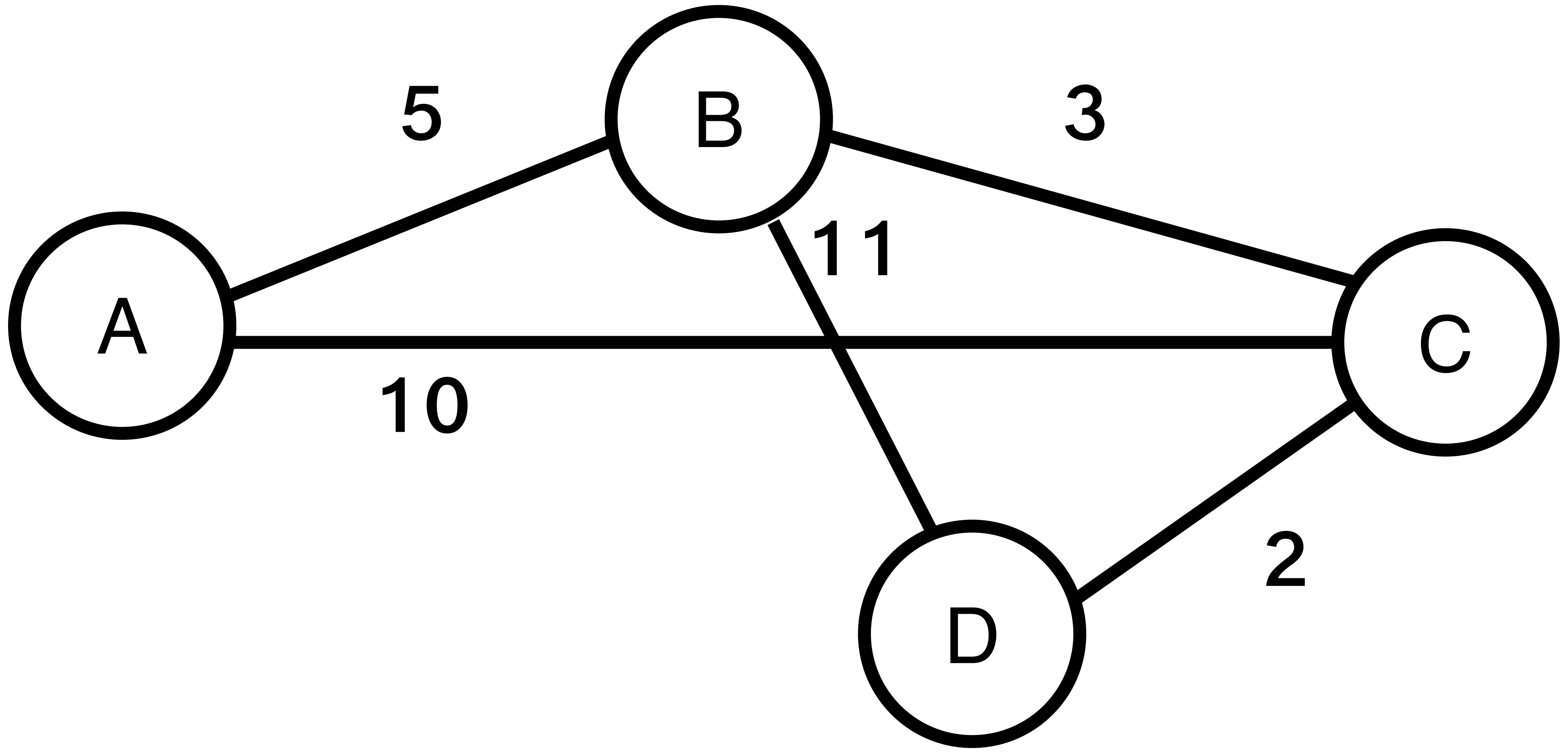



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)    /* costs of directly connected nodes */
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)      /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

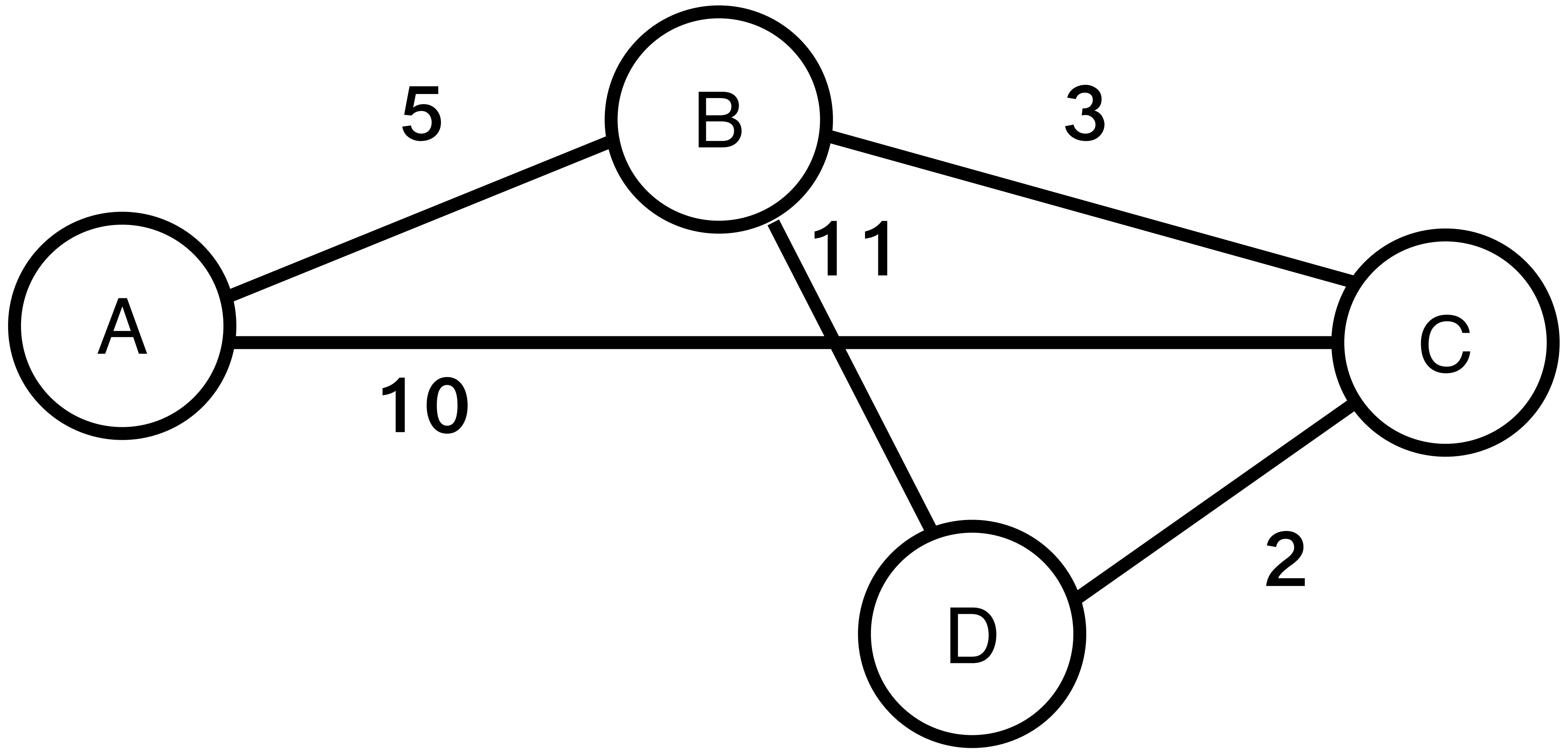
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| 3 | (D, 0, -), (C, 2, C) | (B, 11, B) | Integrate lowest-cost member of tentative list |
| 4 | (D, 0, -), (C, 2, C) | (B, 5, C), (A, 12, C) | Based on C's LSP and recalculate the cost |
| | | | |
| | | | |

# Building Routing Table for Node D

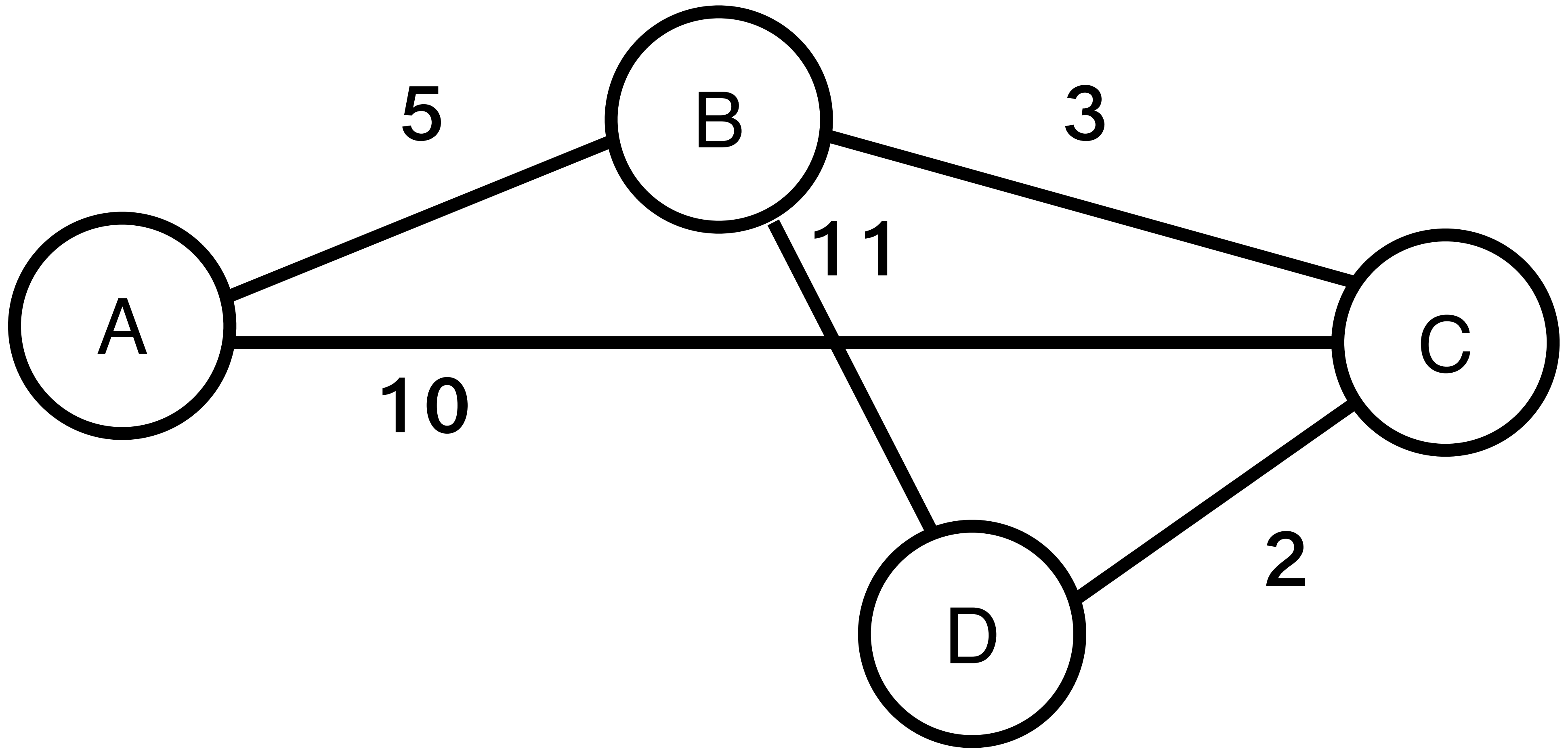



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
M = {S}
for each n in N - {S}
    C(n) = l(s, n)    /* costs of directly connected nodes */
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)     /* recalculate costs */
       C(n) = MIN(C(n), C(w) + l(w,n))
```

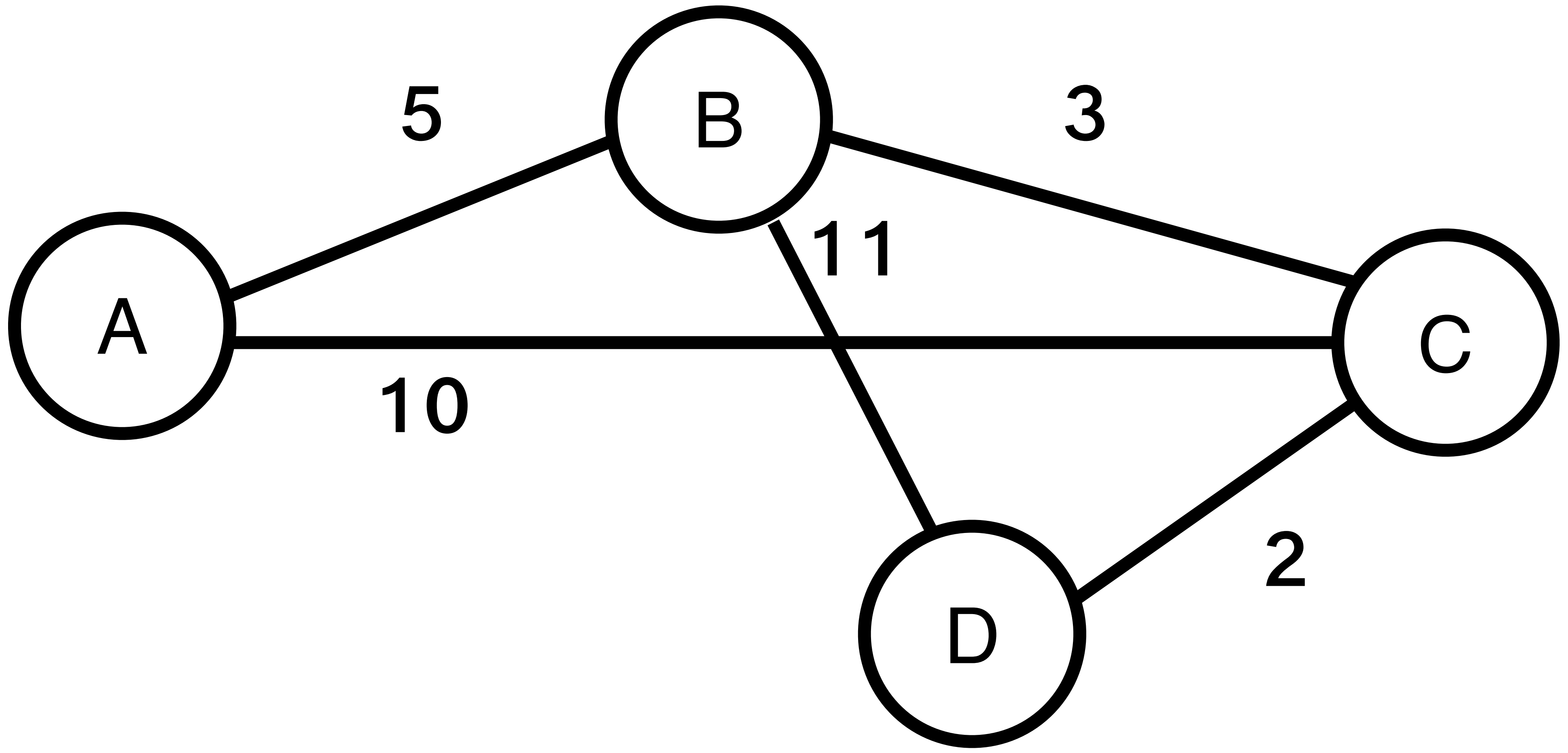
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
| --- | --- | --- | --- |
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| 3 | (D, 0, -), (C, 2, C) | (B, 11, B) | Integrate lowest-cost member of tentative list |
| 4 | (D, 0, -), (C, 2, C) | (B, 5, C), (A, 12, C) | Based on C's LSP and recalculate the cost |
| 5 | (D, 0, -), (C, 2, C), (B, 5, C) | (A, 12, C) | Integrate lowest-cost member of tentative list |
| | | | |

# Building Routing Table for Node D

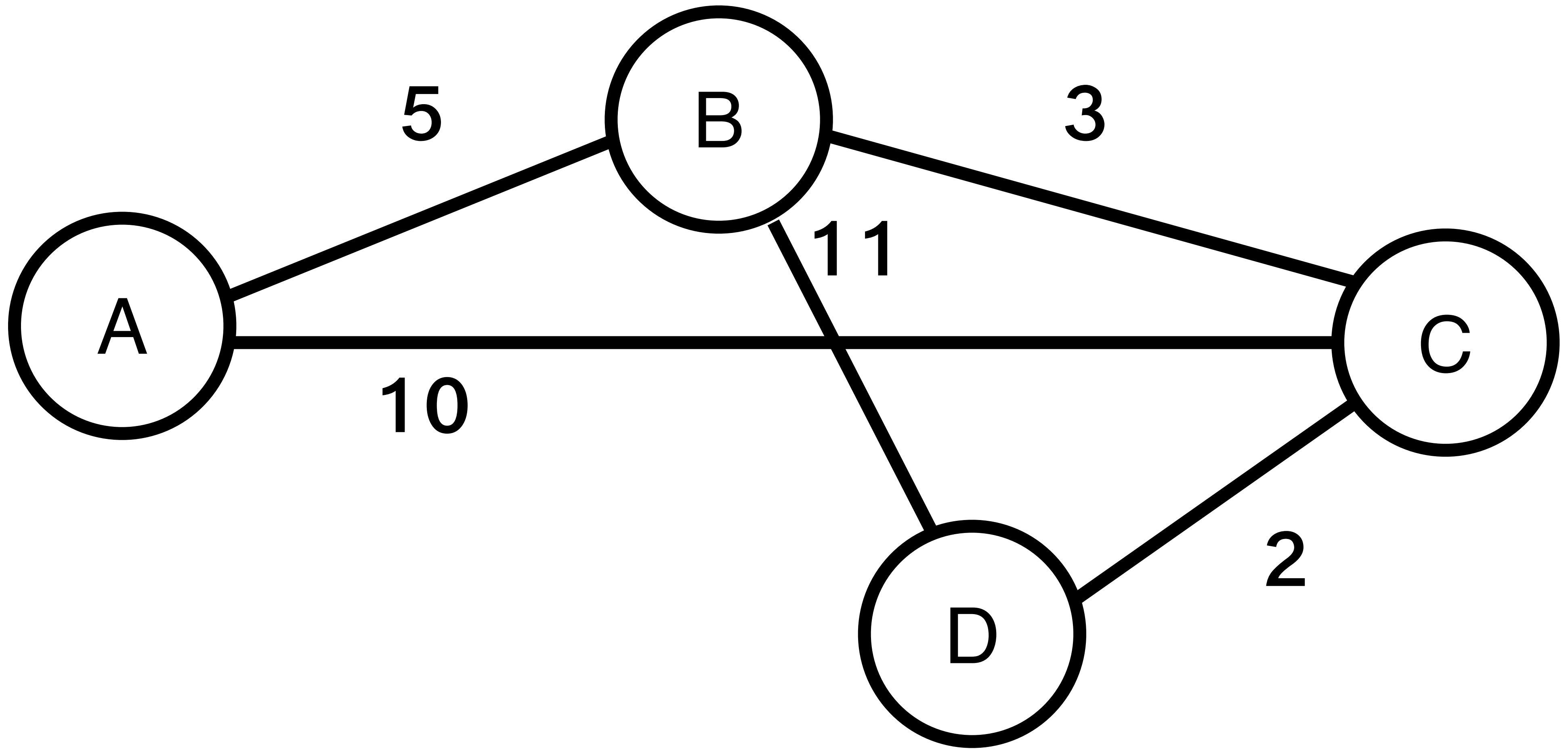



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)    /* costs of directly connected nodes */
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)      /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

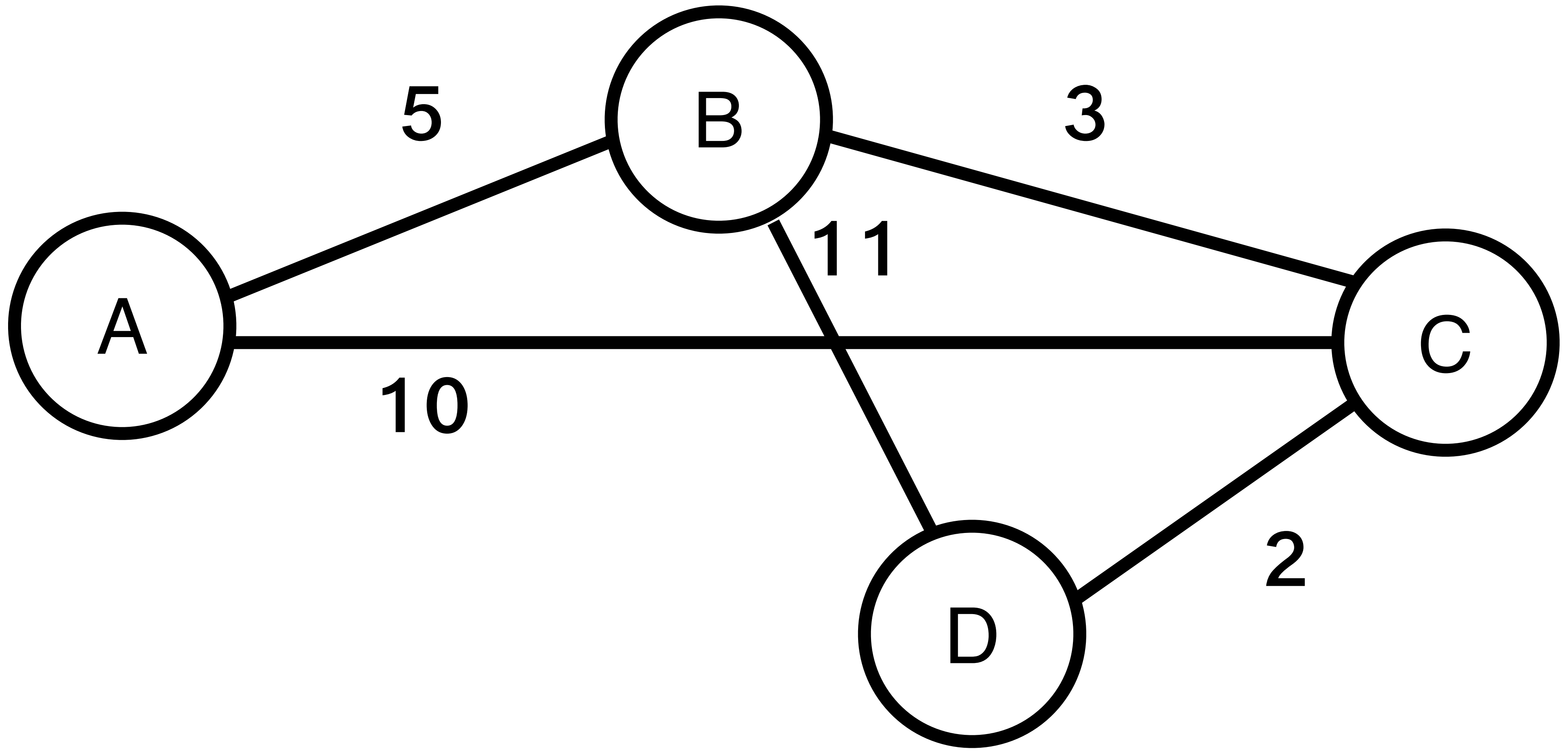
# Building Routing Table for Node D




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| 3 | (D, 0, -), (C, 2, C) | (B, 11, B) | Integrate lowest-cost member of tentative list |
| 4 | (D, 0, -), (C, 2, C) | (B, 5, C), (A, 12, C) | Based on C's LSP and recalculate the cost |
| 5 | (D, 0, -), (C, 2, C), (B, 5, C) | (A, 12, C) | Integrate lowest-cost member of tentative list |
| 6 | (D, 0, -), (C, 2, C), (B, 5, C) | (A, 10, C) | Based on B's LSP, i.e., l(D, A) = l(D, B) + l(B, A) |

# Building Routing Table for Node D

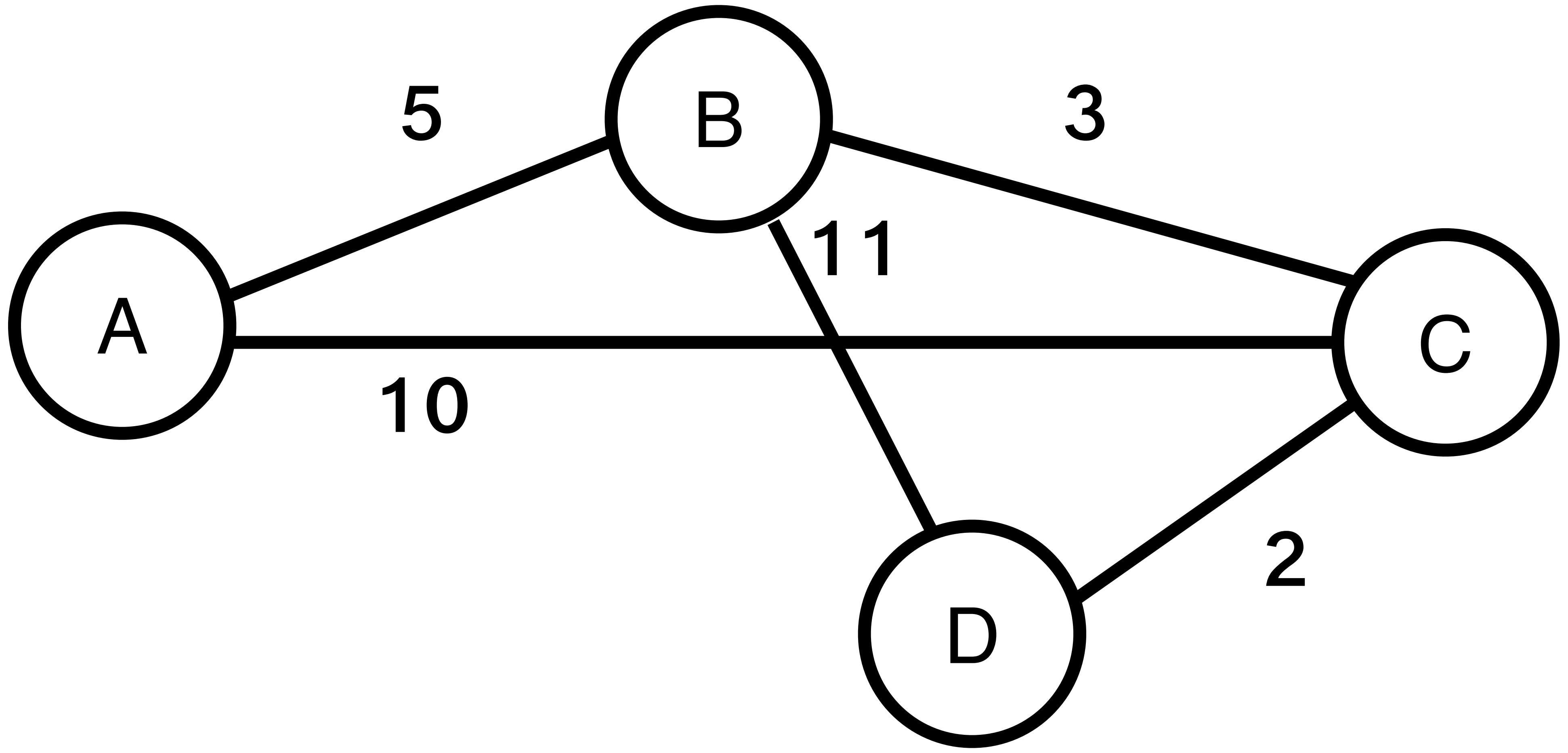



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 6 | (D, 0, -), (C, 2, C), (B, 5, C) | (A, 10, C) | Based on B's LSP, i.e., l(D, A) = l(D, B) + l(B, A) |

```
M = {S}
for each n in N - {S}
   C(n) = l(s, n)    /* costs of directly connected nodes */
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)     /* recalculate costs */
      C(n) = MIN(C(n), C(w) + l(w,n))
```

# Building Routing Table for Node D

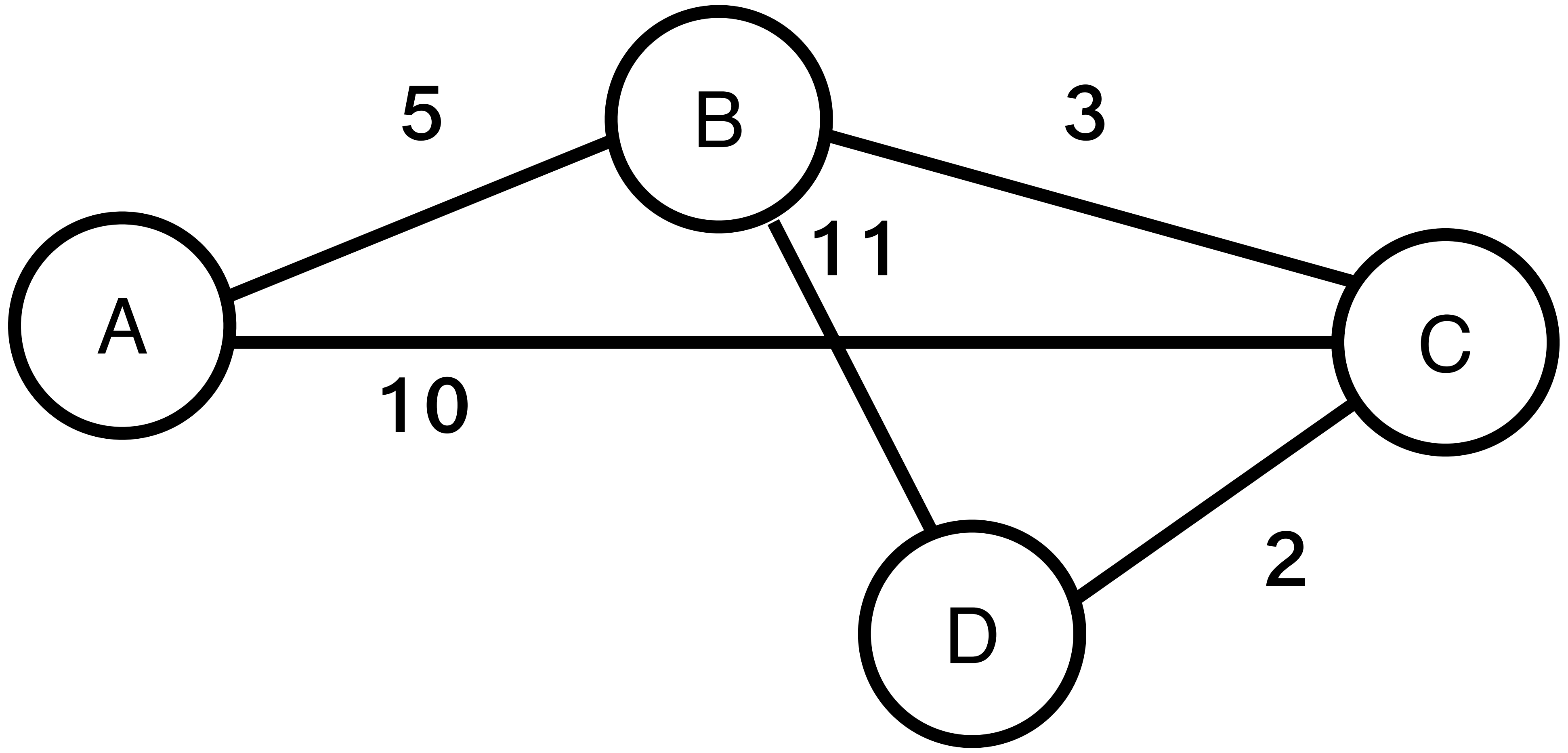



Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (D, 0, -) | | Initialize with an entry for myself |
| 2 | (D, 0, -) | (B, 11, B), (C, 2, C) | Based on D's LSP |
| 3 | [illegible] | [illegible] | …tative list |
| 4 | [illegible] | [illegible] | …he cost |
| 5 | (D, … | [illegible] | …tative list |
| 6 | (D, 0, -), (C, 2, C), (B, 5, C) | (A, 10, C) | Based on B's LSP, i.e., l(D, A) = l(D, B) + l(B, A) |

(D, 0, -), (C, 2, C), (B, 5, C), (A, 10, C)

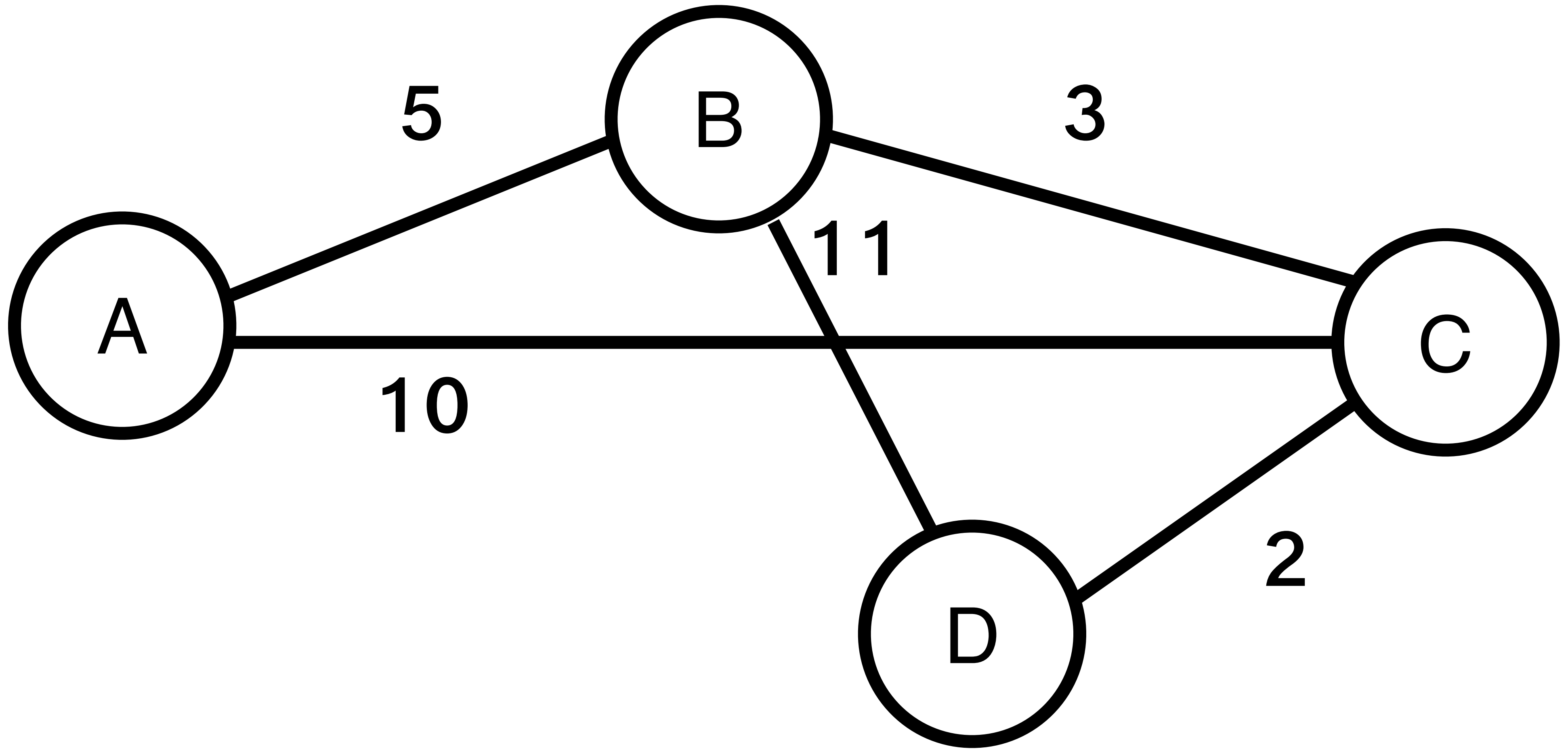
# Building Routing Table for Node A




Routing table entry:
(Destination, Cost, NextHop)

| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

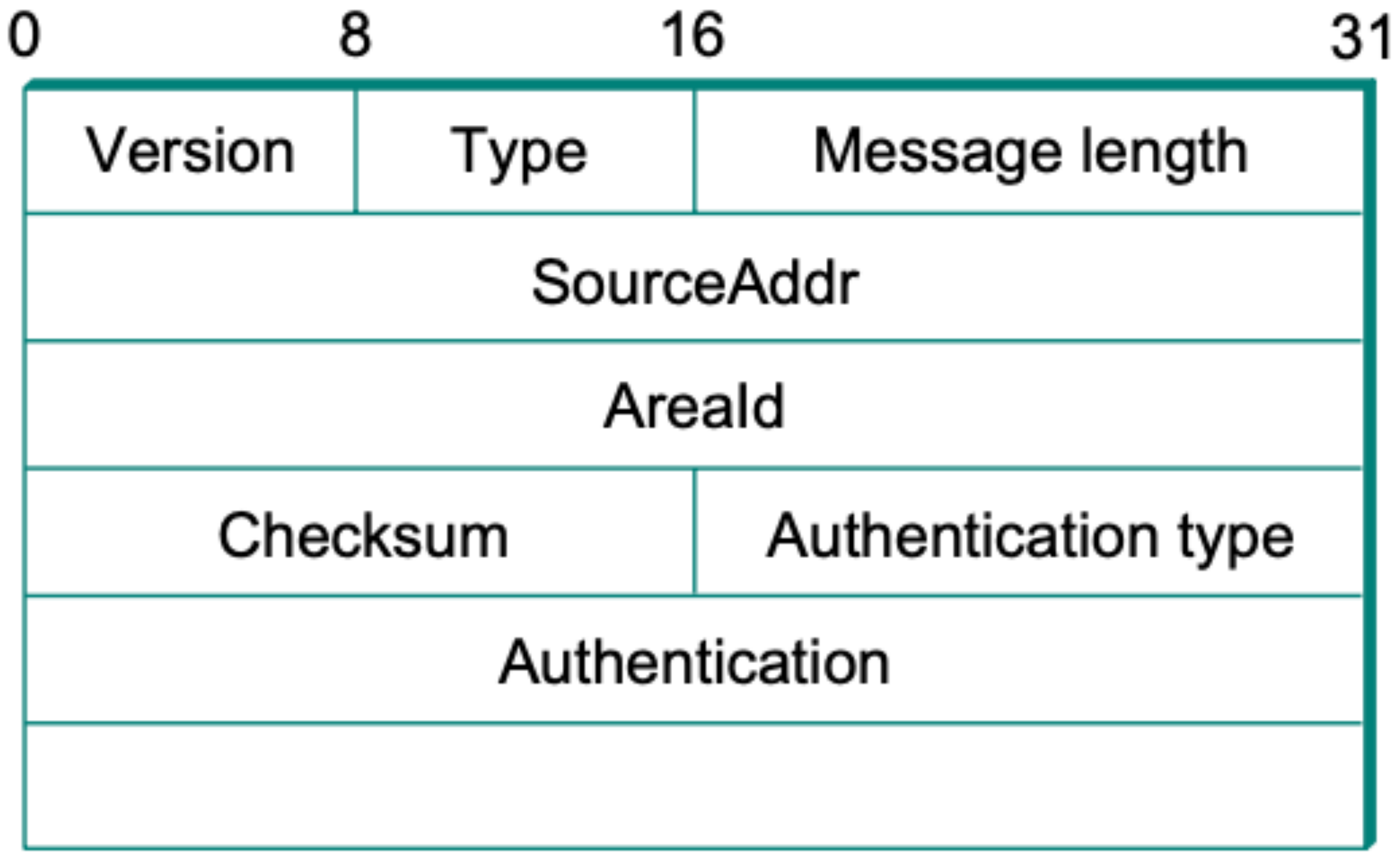
# Building Routing Table for Node A




Routing table entry:
(Destination, Cost, NextHop)

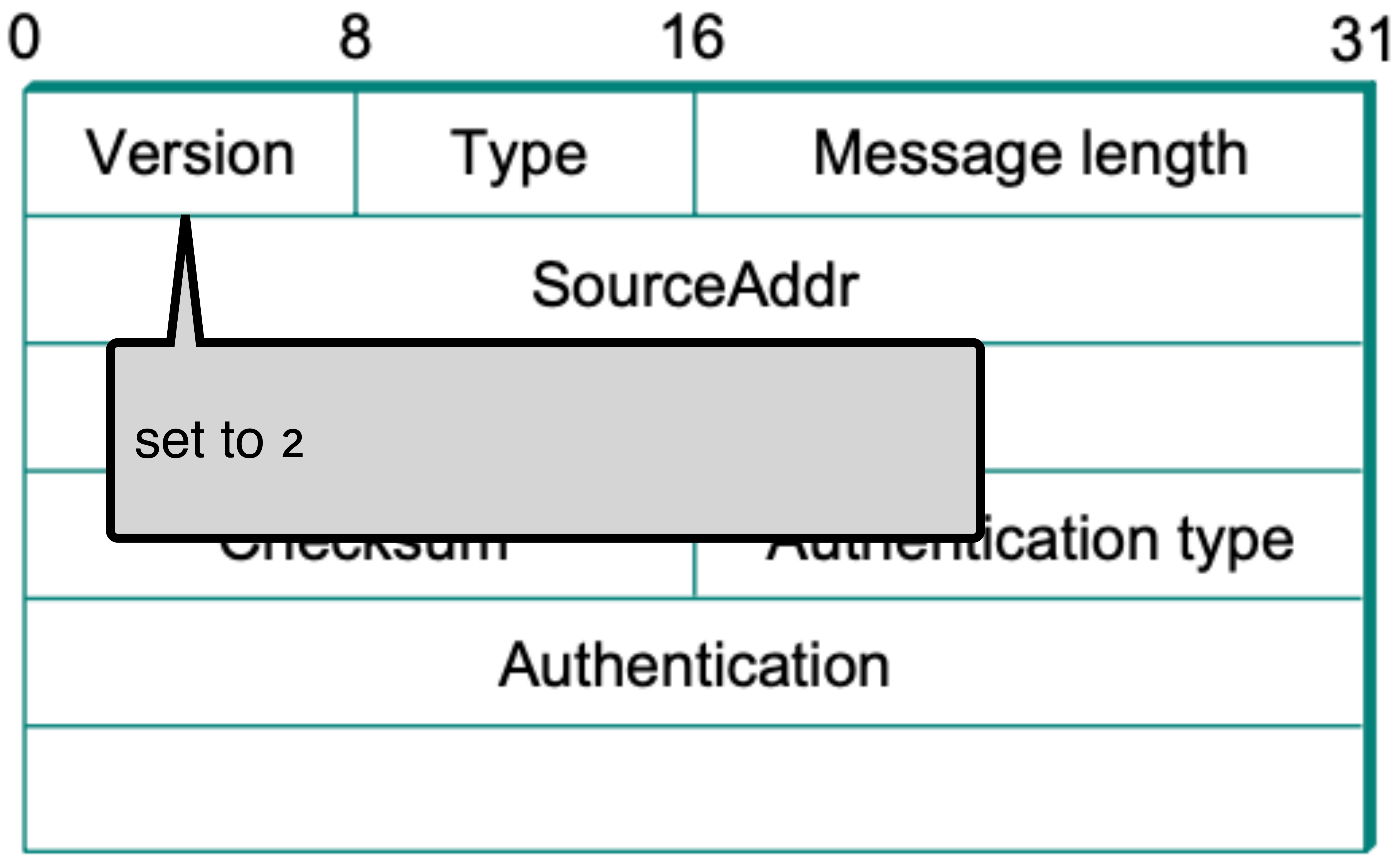
| Step | Confirmed list | Tentative list | Comment |
|---|---|---|---|
| 1 | (A, 0, -) | | Initialize an entry for my self |
| 2 | (A, 0, -) | (B, 5, B), (C, 10, C) | Based on A's LSP |
| 3 | (A, 0, -), (B, 5, B) | (C, 10, C) | Integrate lowest-cost member of tentative list |
| 4 | (A, 0, -), (B, 5, B) | (C, 8, B), (D, 16, B) | Based on B's LSP and recalculate the cost |
| 5 | (A, 0, -), (B, 5, B), (C, 8, B) | (D, 16, B) | Integrate lowest-cost member of tentative list |
| 6 | (A, 0, -), (B, 5, B), (C, 8, B) | (D, 10, B) | Based on C's LSP, i.e., l(A, D) = l(A, C) + l(C, D) |
| 7 | (A, 0, -), (B, 5, B), (C, 8, B), (D, 10, B) | | Integrate lowest-cost member of tentative list |

# Open Shortest Path First (OSPF)

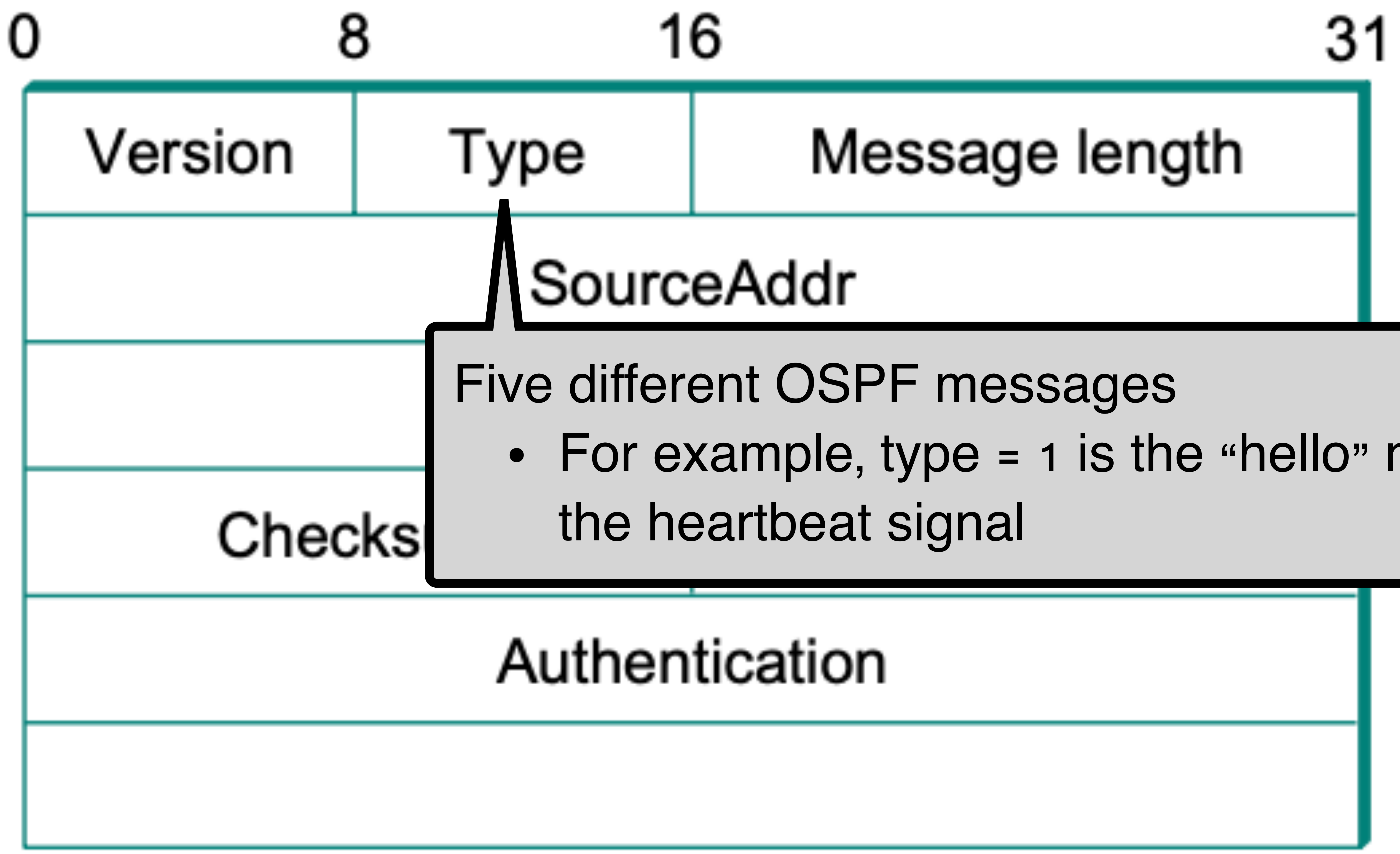



**OSPF header format**

# Open Shortest Path First (OSPF)



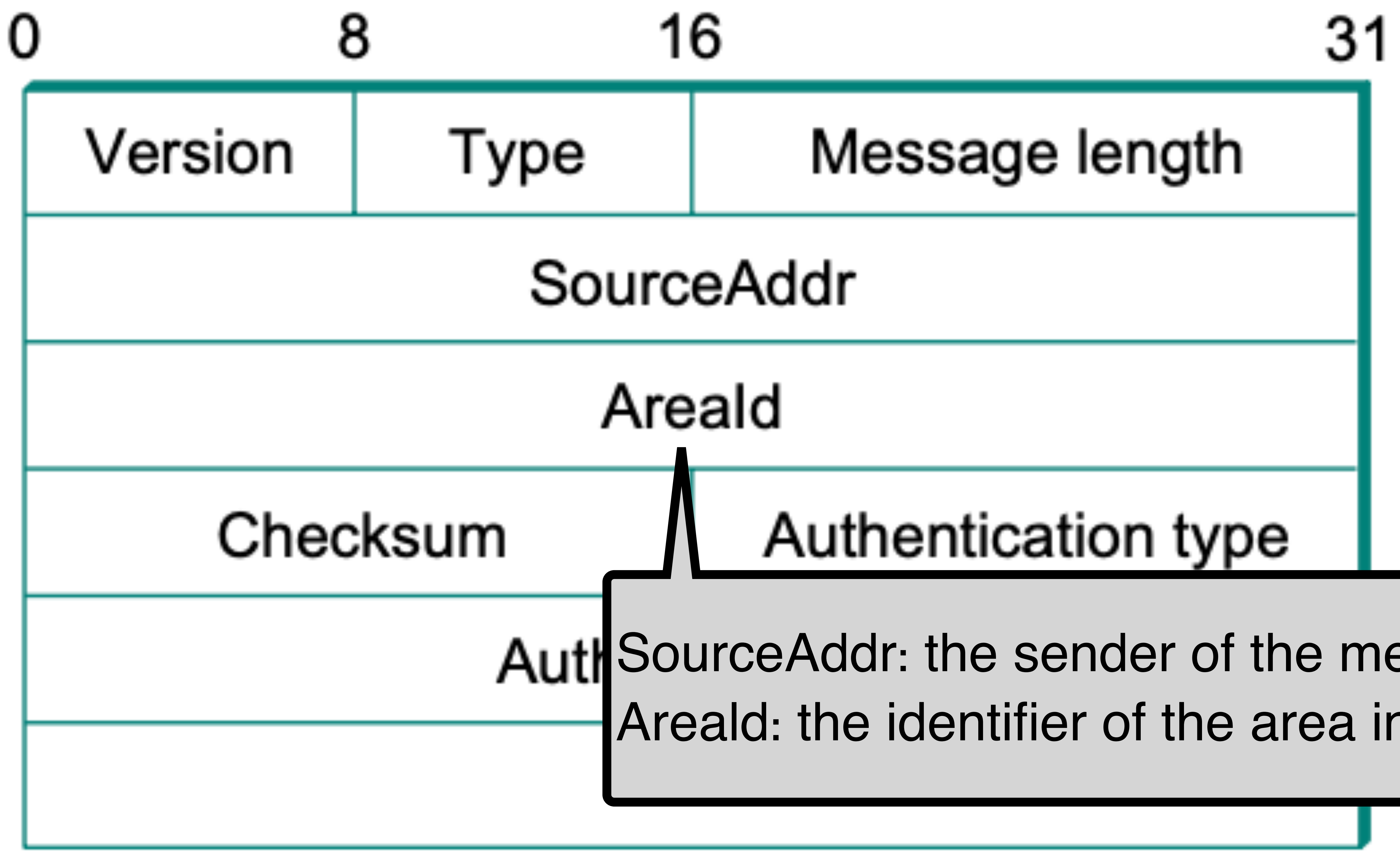
# Open Shortest Path First (OSPF)




Five different OSPF messages

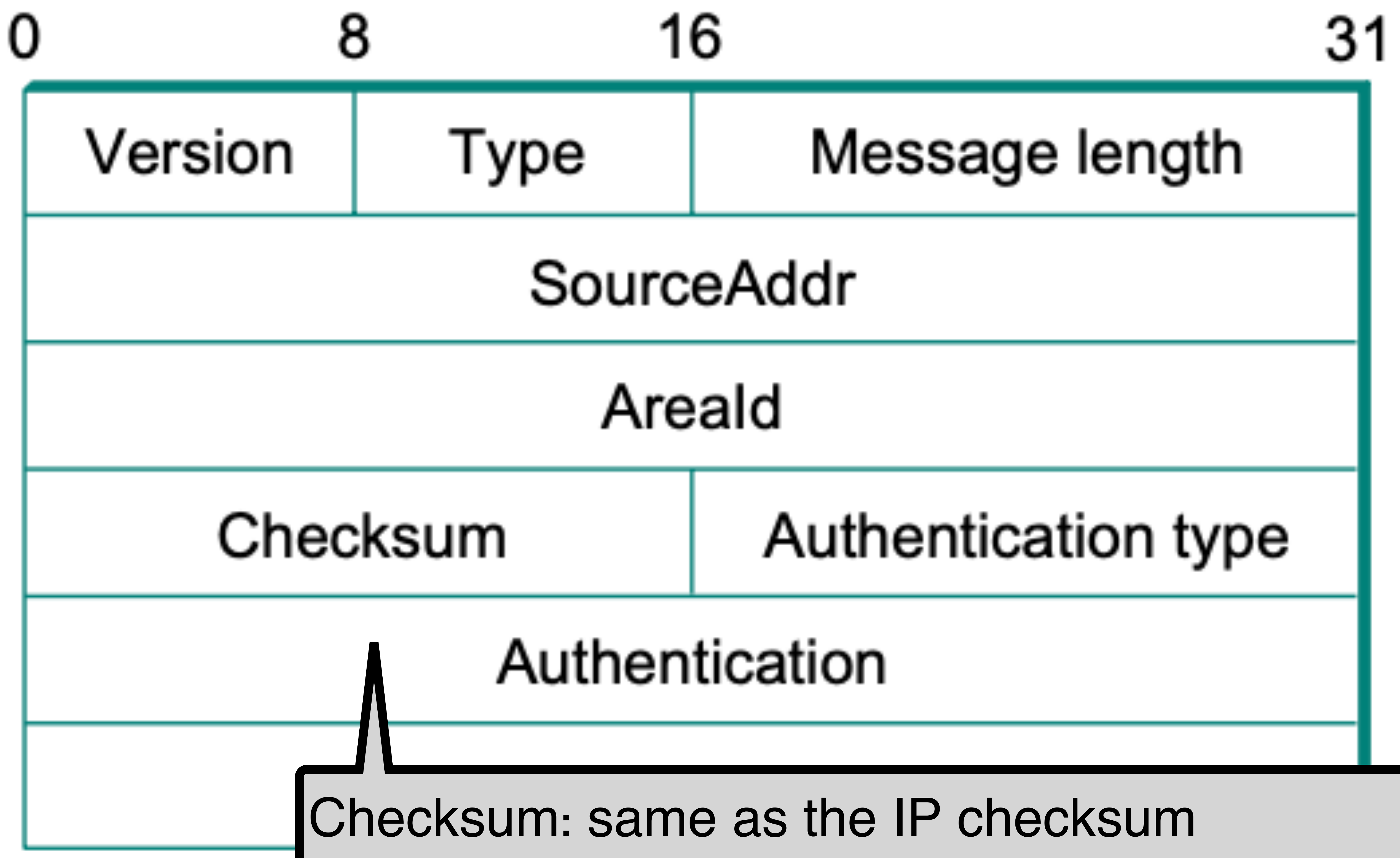
- For example, type = 1 is the “hello” message as the heartbeat signal

# Open Shortest Path First (OSPF)




SourceAddr: the sender of the message
AreaId: the identifier of the area in which the node is located

# Open Shortest Path First (OSPF)




Checksum: same as the IP checksum
Authentication:

- 0, no authentication
- 1, a simple password
- 2, a cryptographic authentication checksum

# Open Shortest Path First (OSPF)

<table>
<tr><td colspan="3">LS Age</td><td>Options</td><td>Type=1</td></tr>
<tr><td colspan="5">Link-state ID</td></tr>
<tr><td colspan="5">Advertising router</td></tr>
<tr><td colspan="5">LS sequence number</td></tr>
<tr><td colspan="3">LS checksum</td><td colspan="2">Length</td></tr>
<tr><td>0</td><td>Flags</td><td>0</td><td colspan="2">Number of links</td></tr>
<tr><td colspan="5">Link ID</td></tr>
<tr><td colspan="5">Link data</td></tr>
<tr><td colspan="2">Link type</td><td>Num_TOS</td><td colspan="2">Metric</td></tr>
<tr><td colspan="5">Optional TOS information</td></tr>
<tr><td colspan="5">More links</td></tr>
</table>

**OSPF link-state advertisement**

# Open Shortest Path First (OSPF)

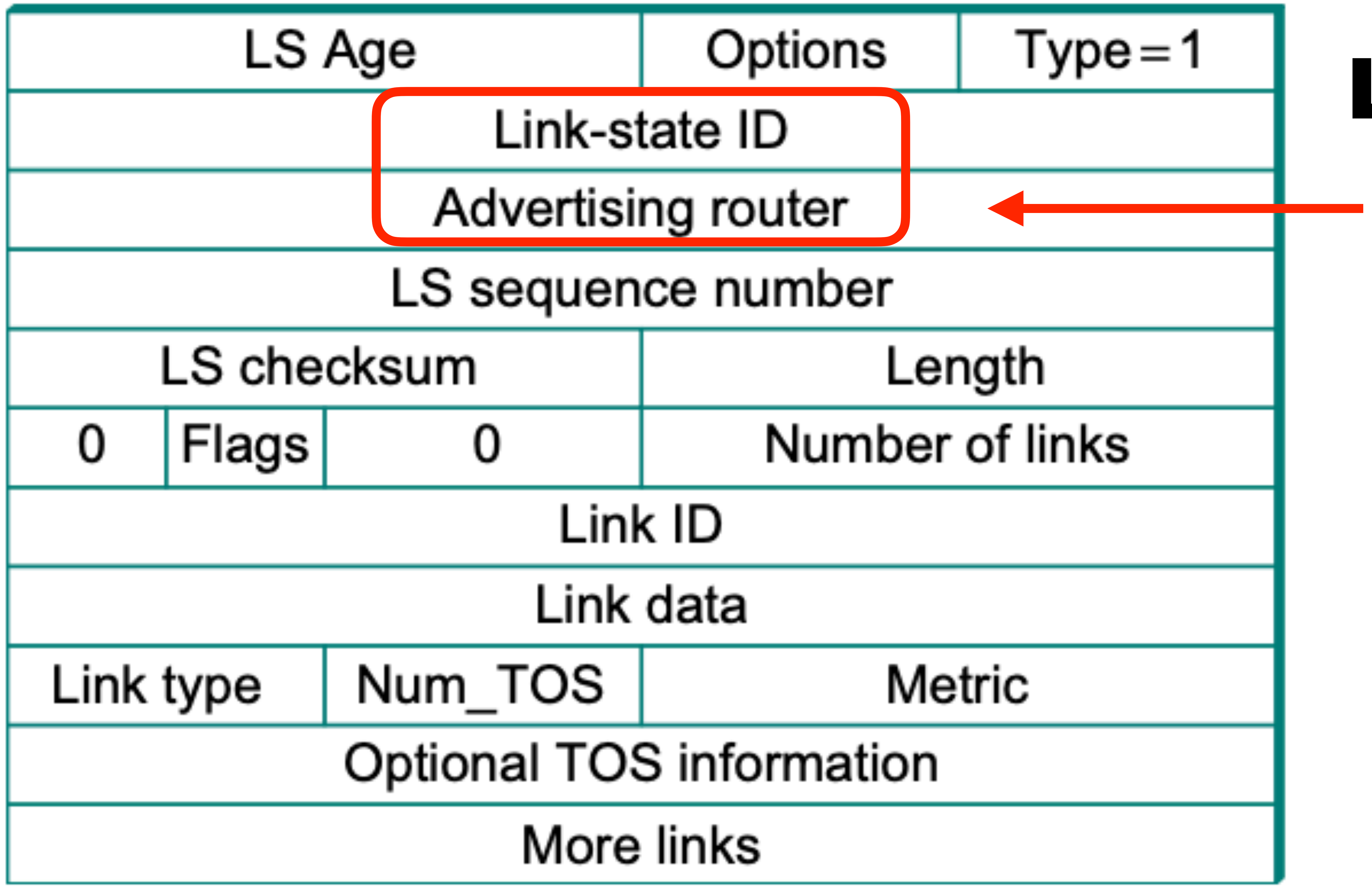
<table>
<tr><td colspan="3">LS Age</td><td>Options</td><td>Type=1</td></tr>
<tr><td colspan="5">Link-state ID</td></tr>
<tr><td colspan="5">Advertising router</td></tr>
<tr><td colspan="5">LS sequence number</td></tr>
<tr><td colspan="3">LS checksum</td><td colspan="2">Length</td></tr>
<tr><td>0</td><td>Flags</td><td>0</td><td colspan="2">Number of links</td></tr>
<tr><td colspan="5">Link ID</td></tr>
<tr><td colspan="5">Link data</td></tr>
<tr><td colspan="2">Link type</td><td>Num_TOS</td><td colspan="2">Metric</td></tr>
<tr><td colspan="5">Optional TOS information</td></tr>
<tr><td colspan="5">More links</td></tr>
</table>

**OSPF link-state advertisement**

## Link state packet (LSP)

- ID of the node that created the LSP
- Cost of link to each directly connect neighbor
- Sequence number (SEQ#)
- Time-to-live (TTL) for this packet

# Open Shortest Path First (OSPF)




**OSPF link-state advertisement**

## Link state packet (LSP)

- ID of the node that created the LSP
- Cost of link to each directly connect neighbor
- Sequence number (SEQ#)
- Time-to-live (TTL) for this packet

# Open Shortest Path First (OSPF)

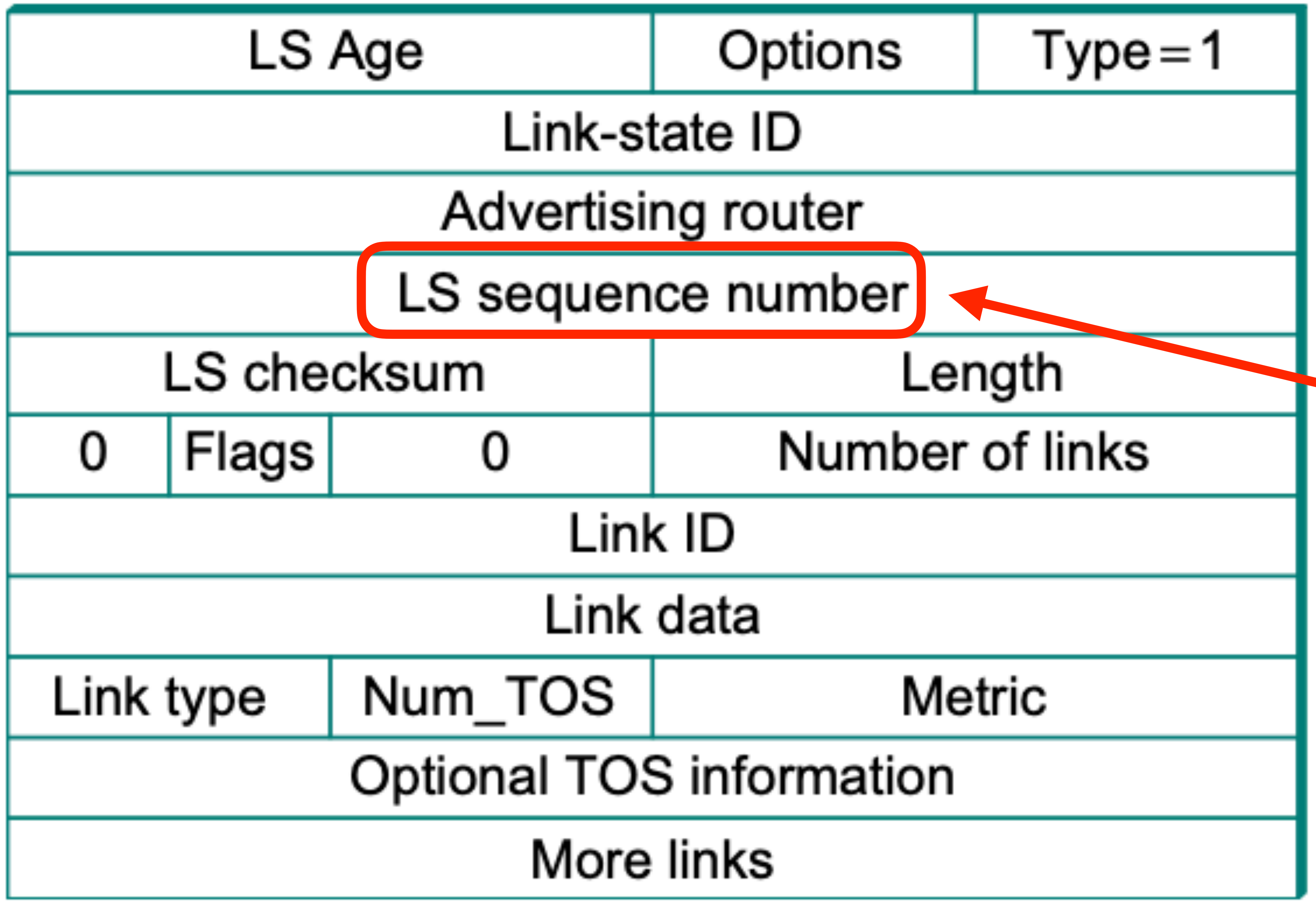
| | | | |
|---|---|---|---|
| LS Age | | Options | Type=1 |
| Link-state ID | | | |
| Advertising router | | | |
| LS sequence number | | | |
| LS checksum | | Length | |
| 0 | Flags | 0 | Number of links |
| Link ID | | | |
| Link data | | | |
| Link type | Num_TOS | Metric | |
| Optional TOS information | | | |
| More links | | | |

**OSPF link-state advertisement**

## Link state packet (LSP)

- ID of the node that created the LSP
- Cost of link to each directly connect neighbor
- Sequence number (SEQ#)
- Time-to-live (TTL) for this packet

# Open Shortest Path First (OSPF)

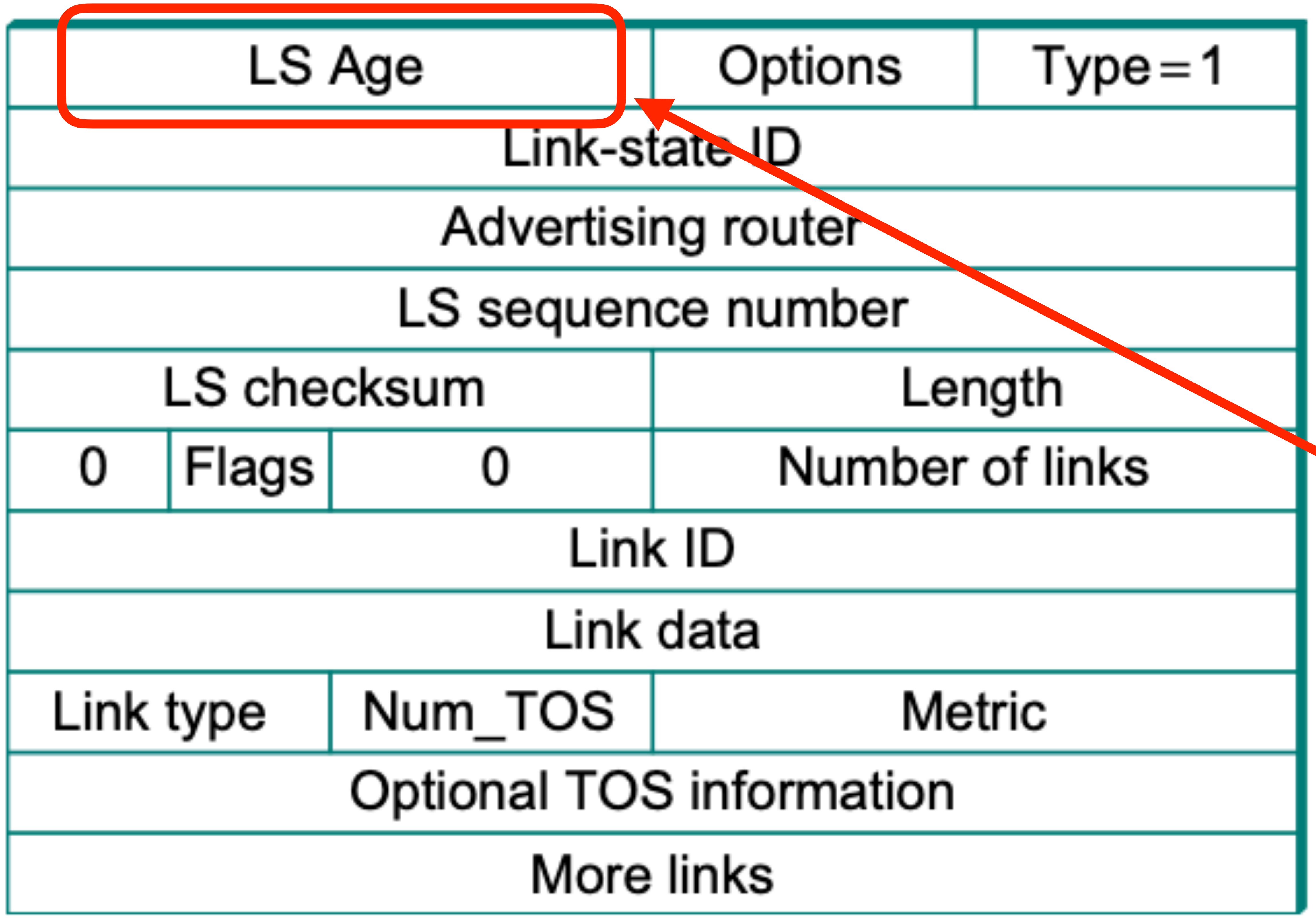



**OSPF link-state advertisement**

## Link state packet (LSP)

- ID of the node that created the LSP
- Cost of link to each directly connect neighbor
- Sequence number (SEQ#)
- Time-to-live (TTL) for this packet

# Open Shortest Path First (OSPF)




**OSPF link-state advertisement**

## Link state packet (LSP)

- ID of the node that created the LSP
- Cost of link to each directly connect neighbor
- Sequence number (SEQ#)
- Time-to-live (TTL) for this packet

# Link State v.s. Distance Vector

# Link State v.s. Distance Vector

## Link State

- High messaging overhead
- Computation complexity

## Distance Vector

- Slow convergence
- Race conditions

## Assumption of distance vector:

- Each node knows the cost of the link to each of its directly connected neighbors

## Assumption of link state:

- Each node can find out the state of the link to its neighbors and the cost of each link

# Metrics for Link Cost

## #1: assign 1 to each link

## #2: original ARPANET metric

- link cost == number of packets enqueued on each link
    - This moves packets toward the shortest queue, not the destination!!
- Take latency or bandwidth into consideration

# Metrics for Link Cost

## #3: new ARPANET metric

- link cost == average delay over some time period
- Stamp each incoming packet with its arrival time (**AT**)
- Record departure time (**DT**)
- When link-level ACK arrives, compute
    - Delay = (**DT** - **AT**) + Transmit + Latency, where transmit and Latency are static for the link
- If timeout, reset **DT** to departure time for retransmission

# Goals in Router/Switch Design

## #1: Throughput

- Ability to forward as many packets per second as possible

## #2: Size

- Number of input/output ports

## #3: Cost

- Minimum cost per port

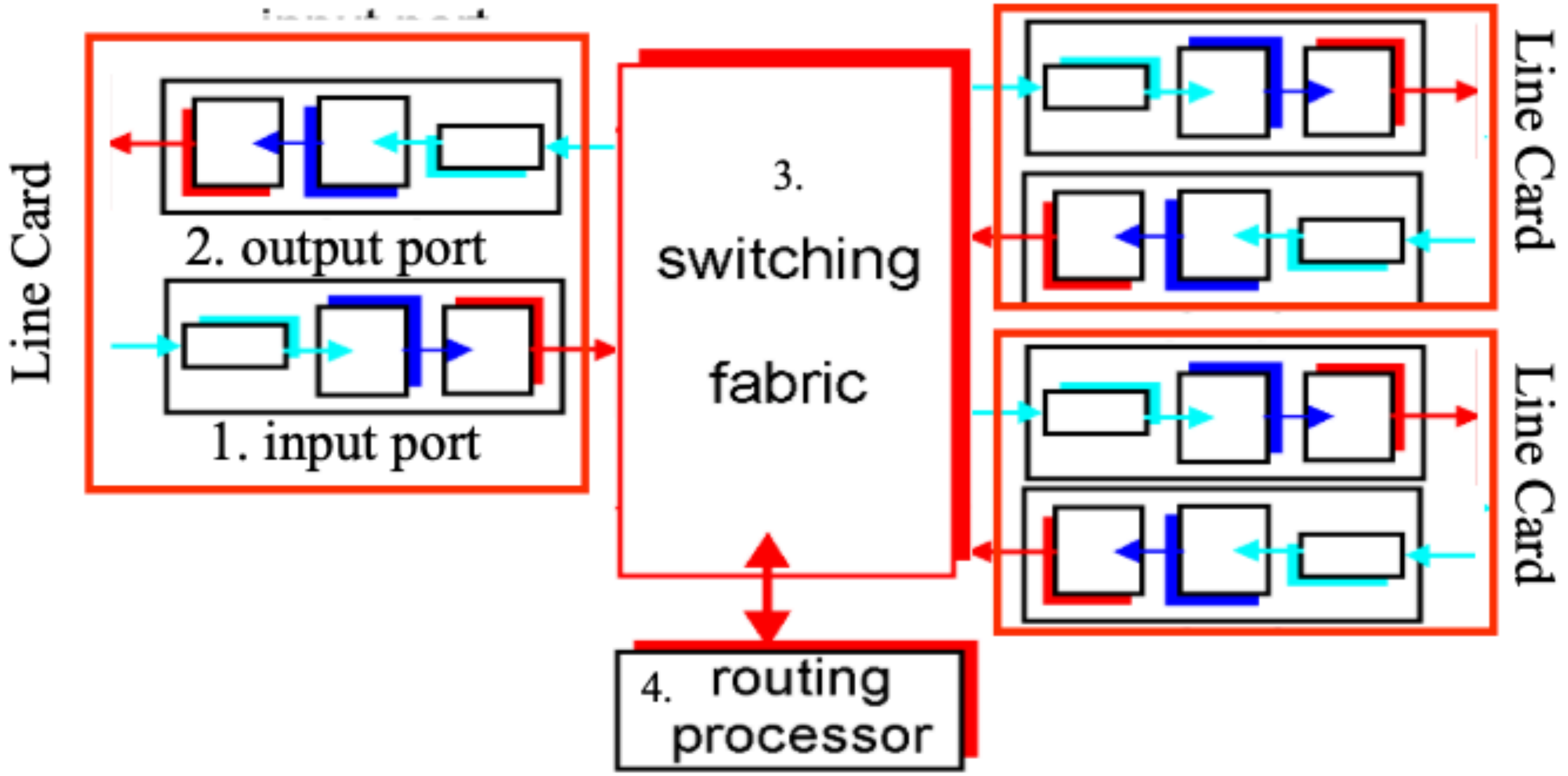
## #4: Functionality

- Forwarding, routing, quality of service (QoS), …

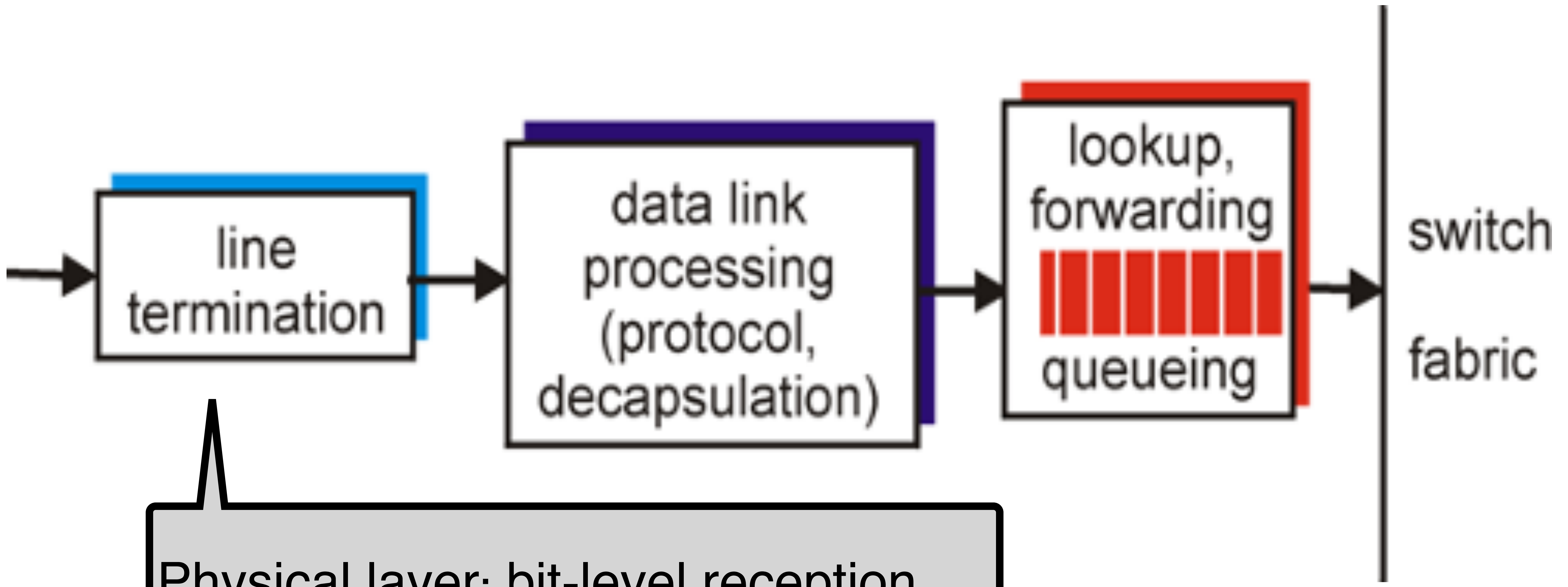
# Router Architecture Overview

## Two key router functions:

- Run routing algorithms/protocol (RIP, OSPF, BGP, etc.)
- Switching datagrams from incoming to outgoing links

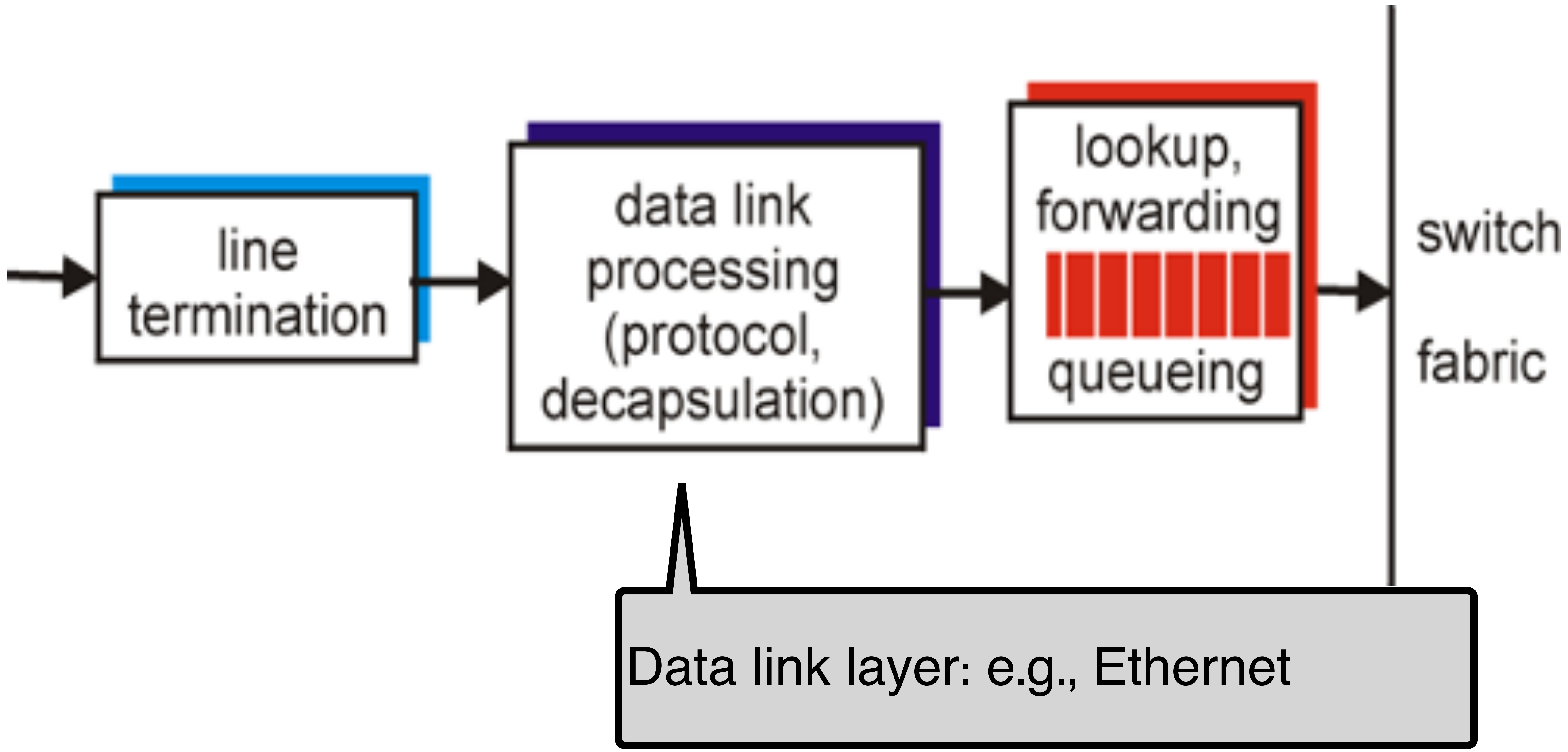


# Line Card: Input Port

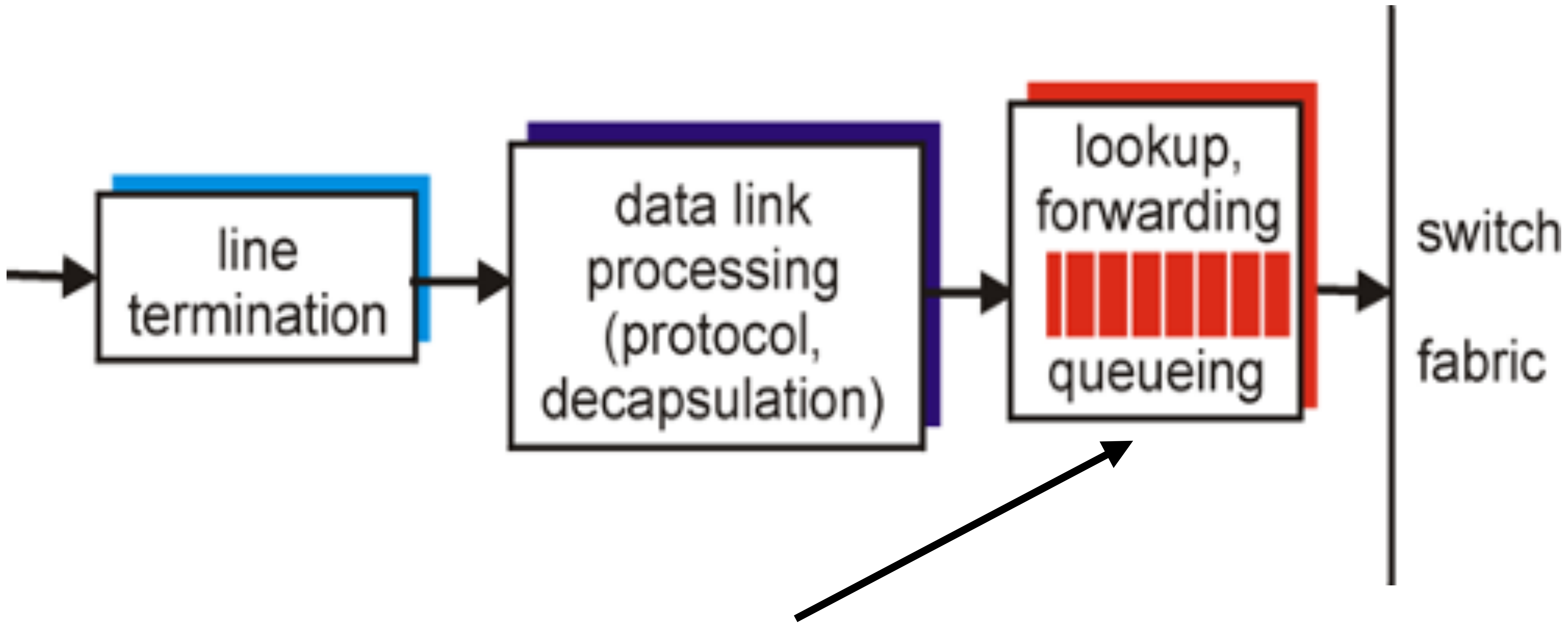



Physical layer: bit-level reception

# Line Card: Input Port



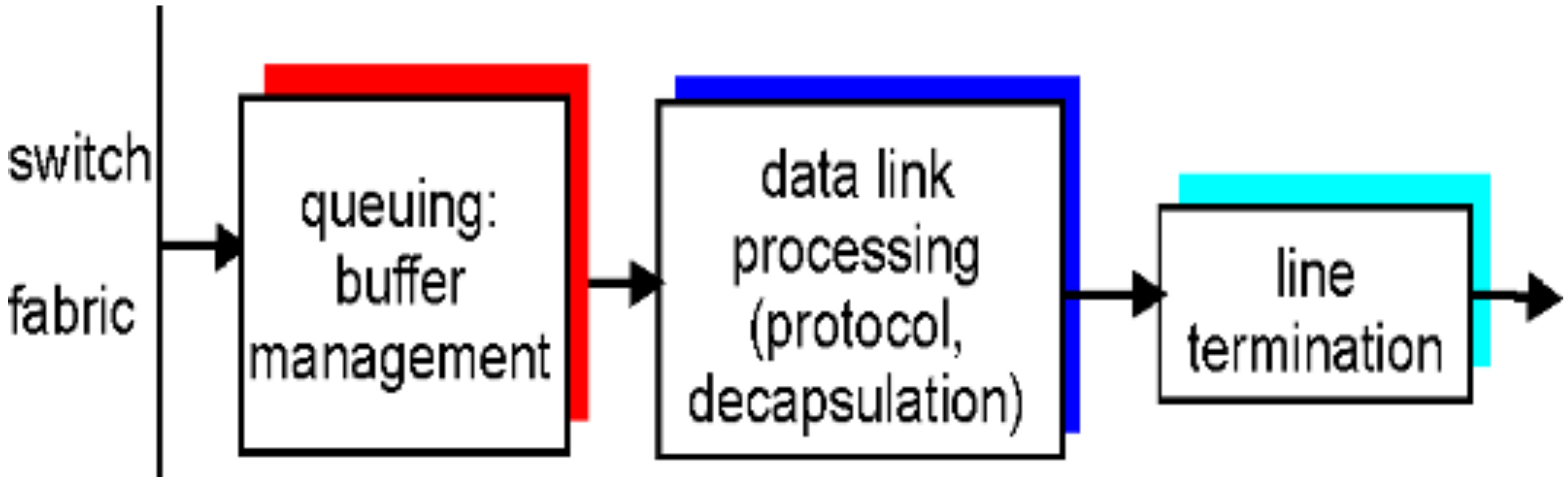
# Line Card: Input Port




## Decentralized switching:

- Process common case ("fast-path"), e.g., decrement TTL, update the checksum
- Lookup output port based on routing table in input port memory
- Queue needed if datagrams arrive faster than forwarding rate into switch fabric

# Line Card: Output Port




**Queueing required when datagrams arrive from fabric faster than the line transmission rate**

# Buffering

## 3 types of buffering

- Input buffering
  - Fabric slower than input ports combined -> queueing may occur at input queues
- Output buffering
  - Buffering when arrival rate via switch exceeds output line speed
- Internal buffering
  - Can have buffer inside switch fabric to deal with limitations of fabric

## What happens when these buffers fill up?

- Packets are **thrown away**!! This is where (most) packet loss comes from

# Routing(Network) Processor

## Run routing protocol and push forwarding table to forward engines

## Perform “slow” path processing

- ICMP error message
- IP option processing
- Fragmentation
- Packets destined to router

# IP Router v.s. Ethernet Switch (Incomplete!)

| | IP Router | Ethernet Switch |
|---|---|---|
| **Layering** | Layer 3 | Layer 2 |
| **Packet Manipulation** | Fragmentation and Reassembly; TTL update | N/A |
| **Packet Forwarding** | Based on the destination IP address | Based on destination Ethernet address; Run the spanning tree protocol to avoid forwarding loops |
| **Routing** | Based on the routing algorithm | N/A |
| **Error Handling** | Speak the ICMP protocol | N/A |

## Terminology

1. Host
2. NIC
3. Multi-port I/O bridge
4. Protocol
5. RTT
6. Packet
7. Header
8. Payload
9. BDP
10. Baud rate
11. Frame/Framing
12. Parity bit
13. Checksum
14. Ethernet
15. MAC
16. (L2) Switch
17. Broadcast
18. Acknowledgement
19. Timeout
20. Datagram
21. TTL
22. MTU
23. Best effort
24. (L3) Router
25. Subnet mask
26. CIDR
27. Converge
28. Count-to-infinity
29. Line card
30. Network processor

## Principle

1. Layering
2. Minimal States
3. Hierarchy

## Technique

1. NRZ Encoding
2. NRZI Encoding
3. Manchester Encoding
4. 4B/5B Encoding
5. Byte Stuffing
6. Byte Counting
7. Bit Stuffing
8. 2-D Parity
9. CRC
10. MAC Learning
11. Store-and-Forward
12. Cut-through
13. Spanning Tree
14. CSMA/CD
15. Stop-and-Wait
16. Sliding Window
16. Fragmentation and Reassembly
17. Path MTU discovery
18. DHCP
19. Subnetting
20. Supernetting
21. Longest prefix match
22. Distance vector routing (RIP)
23. Link state routing (OSPF)

# Summary

## Today's takeaways

#1: Link state routing captures the whole network connectivity by disseminating the link state information and runs the Dijkstra's algorithm to calculate the shortest path

#2: Link cost can be determined by performance metrics

#3: A router has four major components: input line card, output line card, switching fabric and network processor

## Next lecture

- Inter-domain Routing