Introduction to Computer Networks

CS640

Link State Routing

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

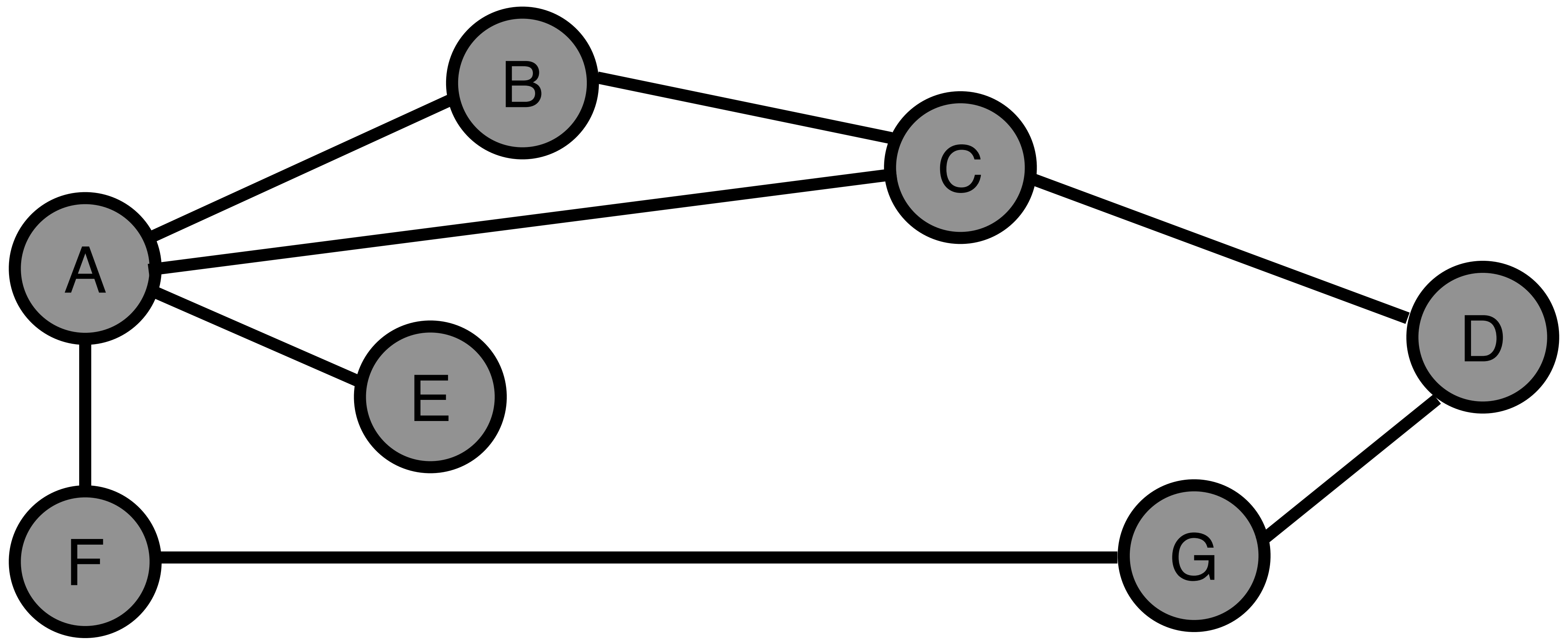
- Last
  - Distance Vector Routing
- Today
  - Link State Routing
- Announcements
  - Quiz2 in-class this Thursday (03/05/2026)

Recap: Distance Vector Routing

- Key idea:
  - Each router constructs a one-dimensional array (vector) that contains the “distance” (cost) to all other nodes
  - Distributes that vector to its immediate neighbors

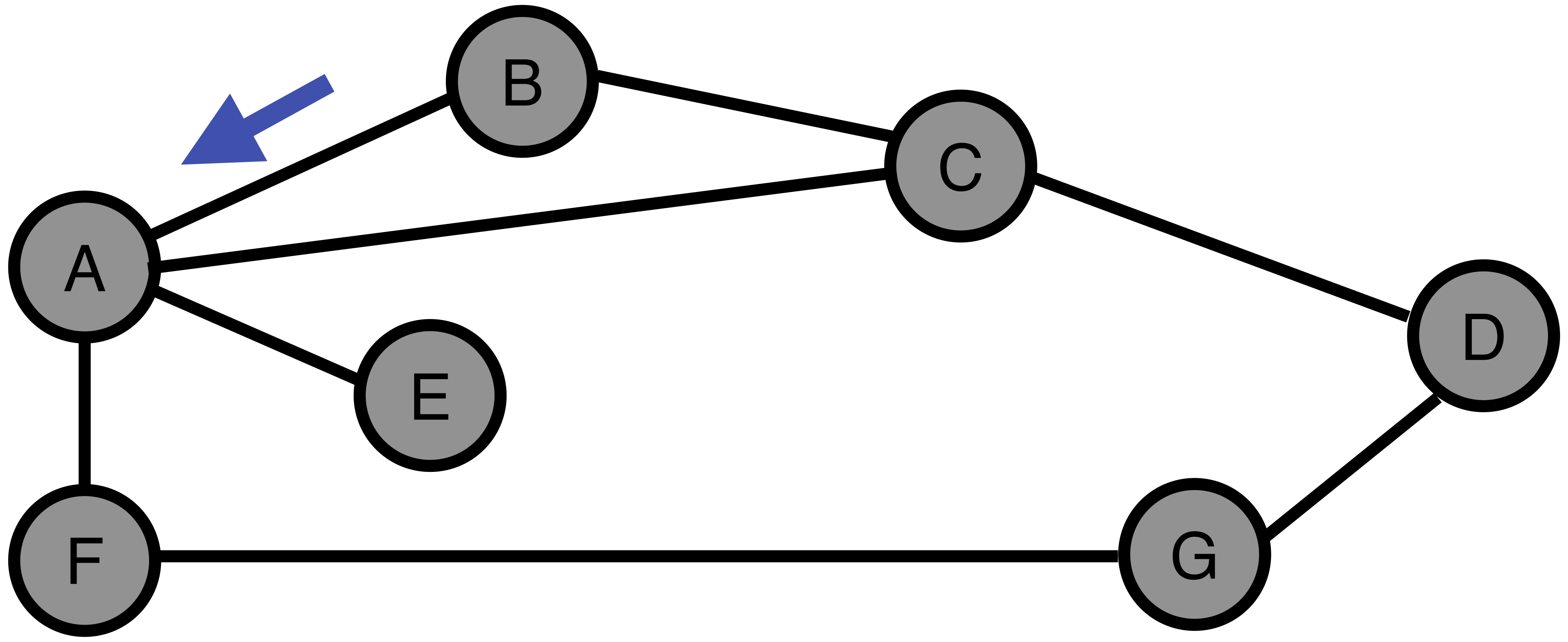
- Assumption
  - Each router knows the cost of the link to its directly connected neghbors

Step 1: Figure Out Initial Distance



	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Step 2: Exchange the Distance Vector



A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	∞	–
E	1	E
F	1	F
G	∞	–

+

B

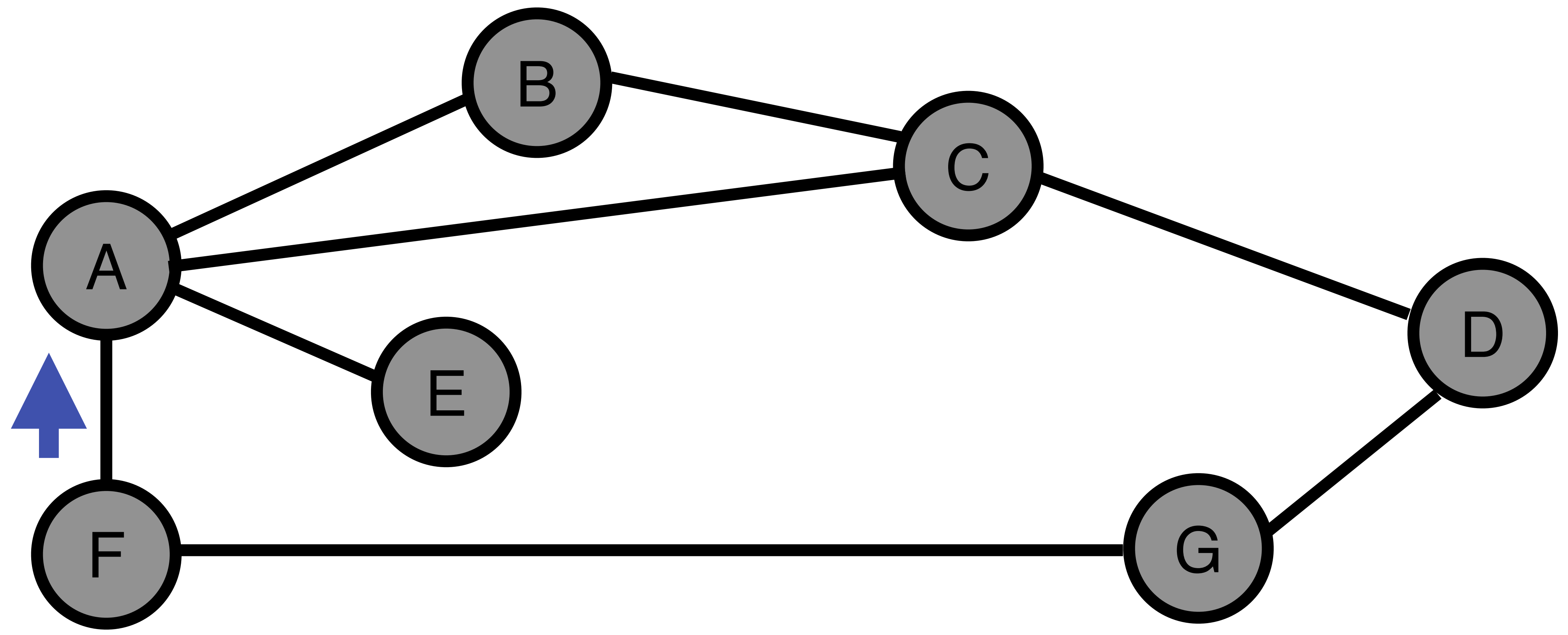
Dest.	Cost	NextHop
A	1	A
C	1	C
D	∞	–
E	∞	–
F	∞	–
G	∞	–

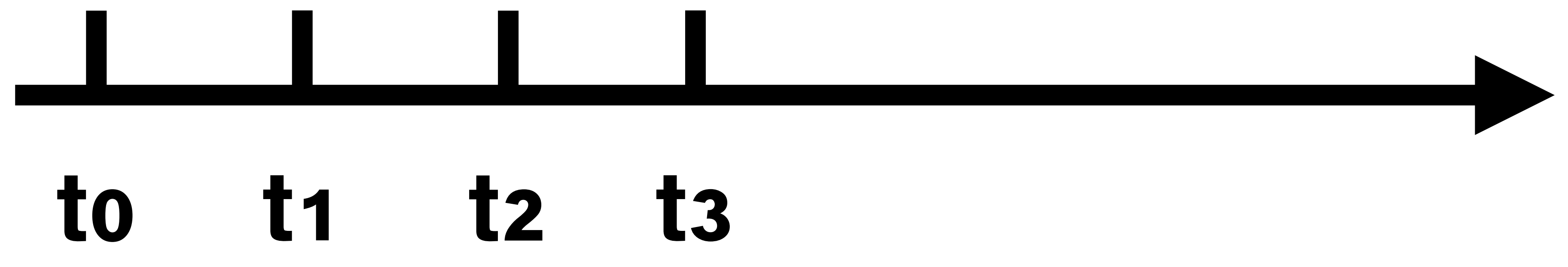
=

A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	∞	–
E	1	E
F	1	F
G	∞	–

Step 3+: Keep Exchange Vectors Until Stable





A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	∞	–

+

F

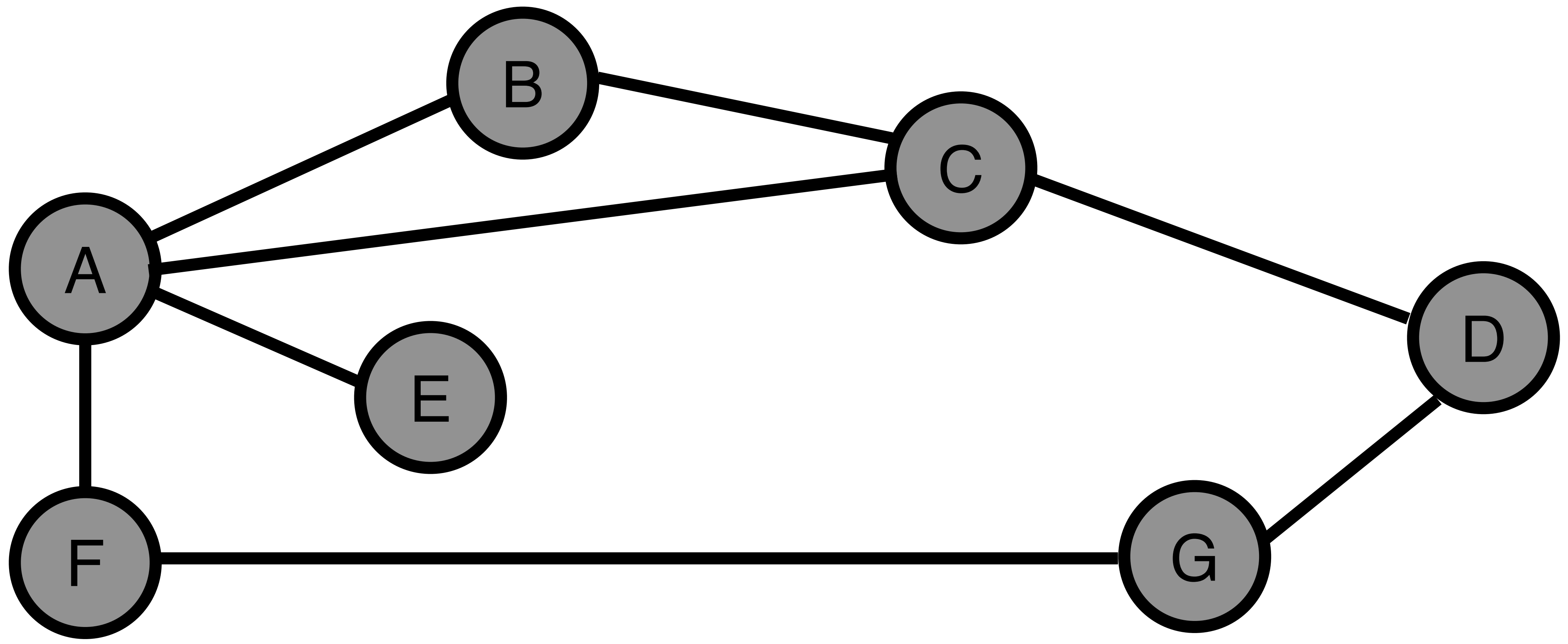
Dest.	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
F	0	F
G	1	G

=

A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

A Temporary Stable Distance Table



	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distance Vector Discussion

- Distance vector routing is based on the Bellman-Ford algorithm
  - Compute shortest paths from a single source vertex to all of the other vertices in a weighted directed graph
- Each router sends its distance vector to its neighbors periodically
- Each router then update its table based on the new vector

Distance Vector Discussion

- Distance vector routing is based on the Bellman-Ford algorithm
  - Compute shortest paths from a single source vertex to all of the other vertices in a weighted directed graph

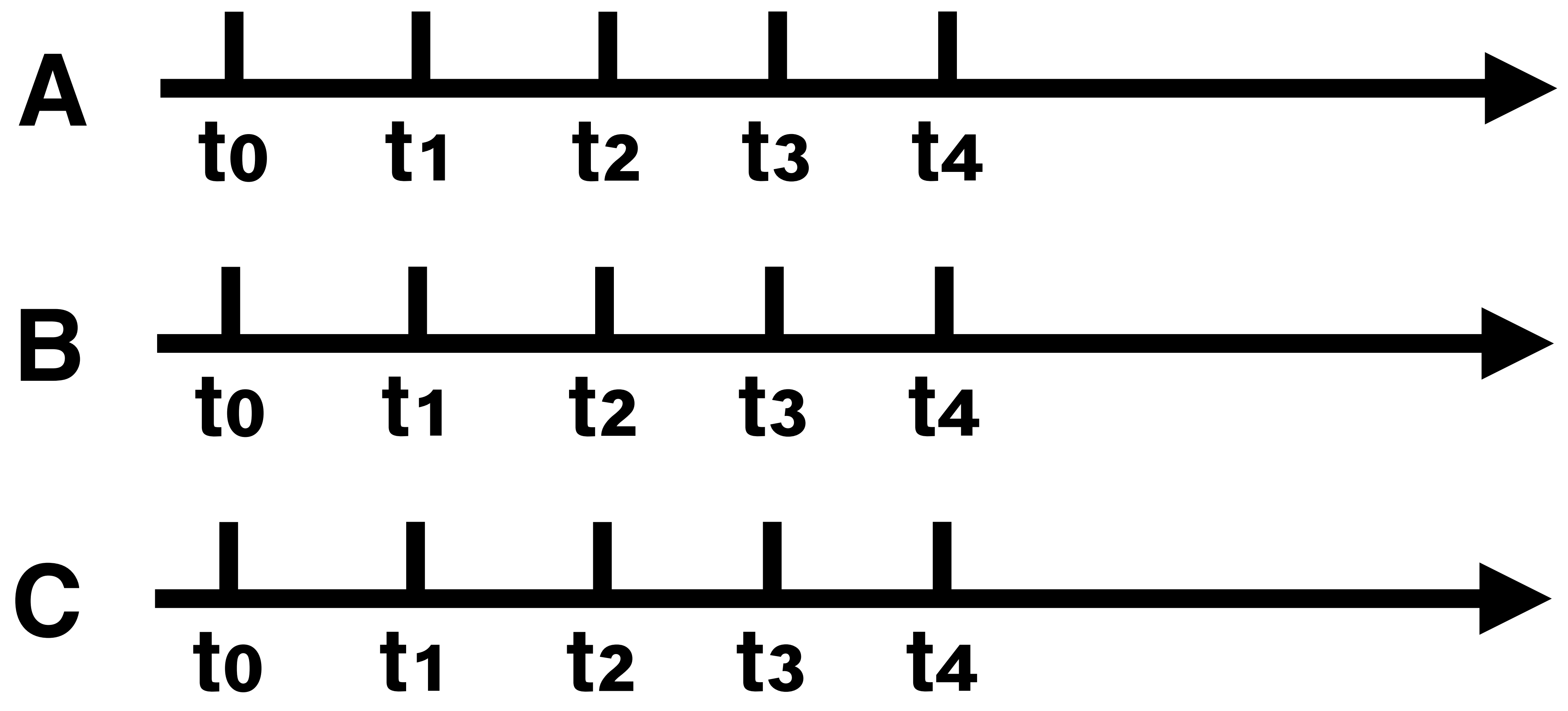
Advantage

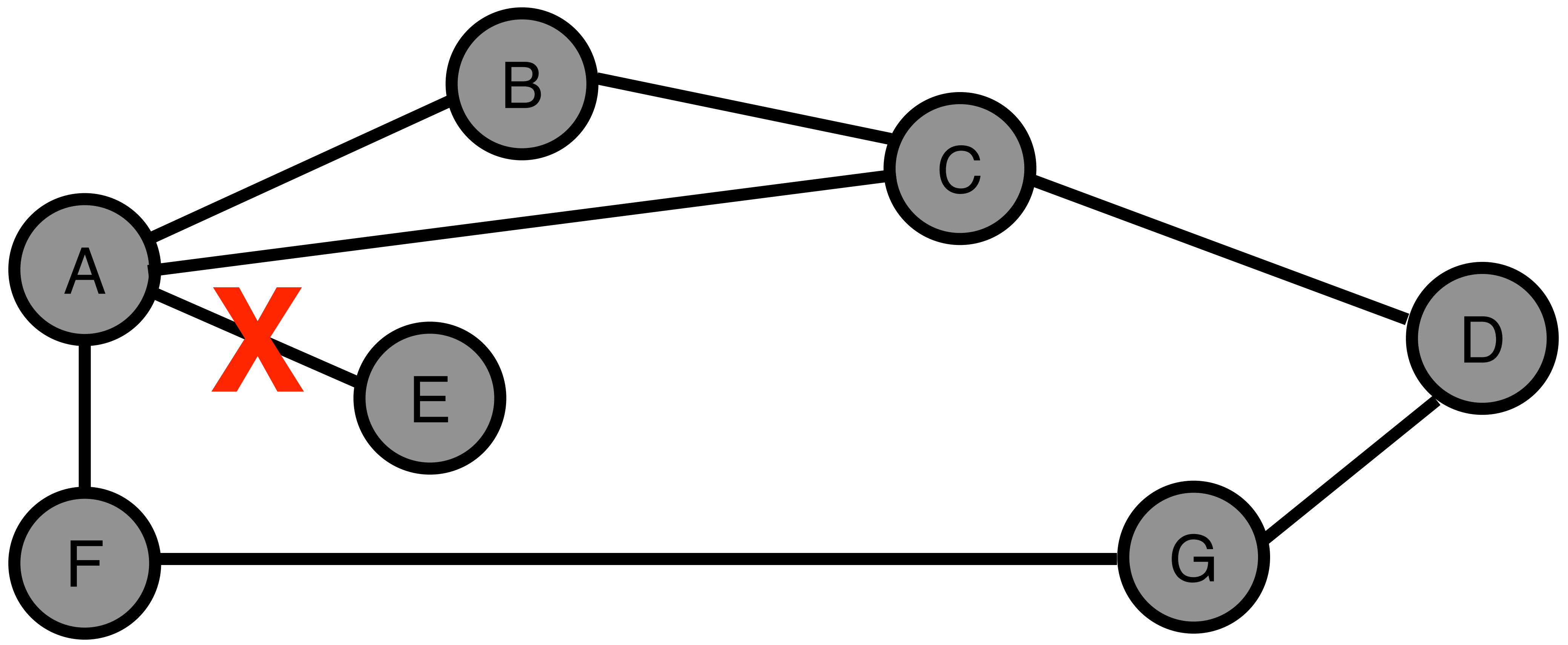
- Fast response to the good news

Disadvantage

- Slow response to the bad news

A Slow Converging Example

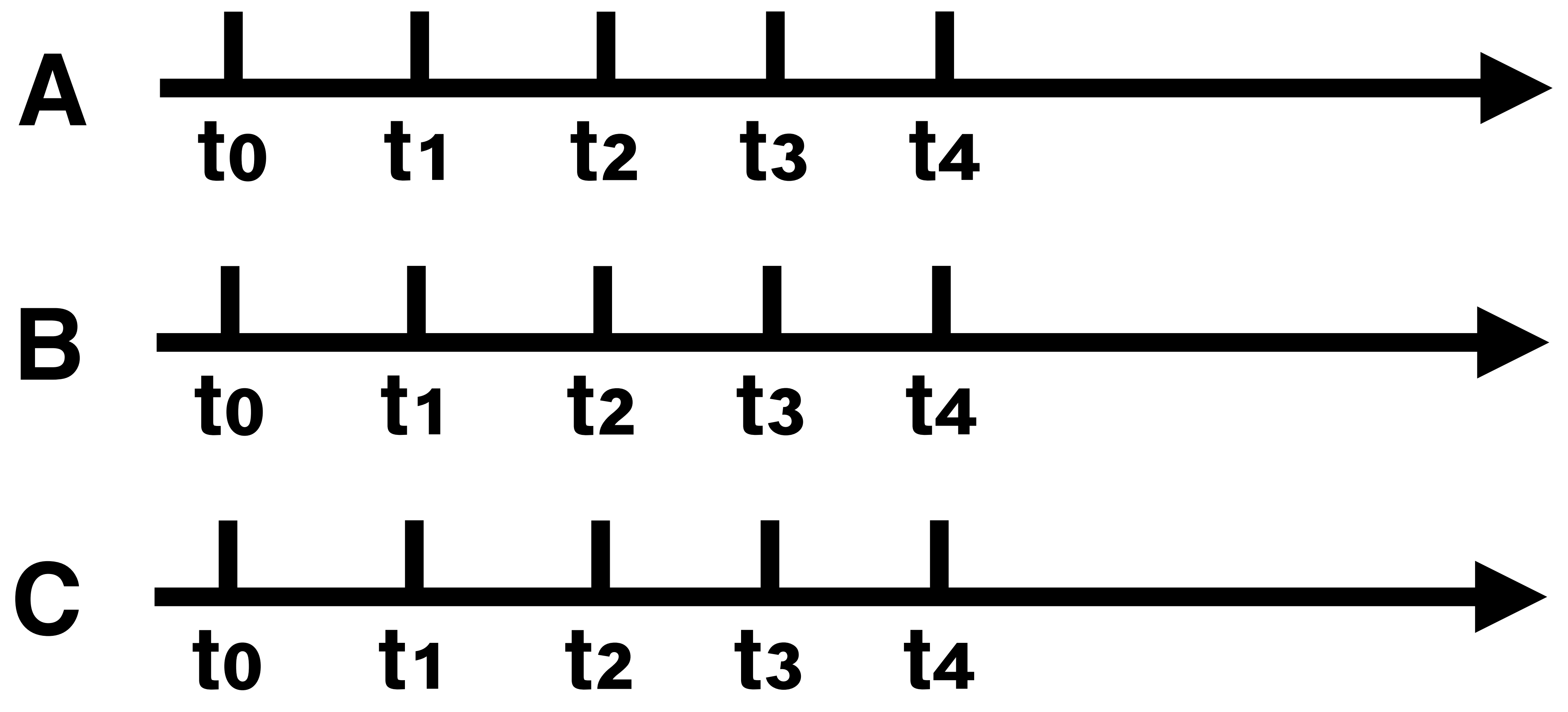


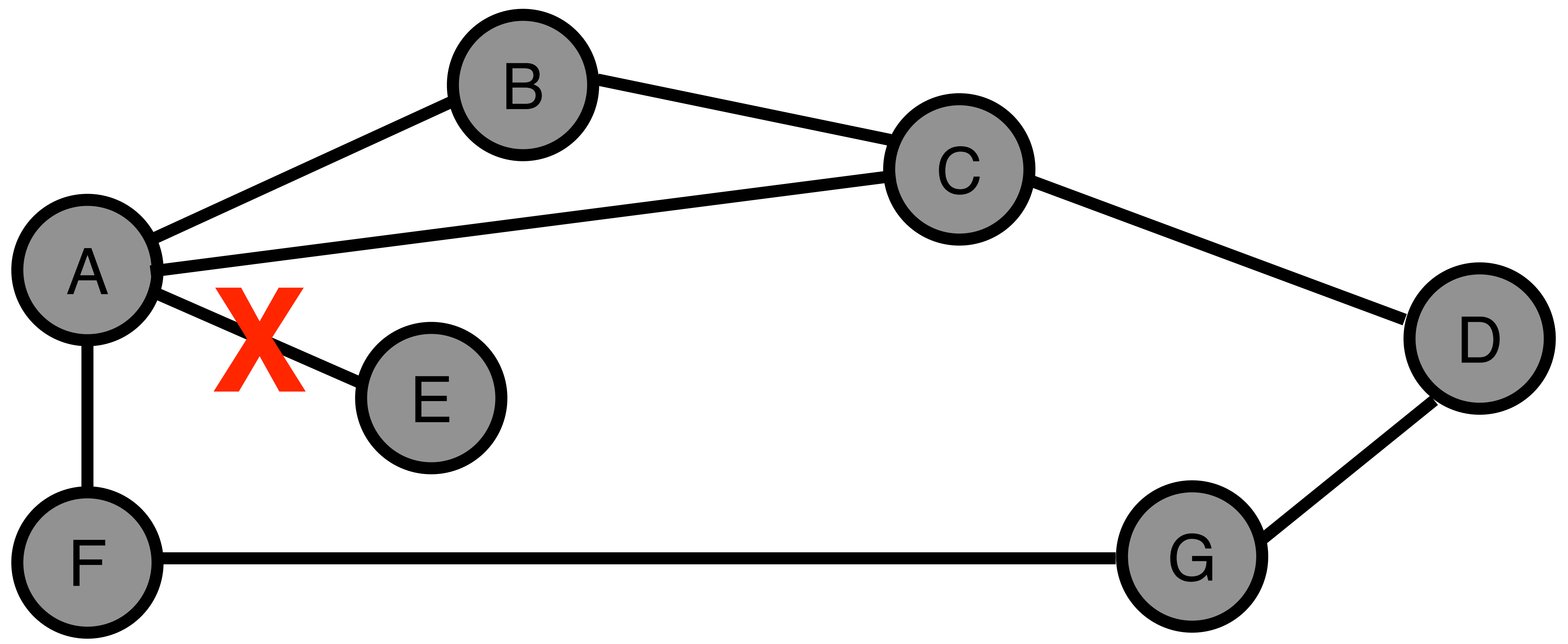


- At **t4**, C receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**
- At **t4**, A receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**
- **A will advertise this new changes to C, then C advertises B, B advertises A, …**

☹

A Slow Converging Example





This cycle stops only when the distances reach some threshold that is large enough to be considered infinite

- **This is called the count-to-infinity problem**

How does the link state routing address the issues of the distance vector routing?

Link State Routing

- Key idea:
  - Send all nodes (not just neighbors) information about the communication cost of direct-connected links (not the entire routing table)
  - Each node has complete information about the whole network
  - Find the shortest path between two nodes of the network

Link State Routing

- Key idea:
  - Send all nodes (not just neighbors) information about the communication cost of direct-connected links (not the entire routing table)
  - Each node has complete information about the whole network
  - Find the shortest path between two nodes of the network

- Advantage:
  - Converge quickly under static conditions

Two Steps

- Step #1: Reliable flooding

- Step #2: Route calculation

Step #1: Reliable Flooding

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

What is the link-state information?

Step #1: Reliable Flooding

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

What is the link-state information?

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

Link State Packet (LSP): Sequence Number

- Goal: identify the latest link cost

Link State Packet (LSP): Sequence Number

- Goal: identify the latest link cost

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
```

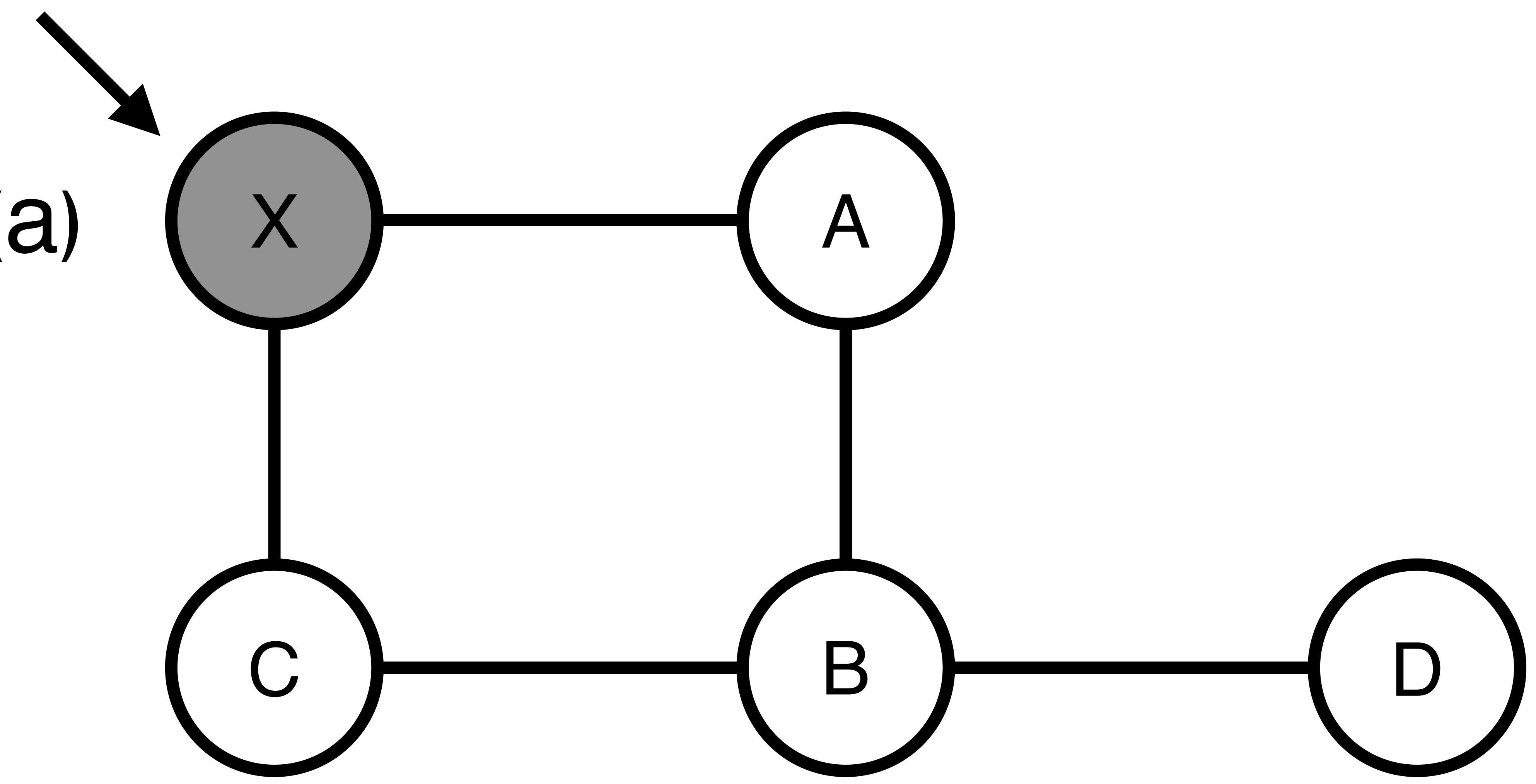
Link State Packet (LSP): Time-To-Live (TTL)

- Decrement the TTL field when storing the LSP

- Discard the LSP when its TTL becomes 0

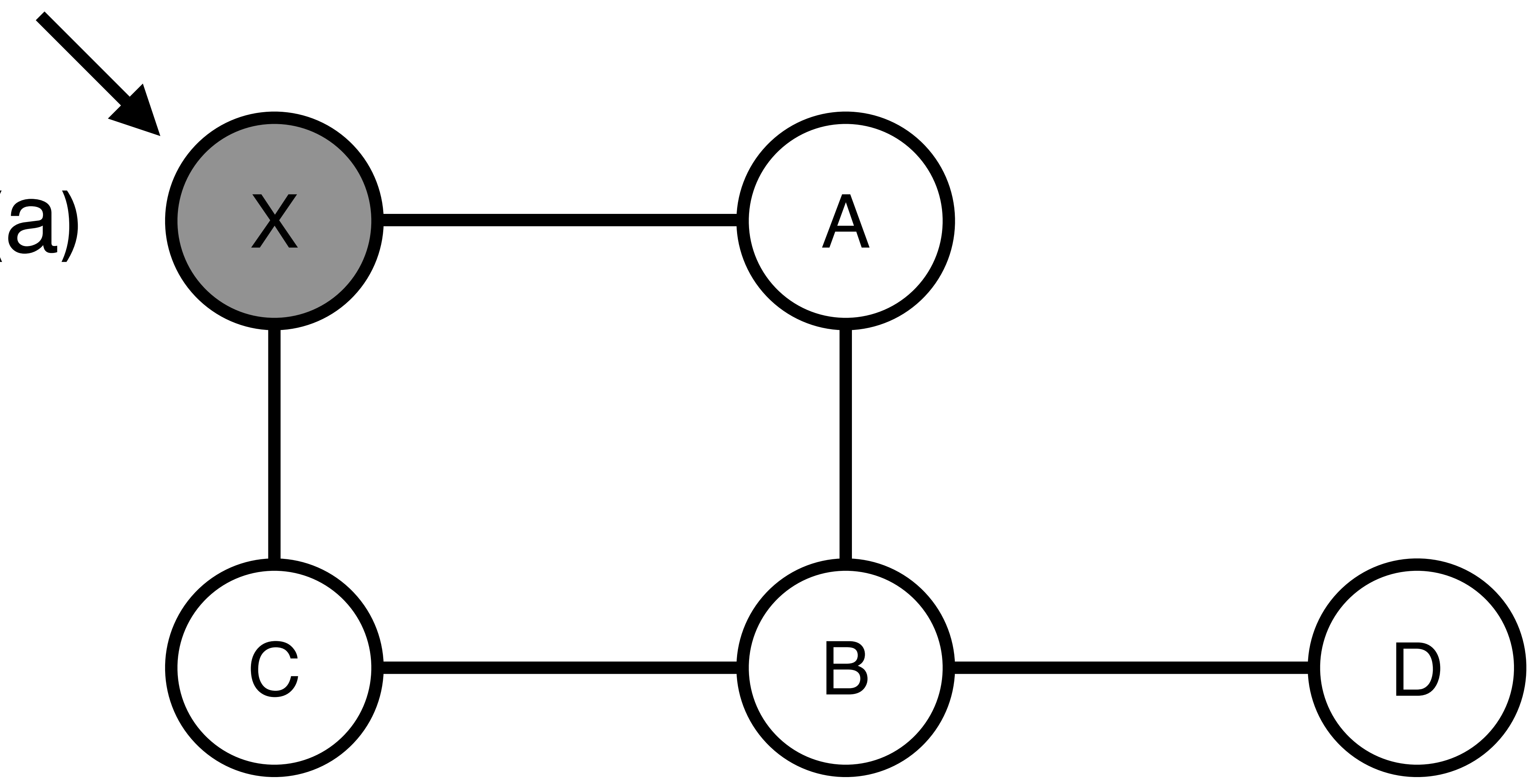
A Flooding Example

(a)

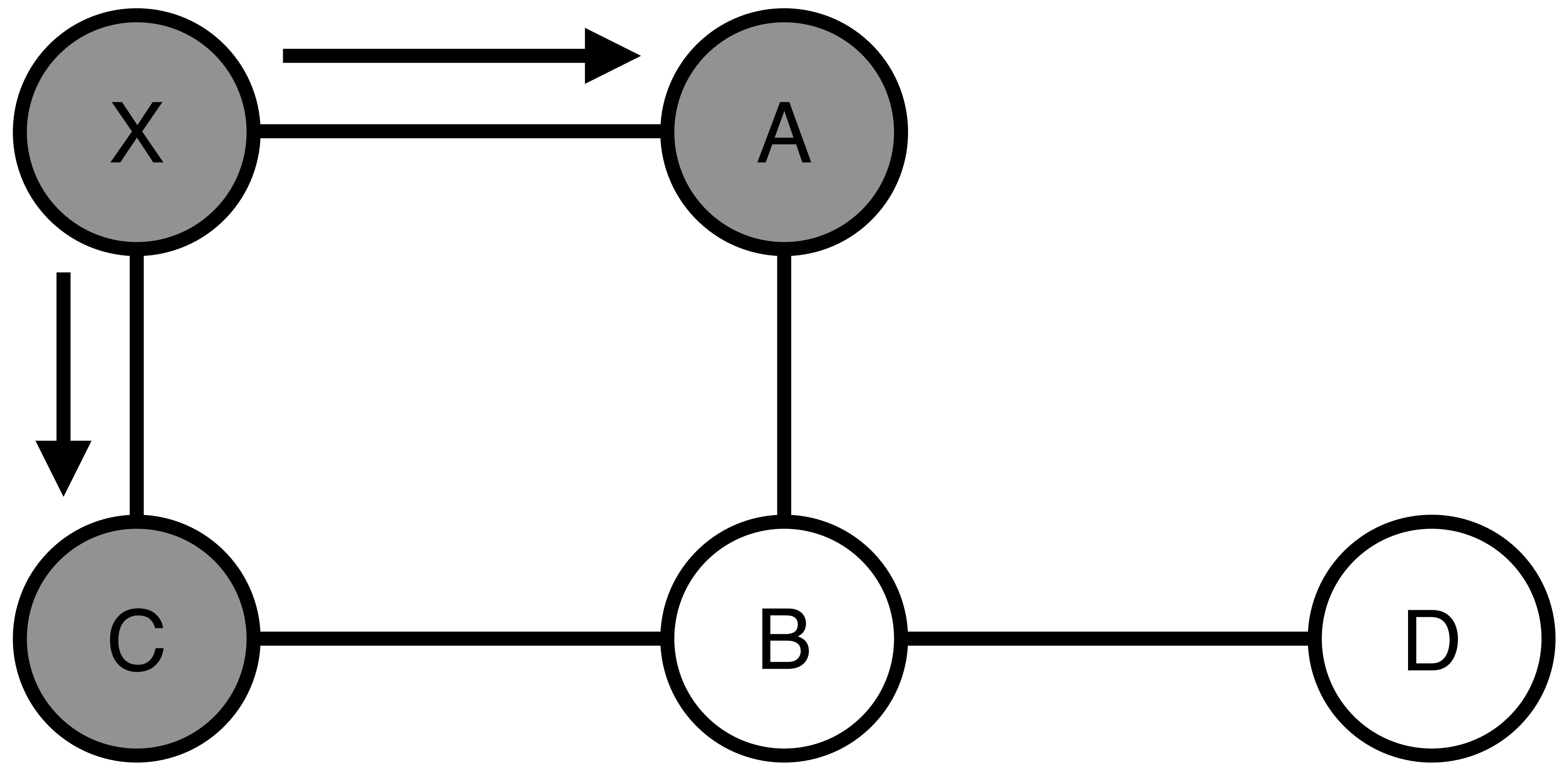


A Flooding Example

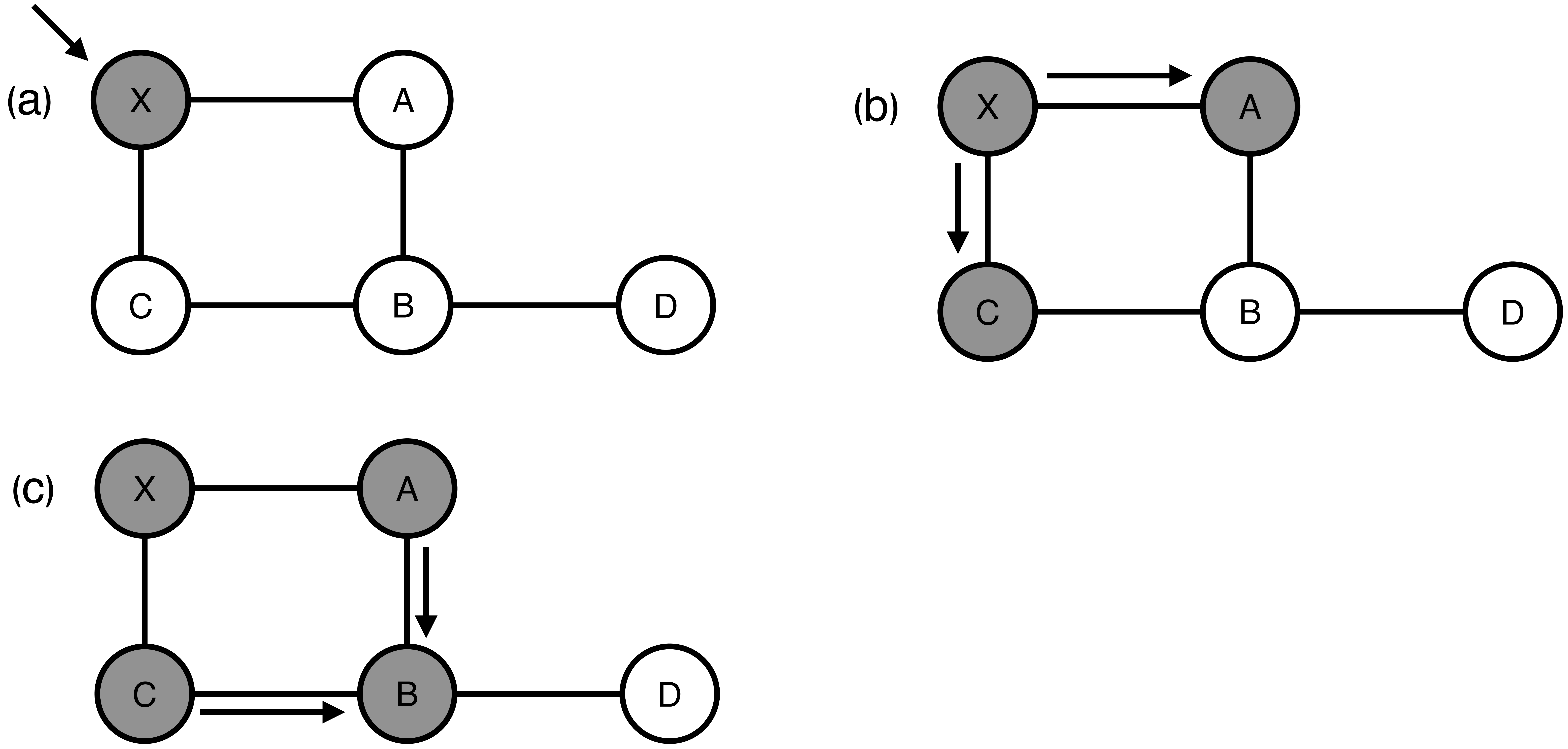
(a)



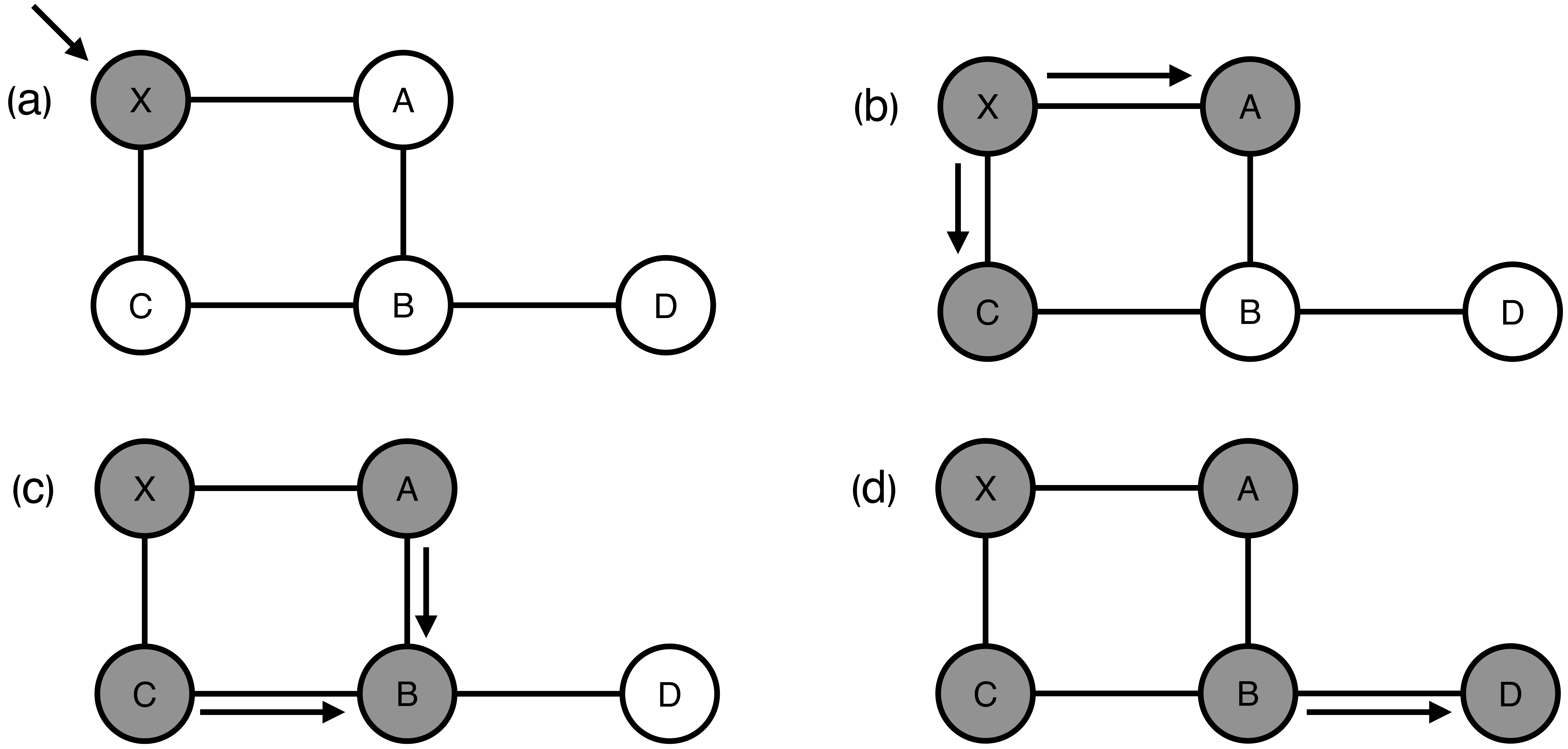
(b)



A Flooding Example



A Flooding Example

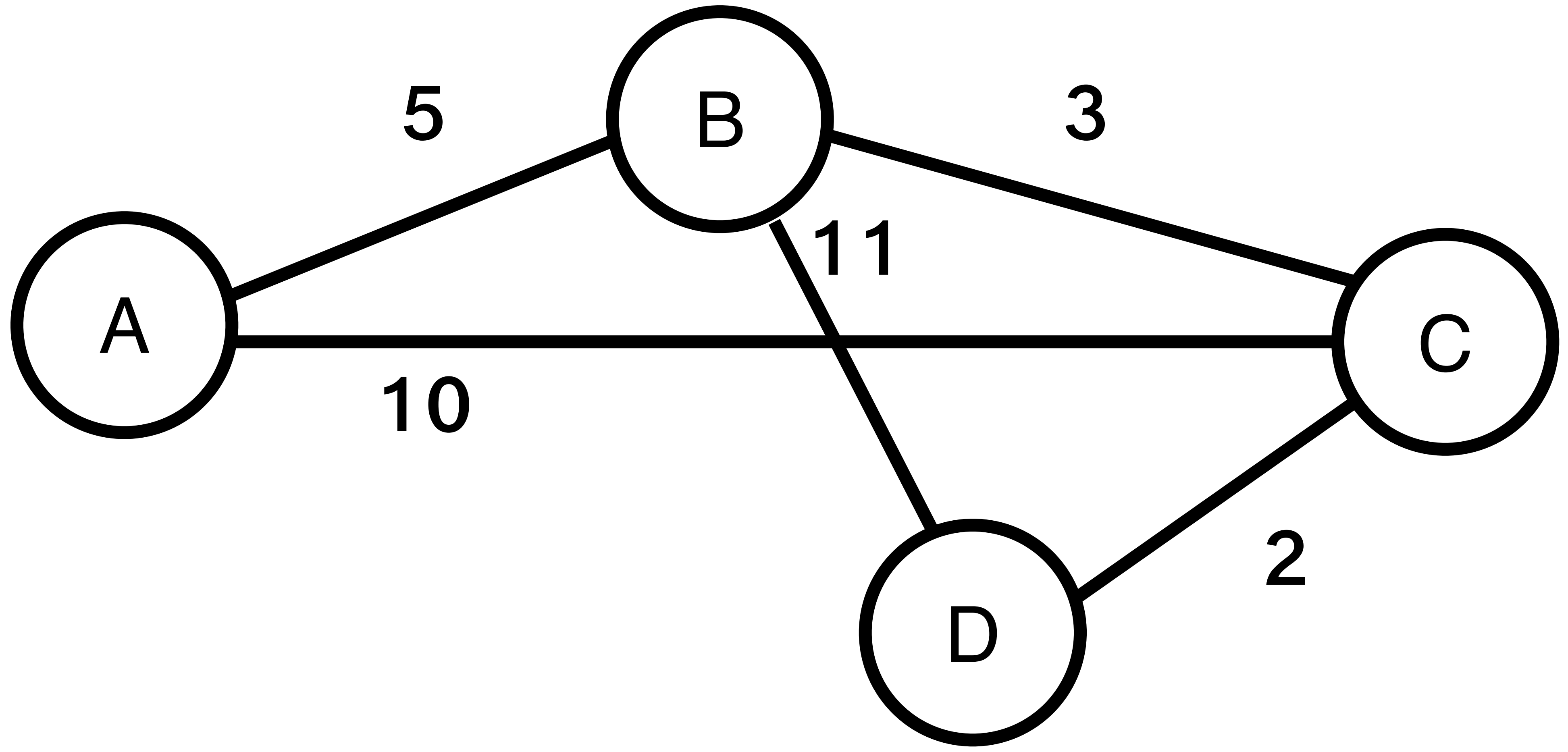


Link State Routing: Two Steps

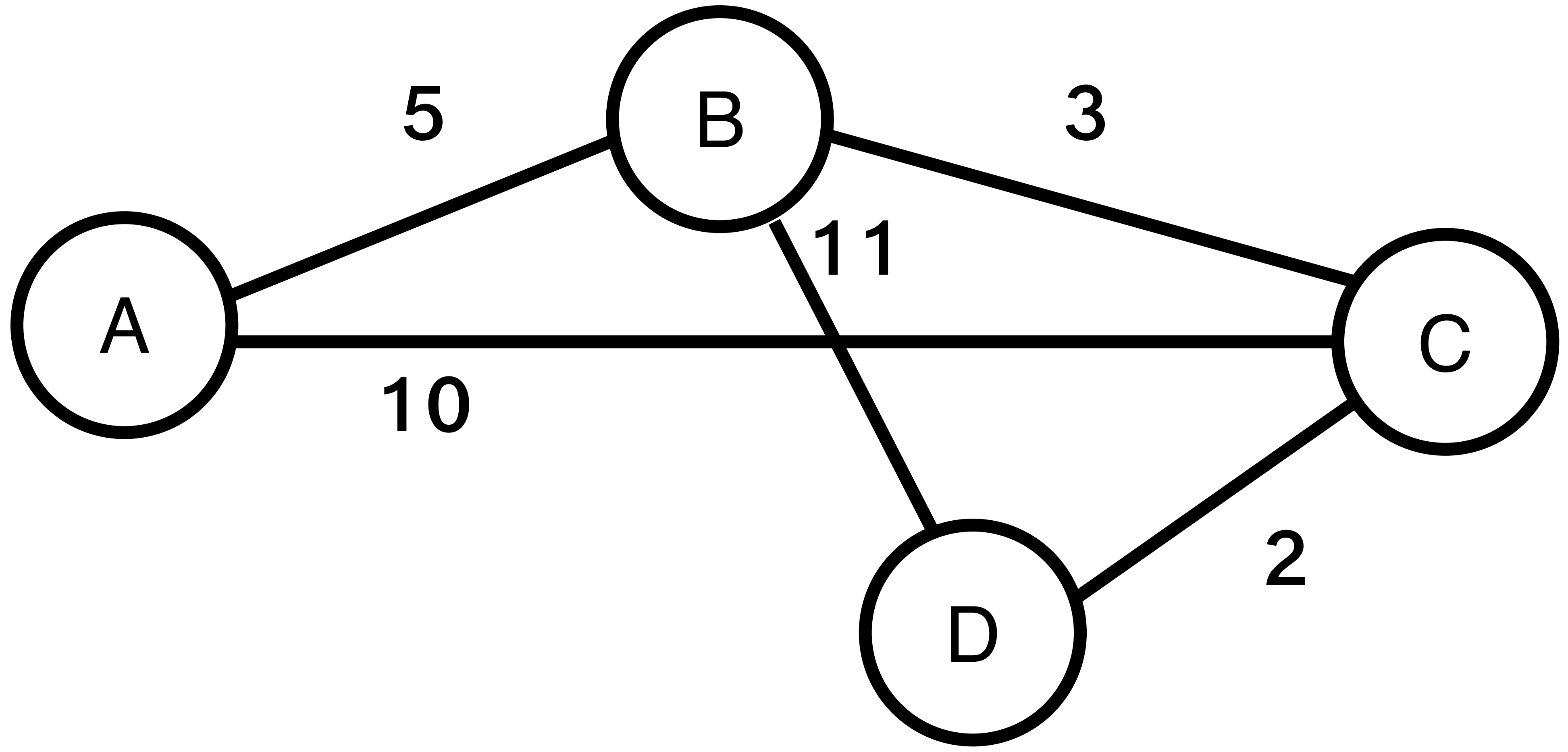
- **Step #1: Reliable flooding**
  - **Each node maintains a global view of the network**

- Step #2: Route calculation

LSP Table

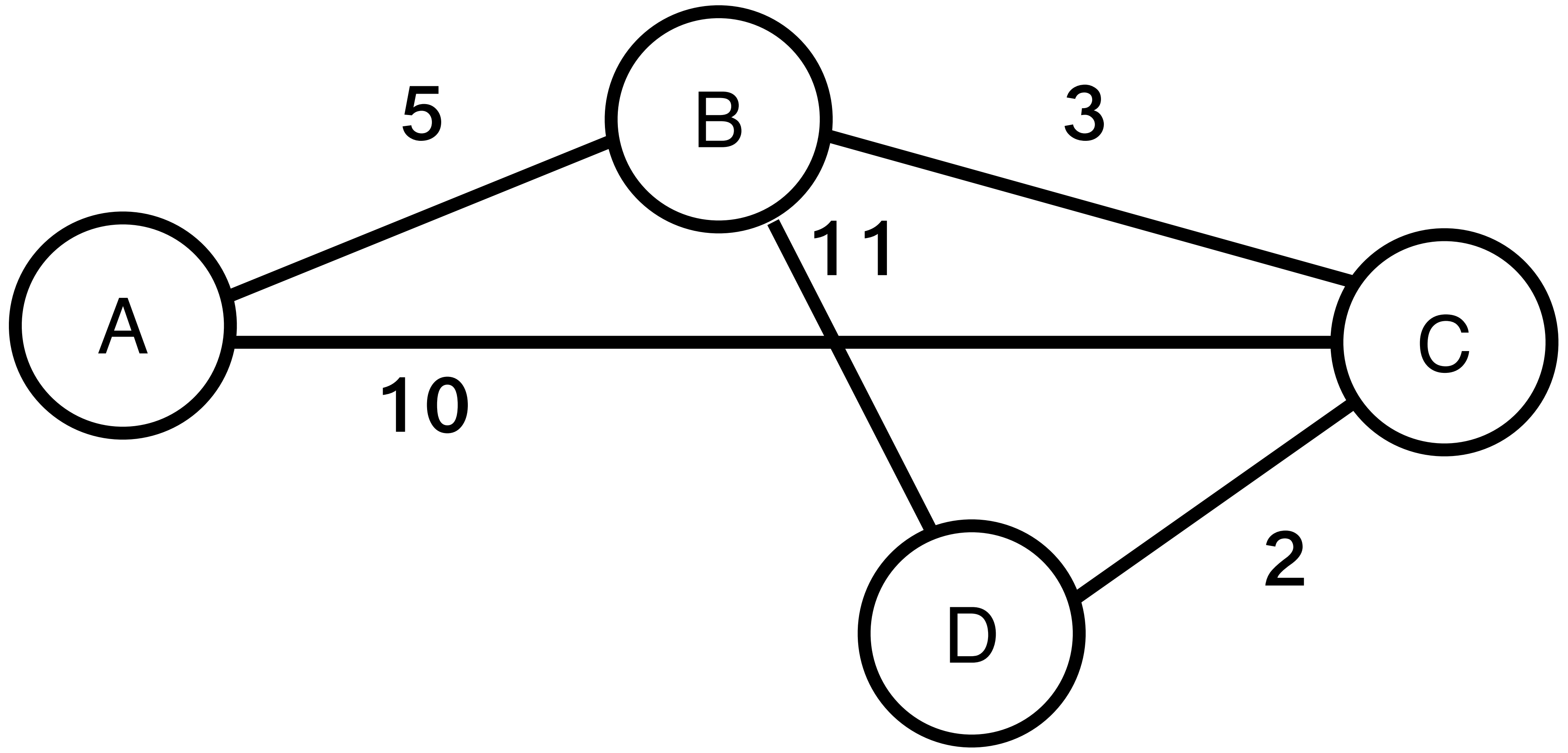


LSP Table



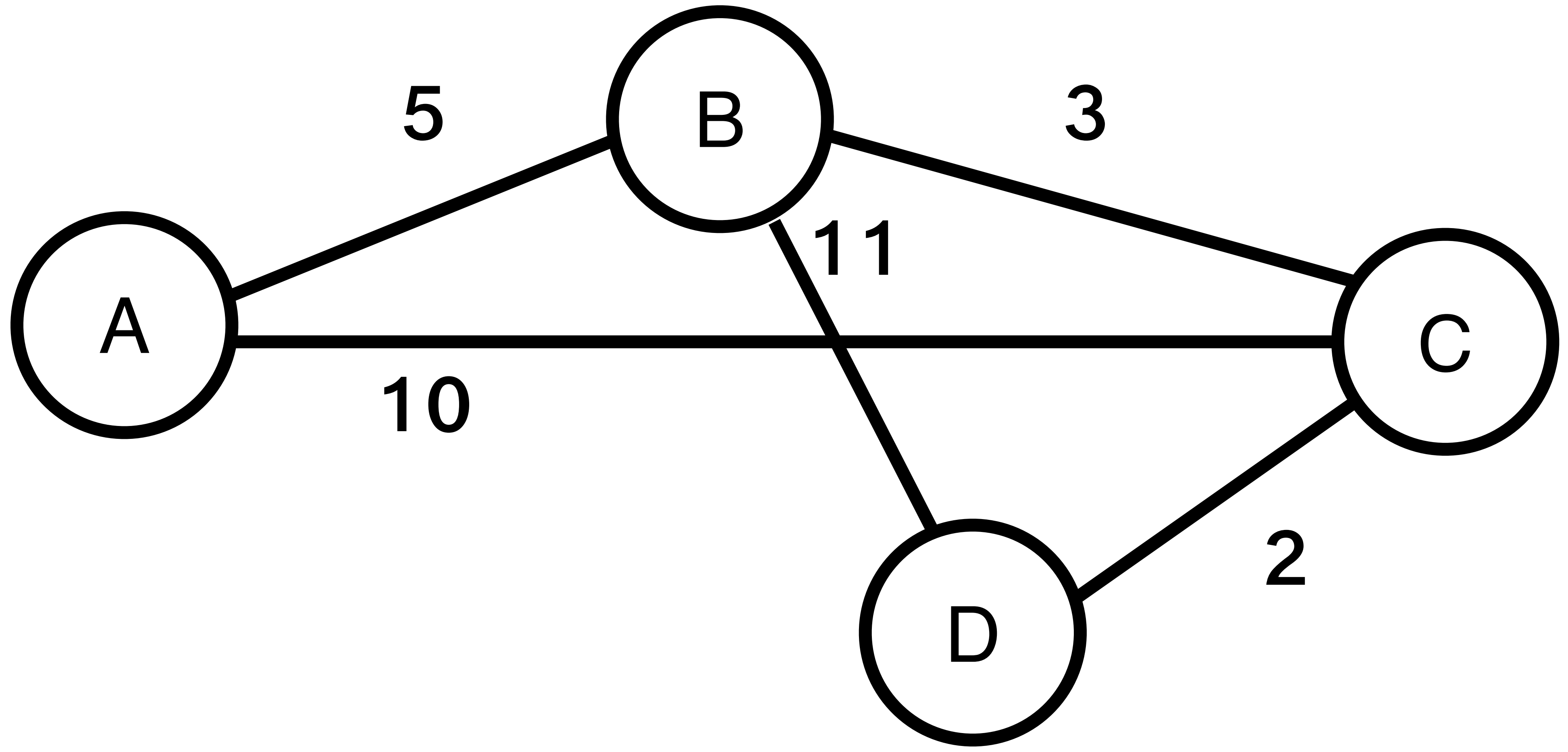
Router A Info.	ID	Link Costs	SEQ#	TTL
A LSP	A	[A, B] = 5, [A, C] = 10	1	64
B LSP	B	[B, A] = 5, [B, C] = 3, [B, D] = 11	1	63
C LSP	C	[C, A] = 10, [C, B] = 3, [C, D] = 2	1	63
D LSP	D	[D, B] = 11, [D, C] = 2	1	62

LSP Table



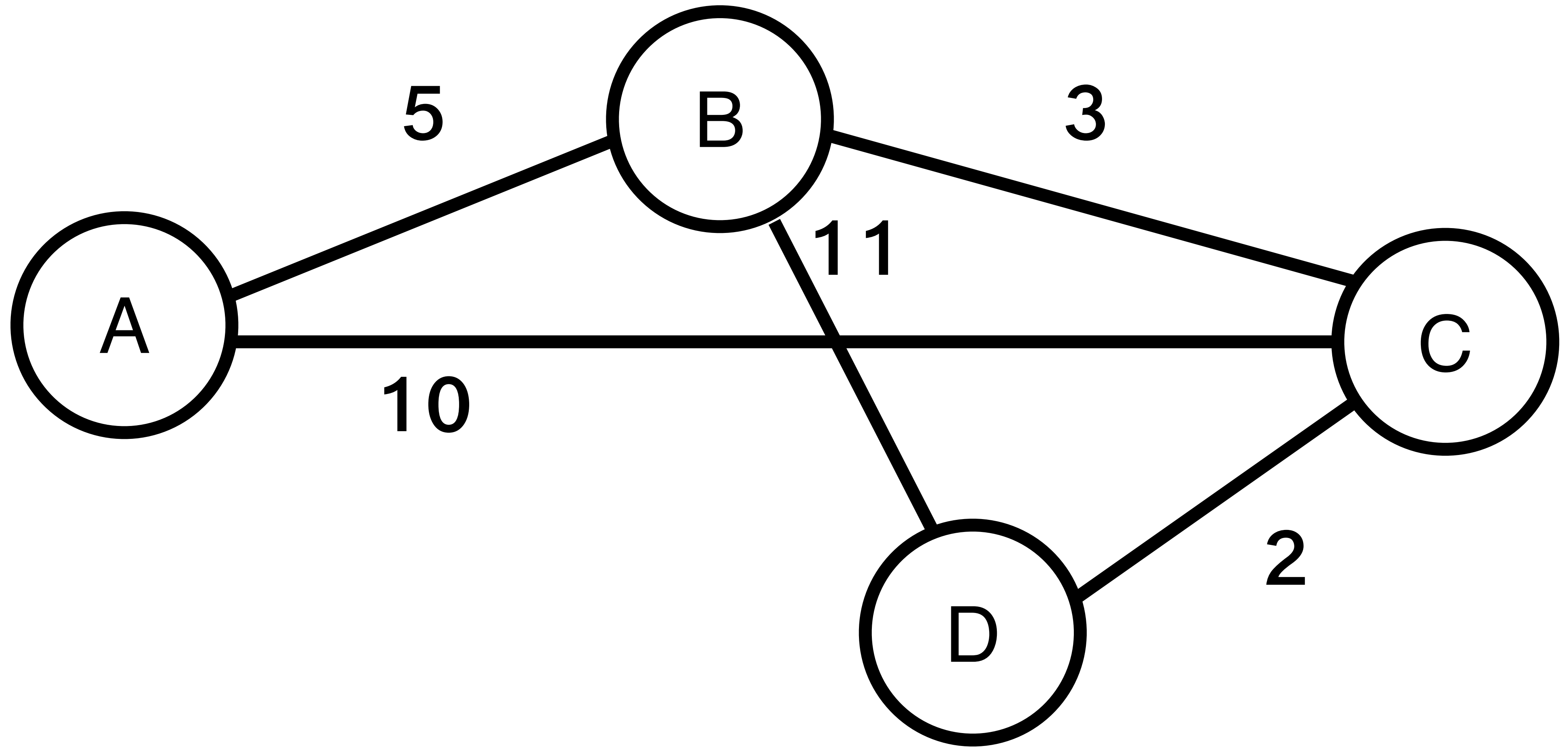
Router B Info.	ID	Link Costs	SEQ#	TTL
A LSP				
B LSP				
C LSP				
D LSP				

LSP Table



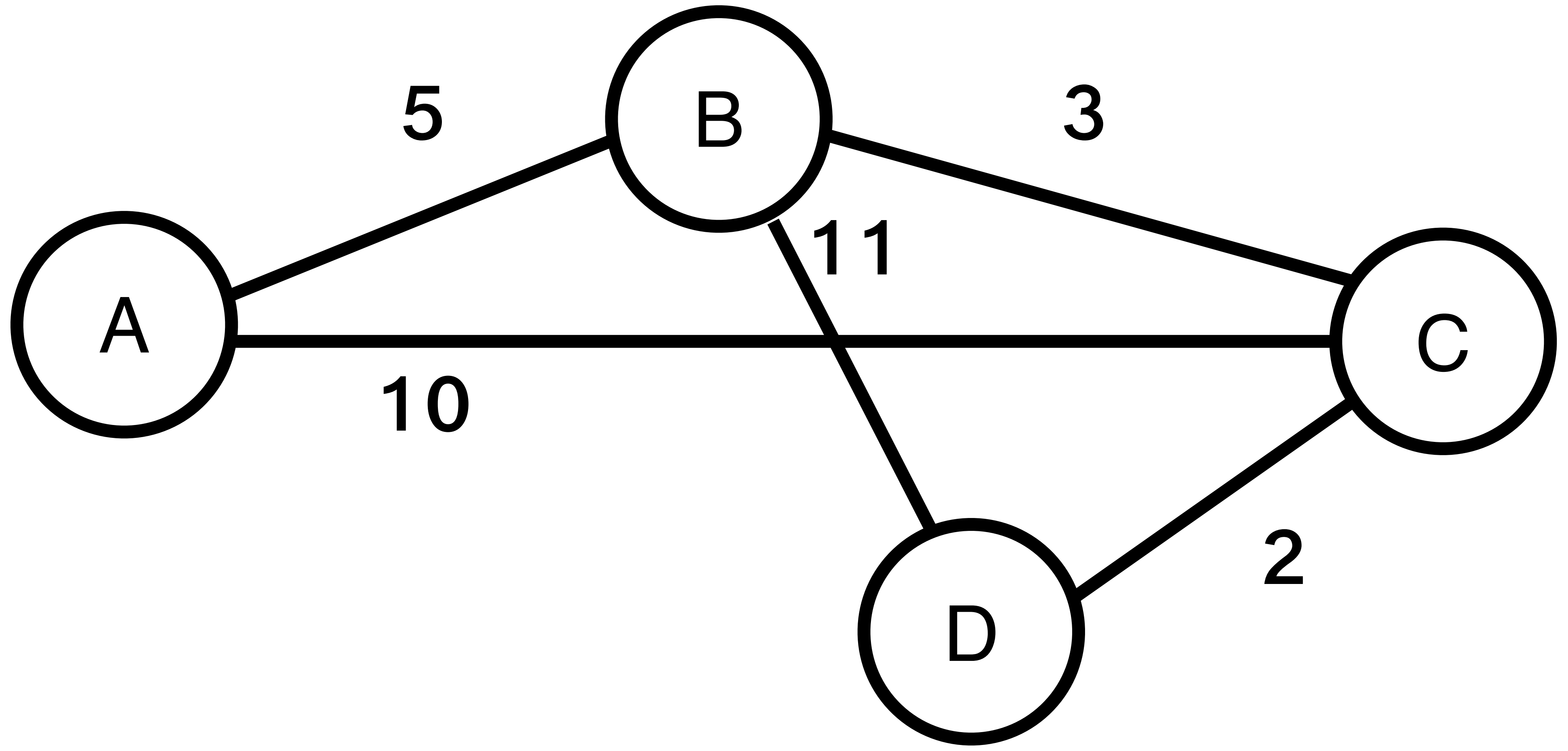
Router B Info.	ID	Link Costs	SEQ#	TTL
A LSP	A	[A, B] = 5, [A, C] = 10	1	63
B LSP	B	[B, A] = 5, [B, C] = 3, [B, D] = 11	1	64
C LSP	C	[C, A] = 10, [C, B] = 3, [C, D] = 2	1	63
D LSP	D	[D, B] = 11, [D, C] = 2	1	63

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 two nodes i, j ∈ N and l(i,j) = ∞ if no edge connects i and j

Problem Formulation: Dijkstra Algorithm



- 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 two nodes i, j∈N and l(i,j) = ∞ if no edge connects i and j

Dijkstra’s Shortest-Path Routing

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

Dijkstra's Shortest-Path Routing

- Inputs:
  - N: the set of nodes in the graph
  - l(i,j): the non-negative cost associated with the edge between two nodes i, j∈ N and l(i,j) = ∞ if no edge connects i and j

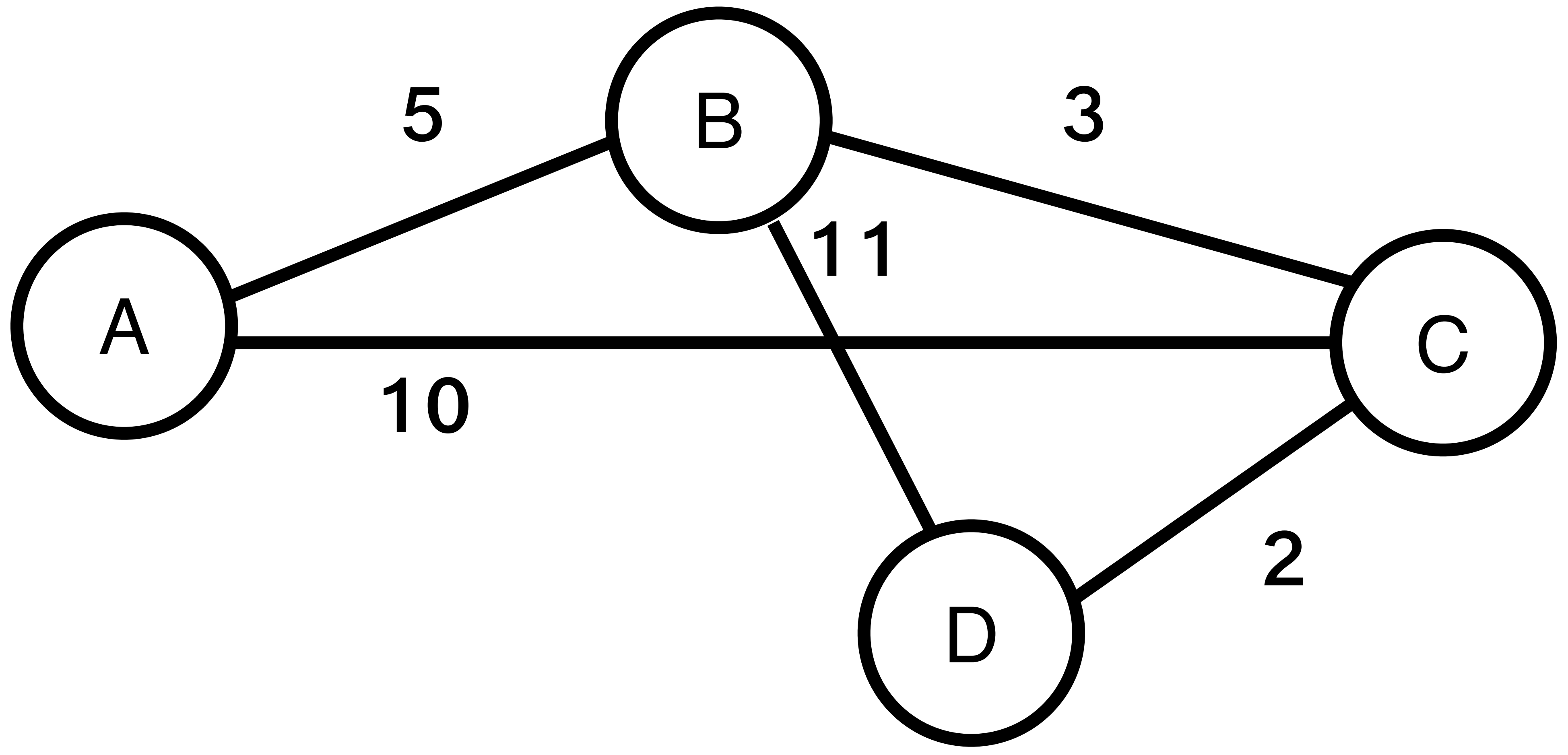
Let s∈N be the starting node which executes the algorithm to find shortest paths to all other nodes in N

Dijkstra’s Algorithm

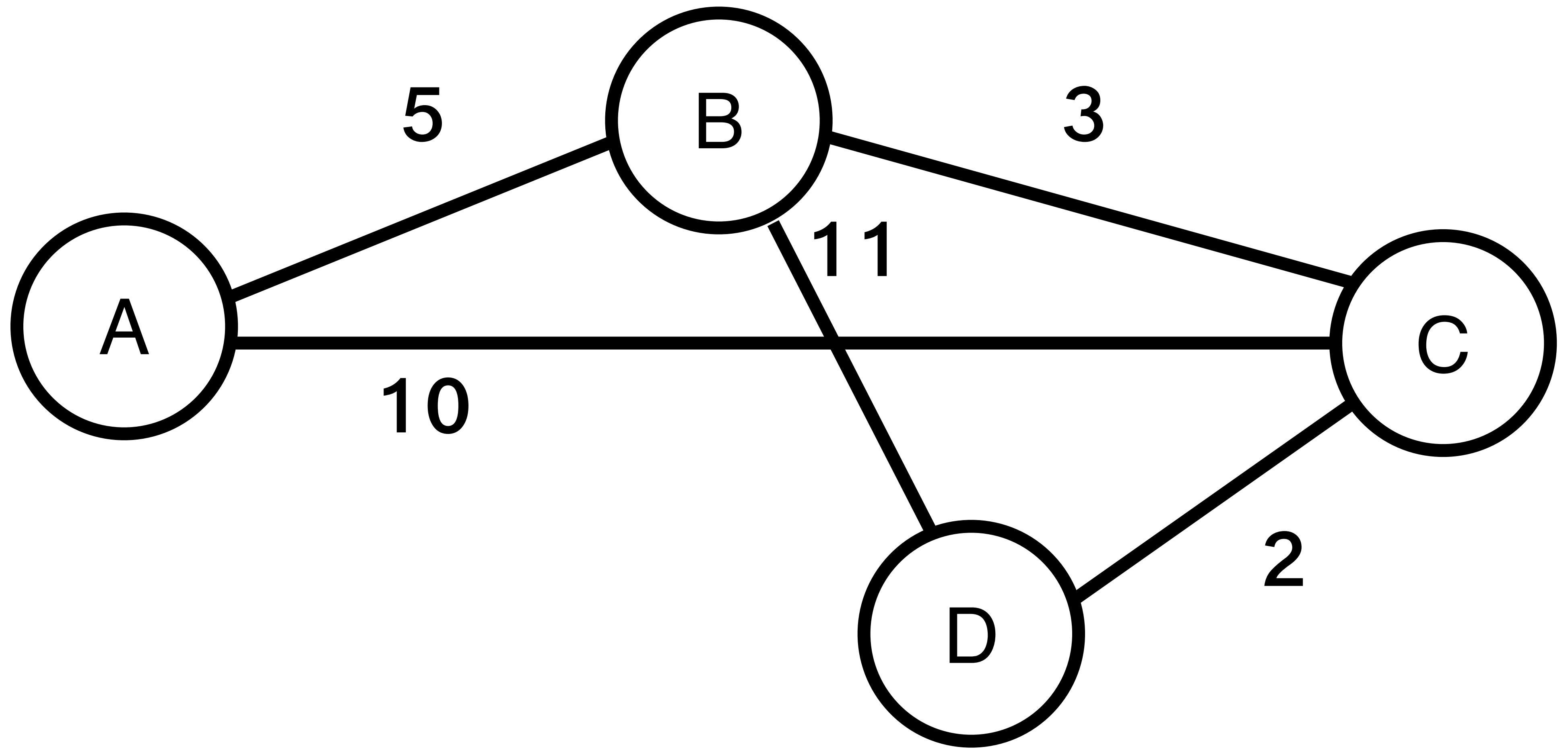
- Variables:
  - M: set of nodes incorporated so far by the algorithm
  - C(n): the cost of a path from s to each node n

```
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

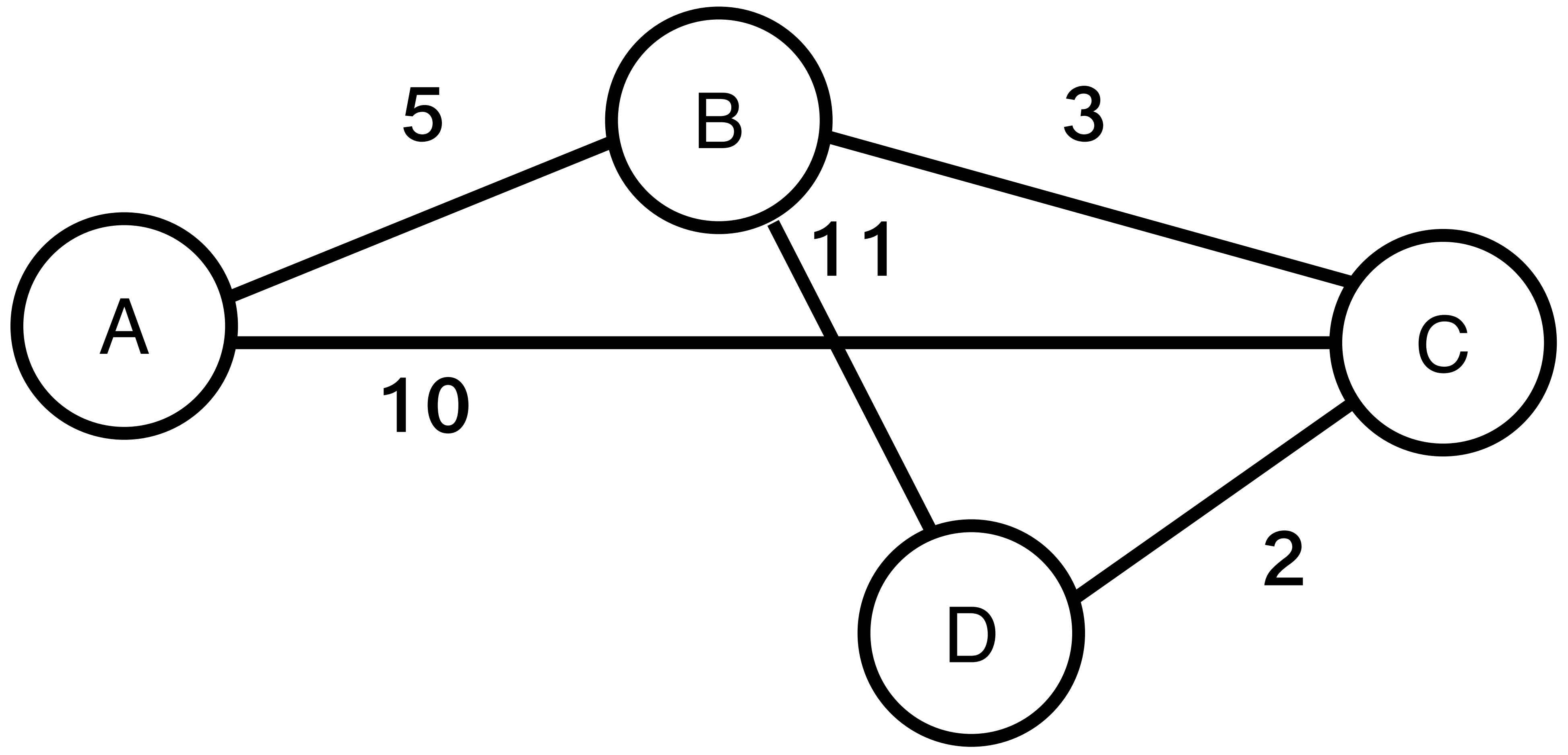


Building Routing Table for Node D



Step	Confirmed list	Tentative list	Comment

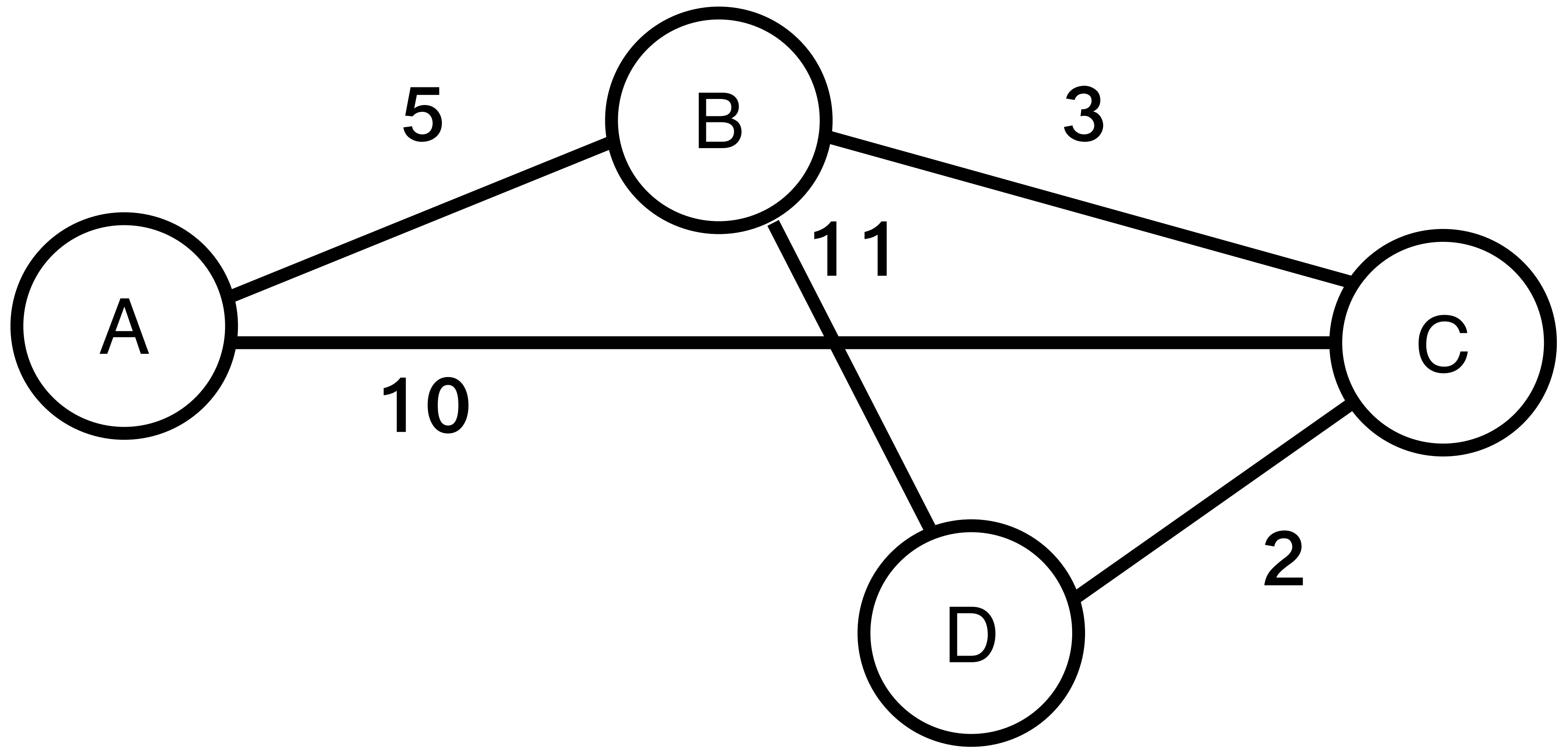
Building Routing Table for Node D



Step	Confirmed list	Tentative list	Comment

M from the above algorithm

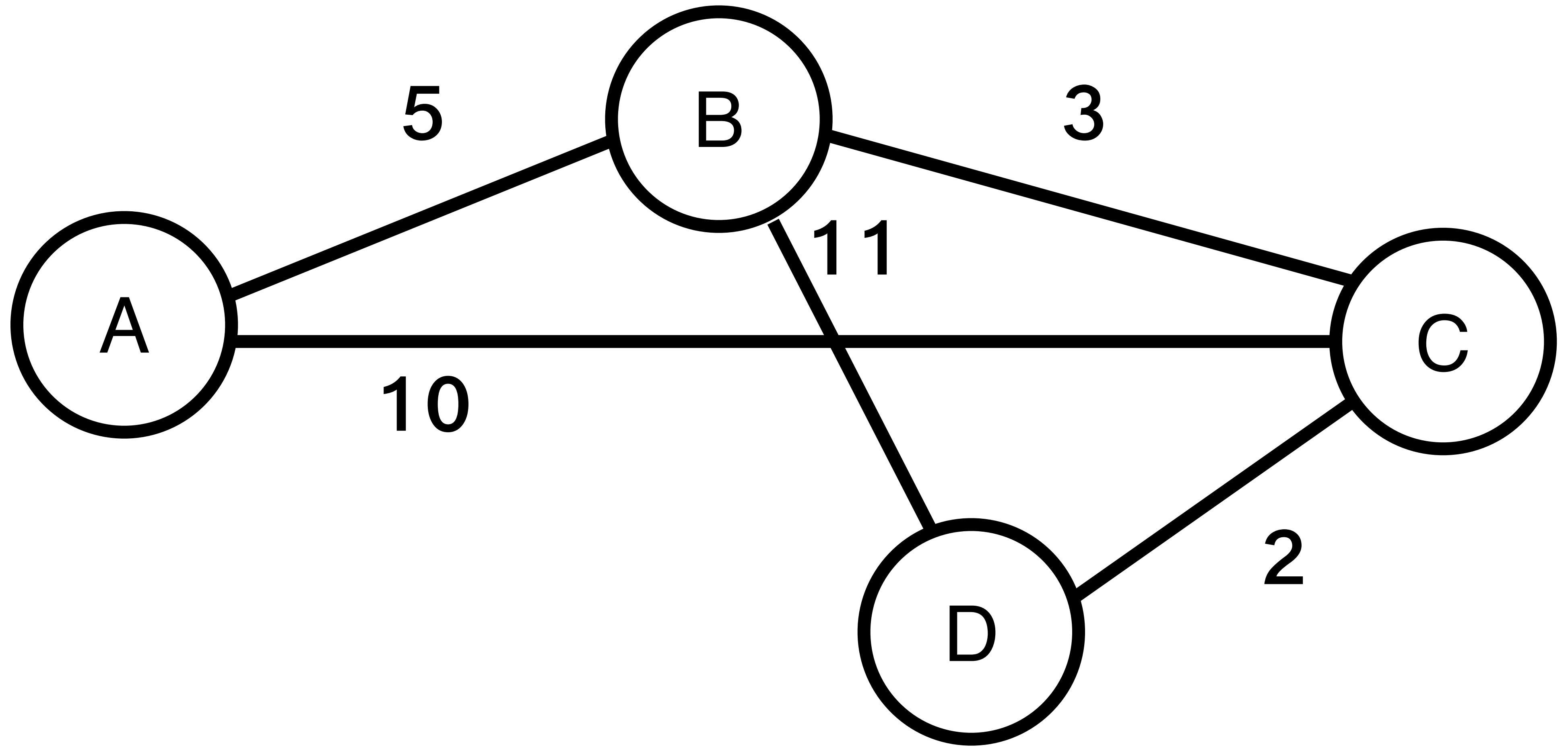
Building Routing Table for Node D



Step	Confirmed list	Tentative list	Comment

(N-M) from the above algorithm

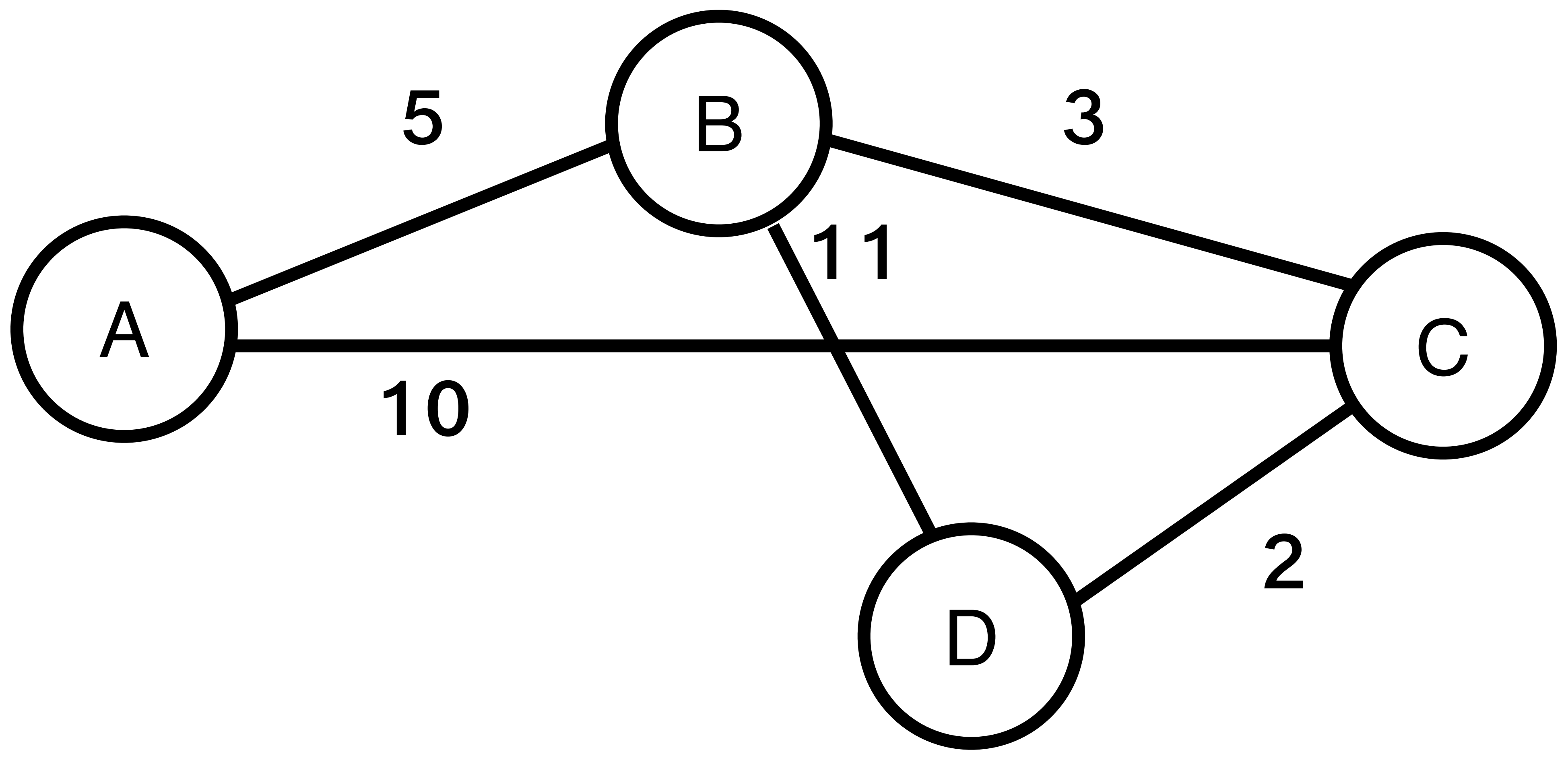
Building Routing Table for Node D



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

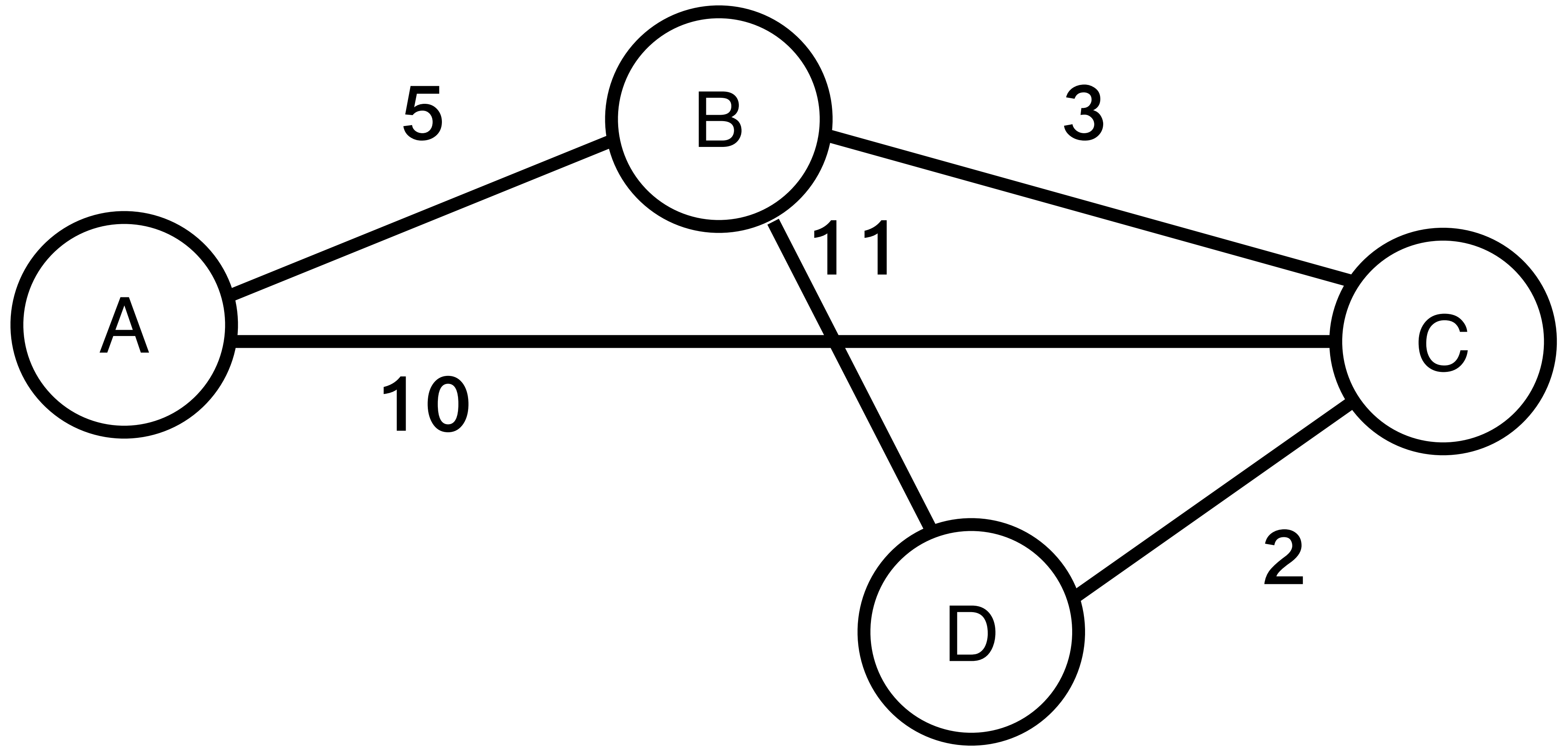


Routing table entry:
(Destination, Cost, NextHop)

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))
```

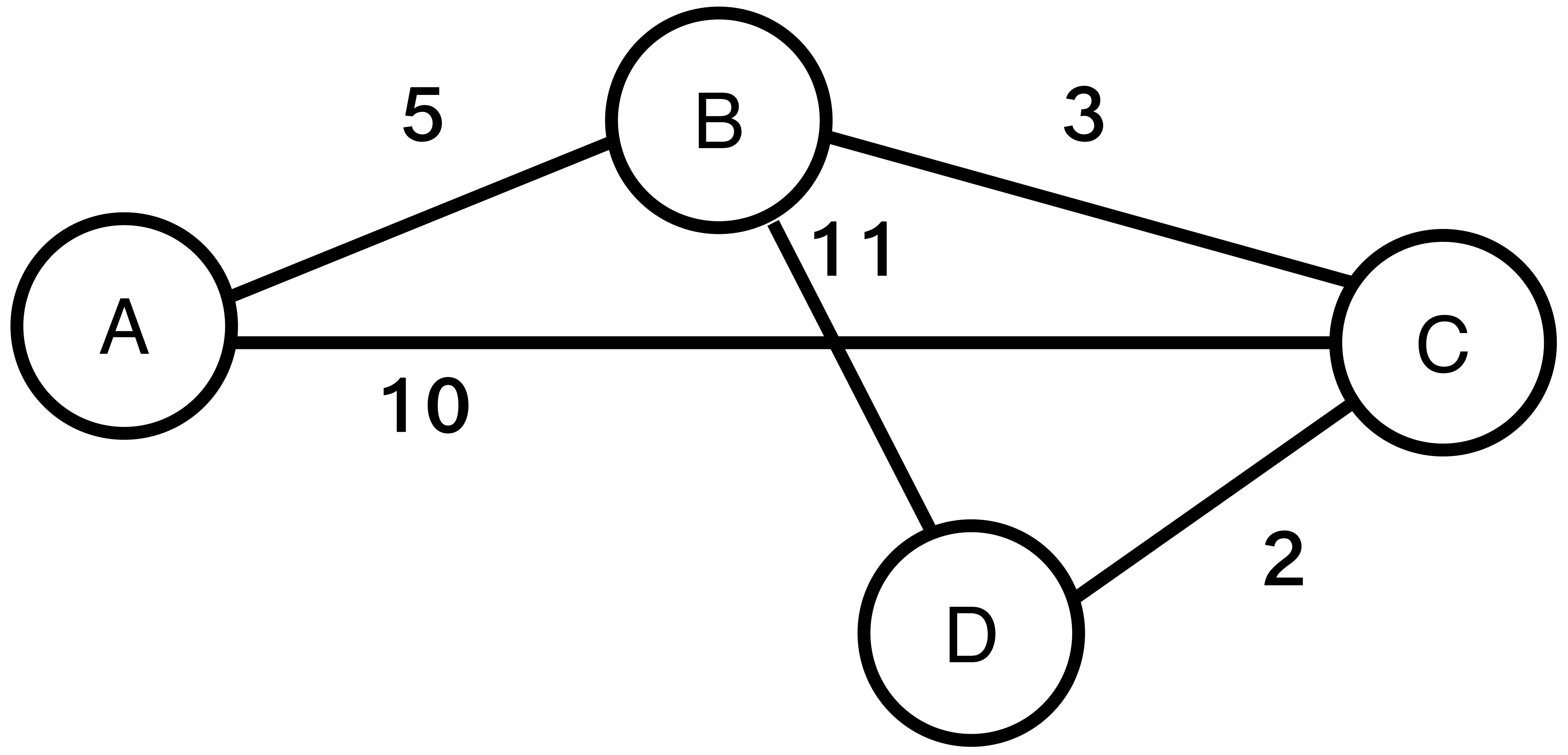
Building Routing Table for Node D



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

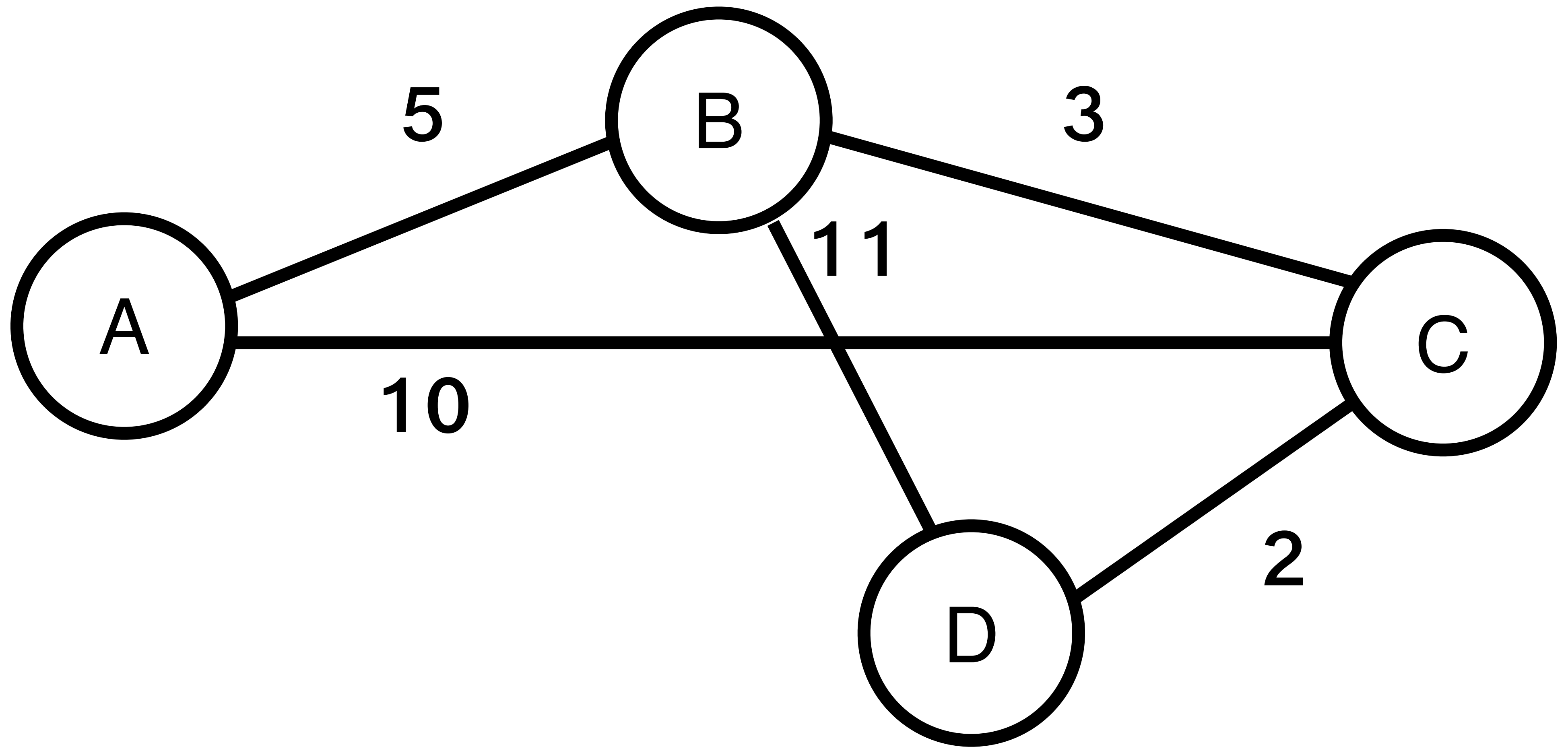


Routing table entry:
(Destination, Cost, NextHop)

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))
```

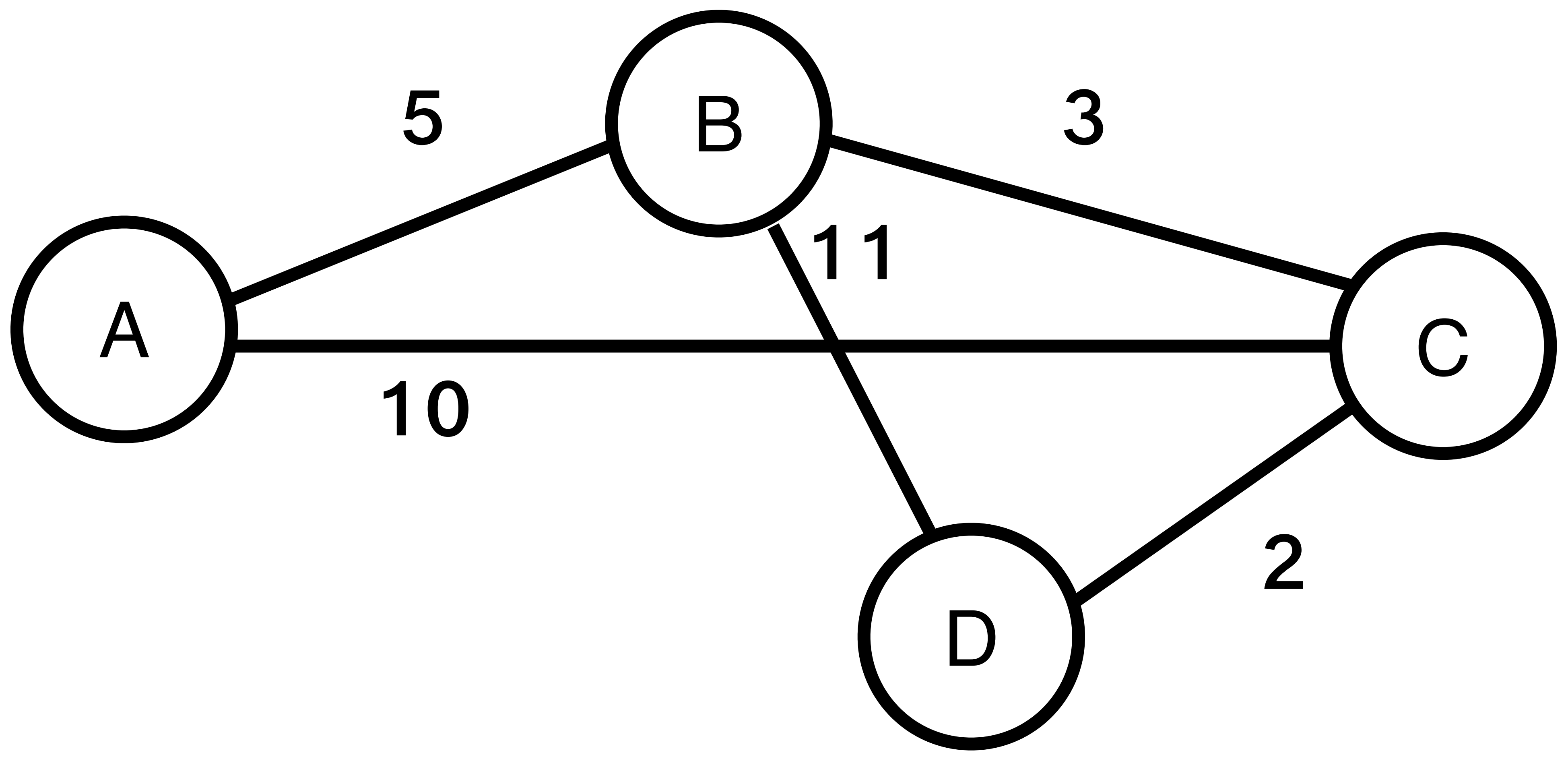
Building Routing Table for Node D



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

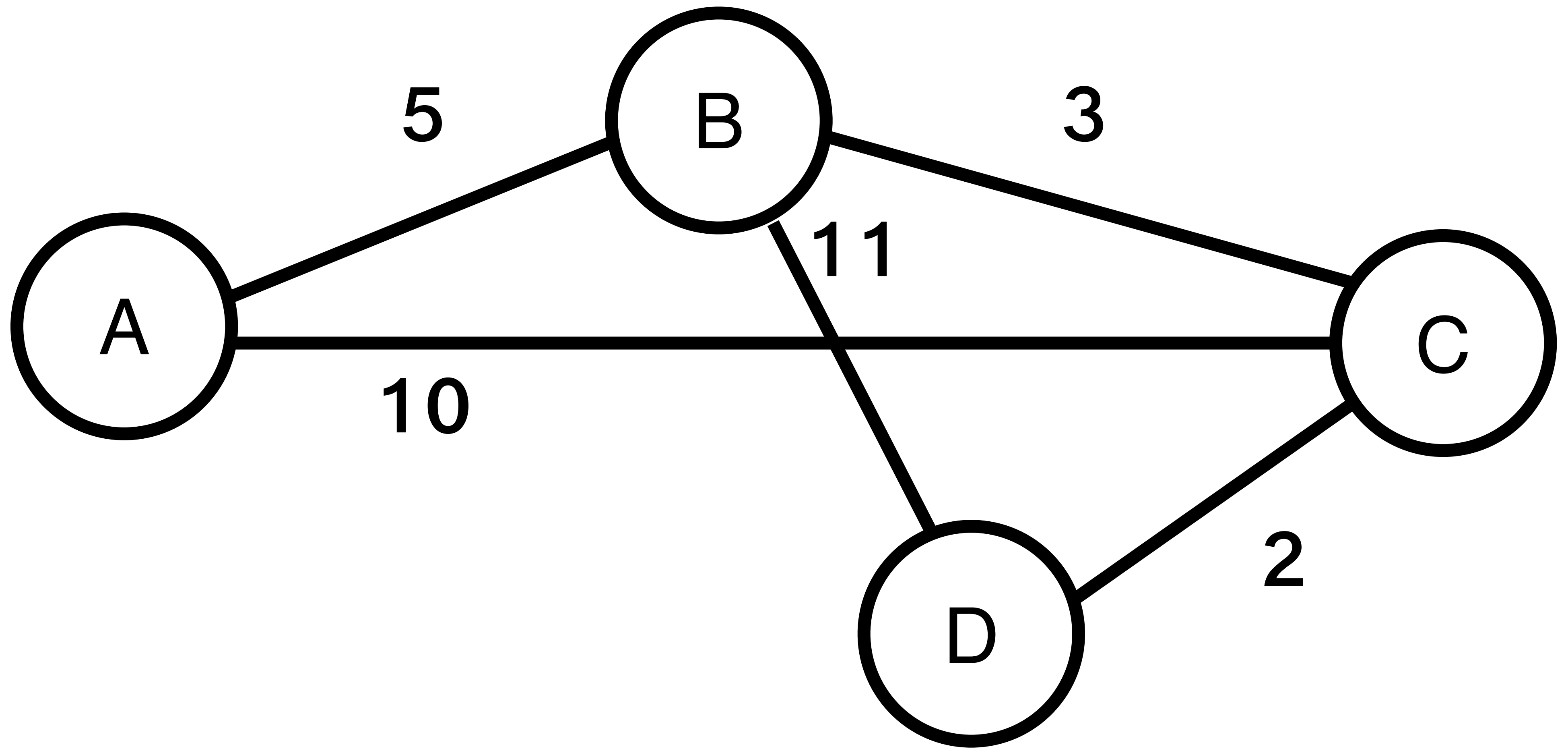


Routing table entry:
(Destination, Cost, NextHop)

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))
```

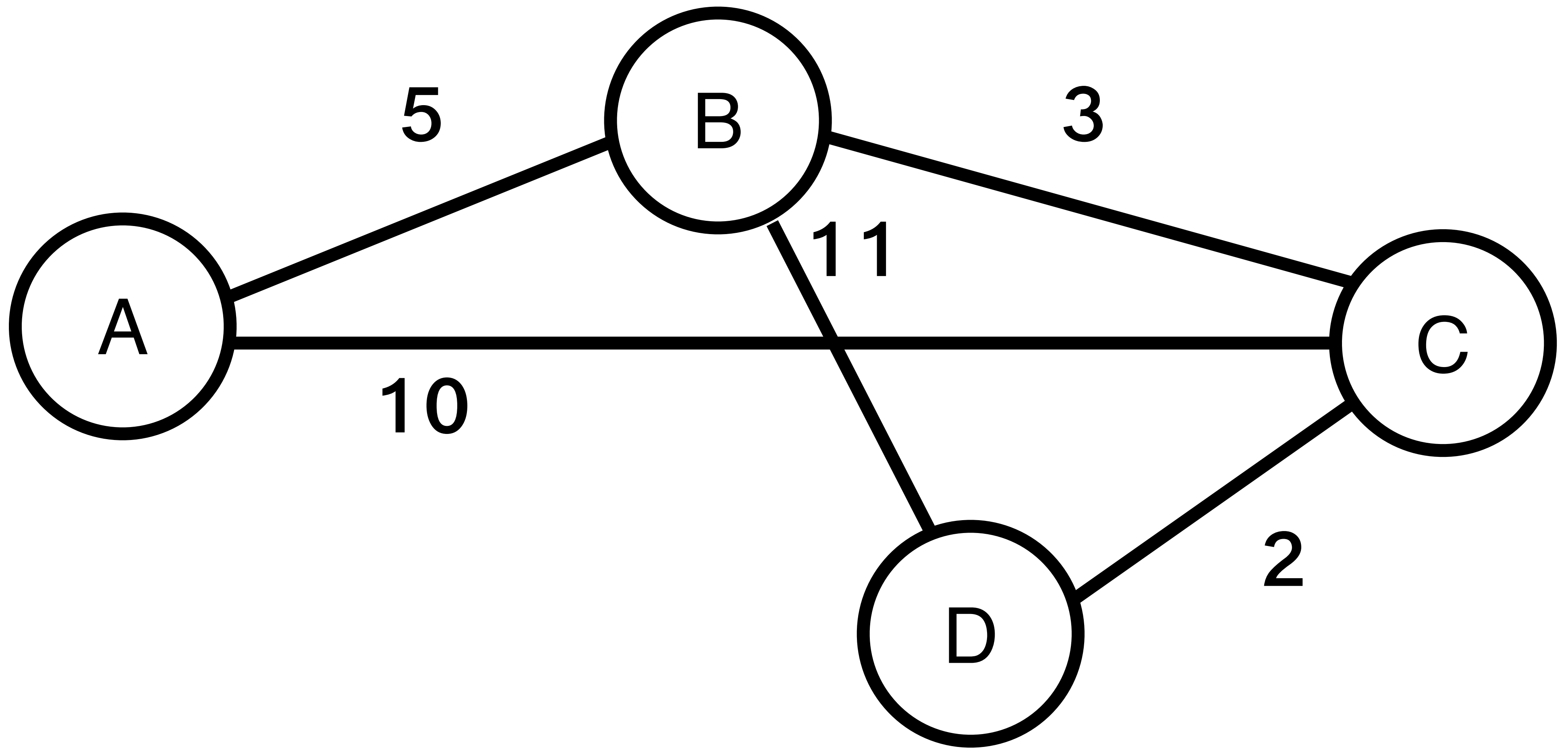
Building Routing Table for Node D



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

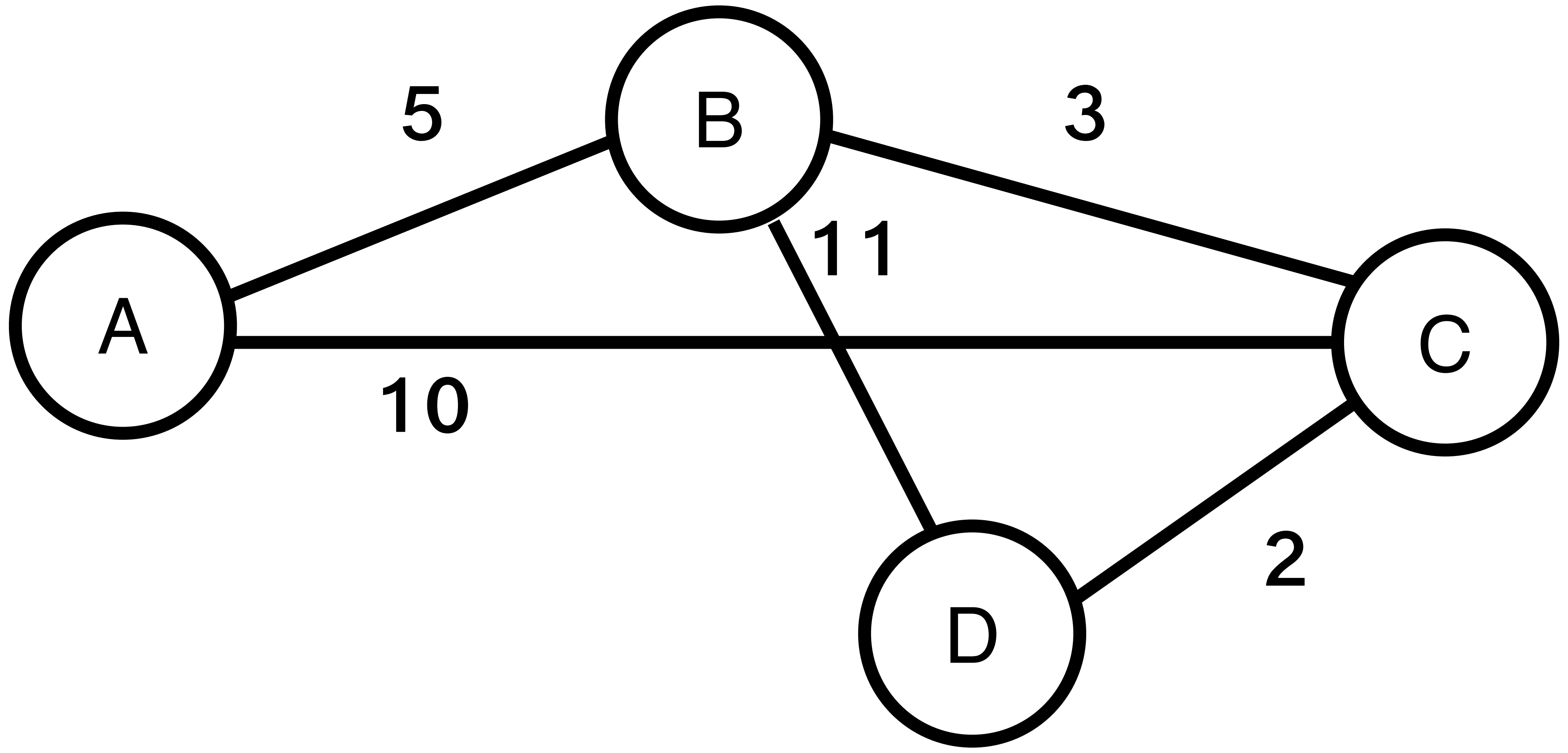


Routing table entry:
(Destination, Cost, NextHop)

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 ($N \neq M$)
    M = $M \cup \{w\}$ 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))
```

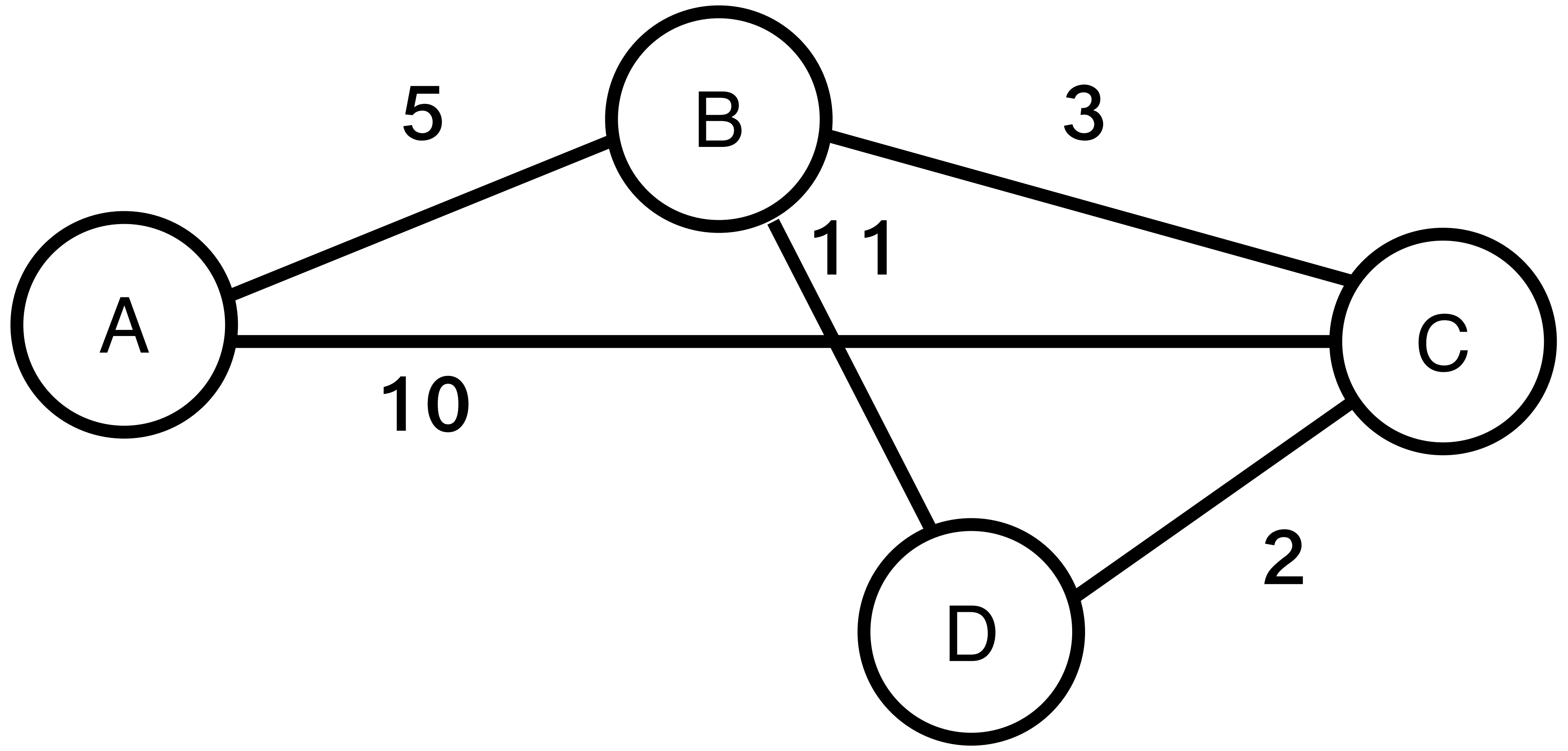
Building Routing Table for Node D



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

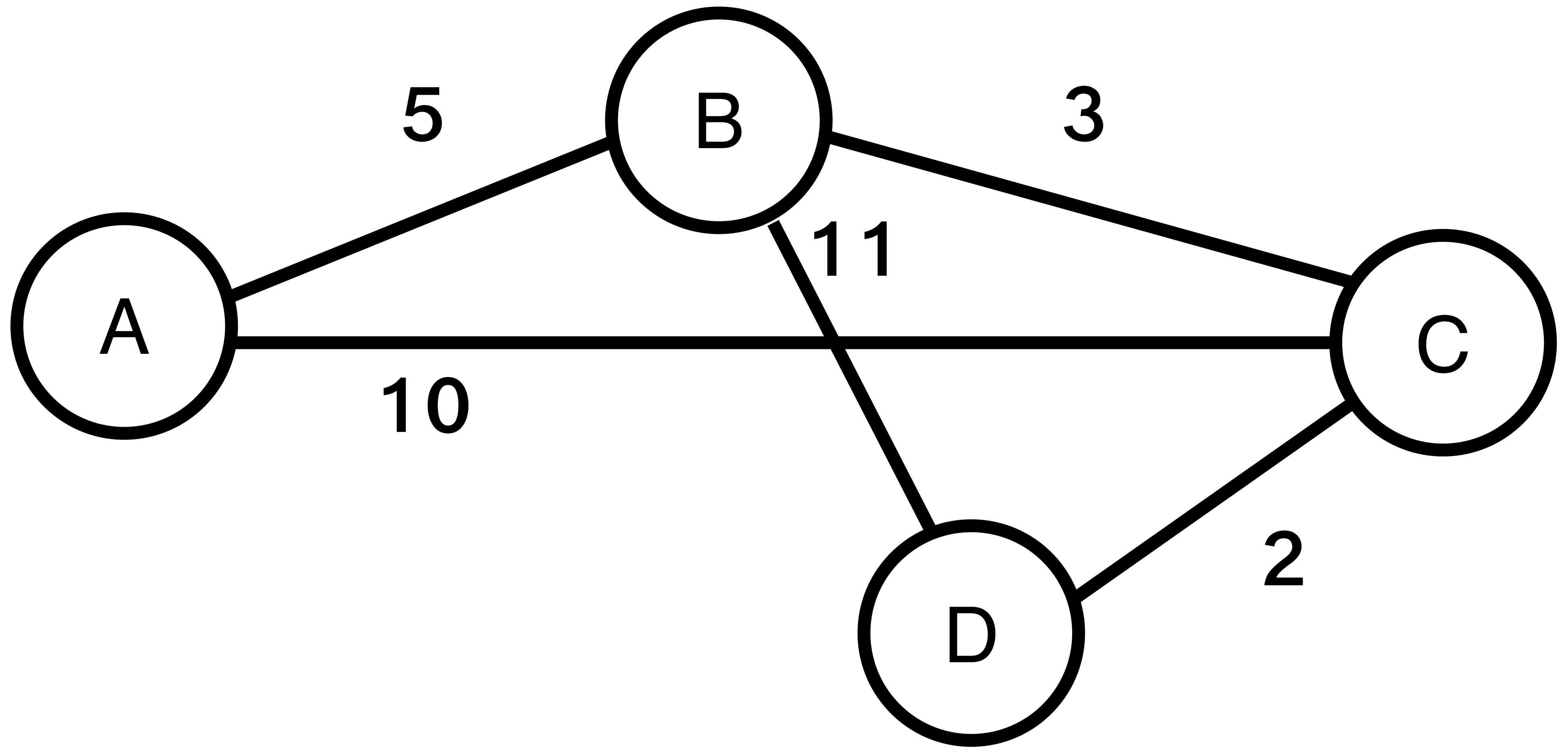


Routing table entry:
(Destination, Cost, NextHop)

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 $(N \neq M)$
    M = M $\cup$ {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))
```

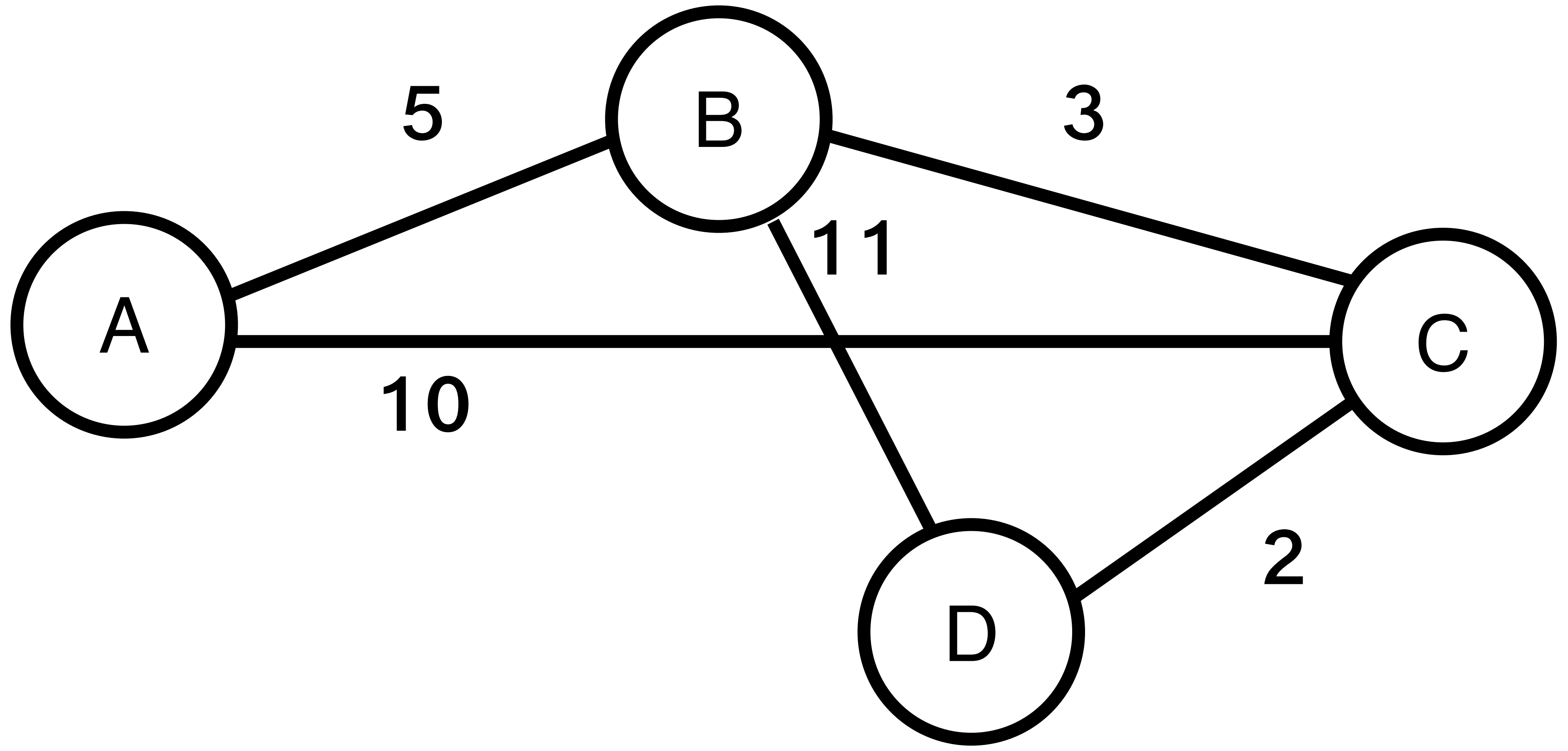
Building Routing Table for Node D



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

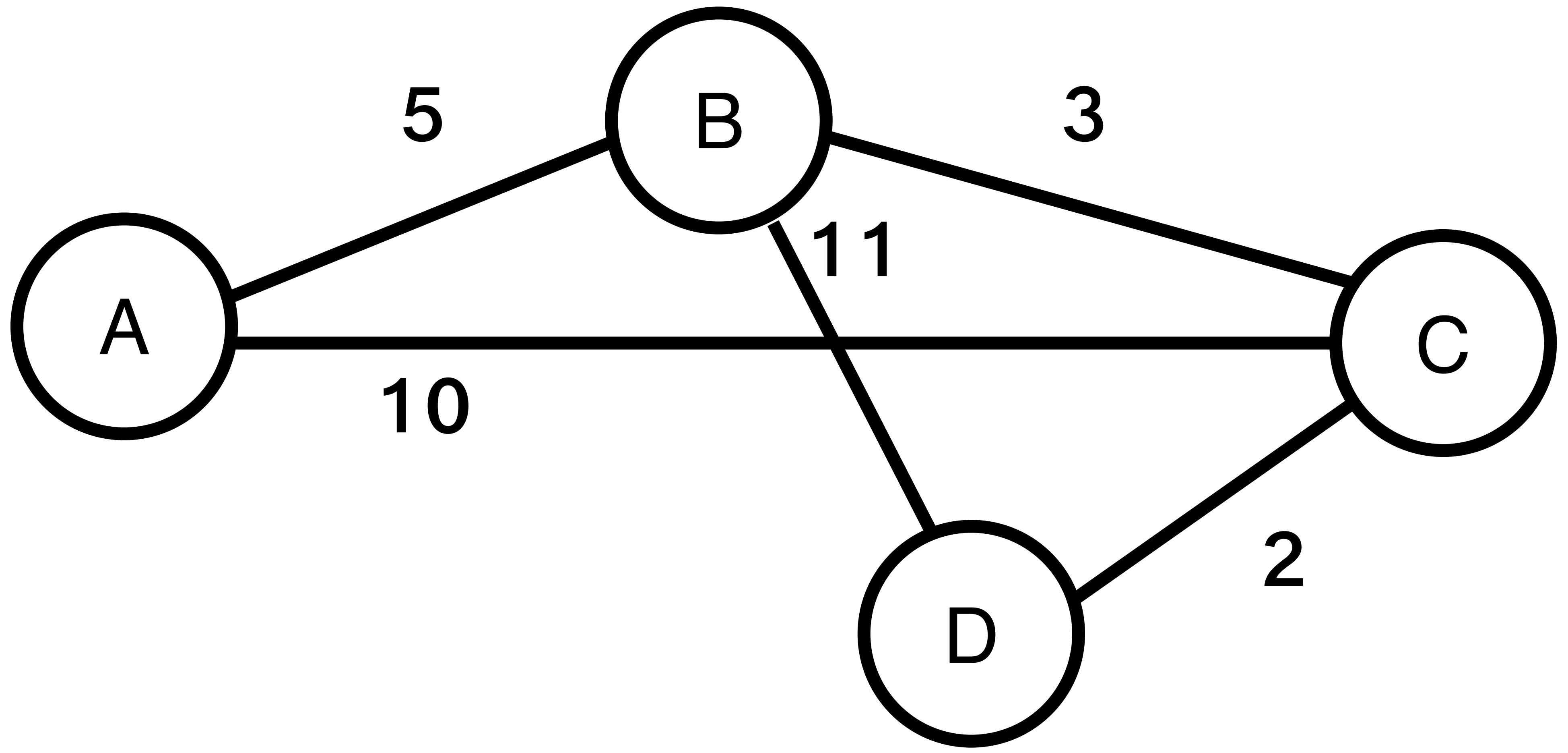


Routing table entry:
(Destination, Cost, NextHop)

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

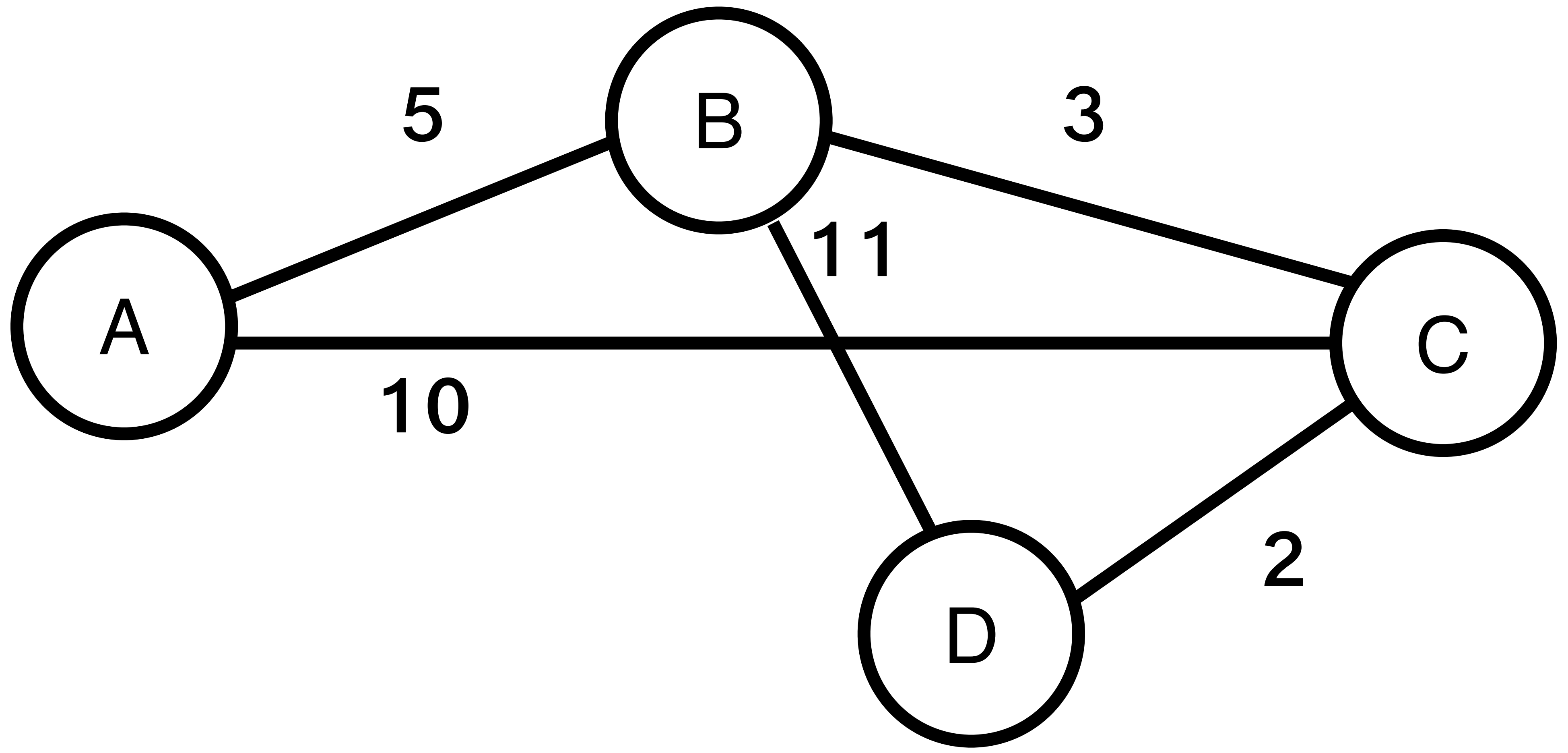


Routing table entry:
(Destination, Cost, NextHop)

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			tative list
4			he cost
5	(D,		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)

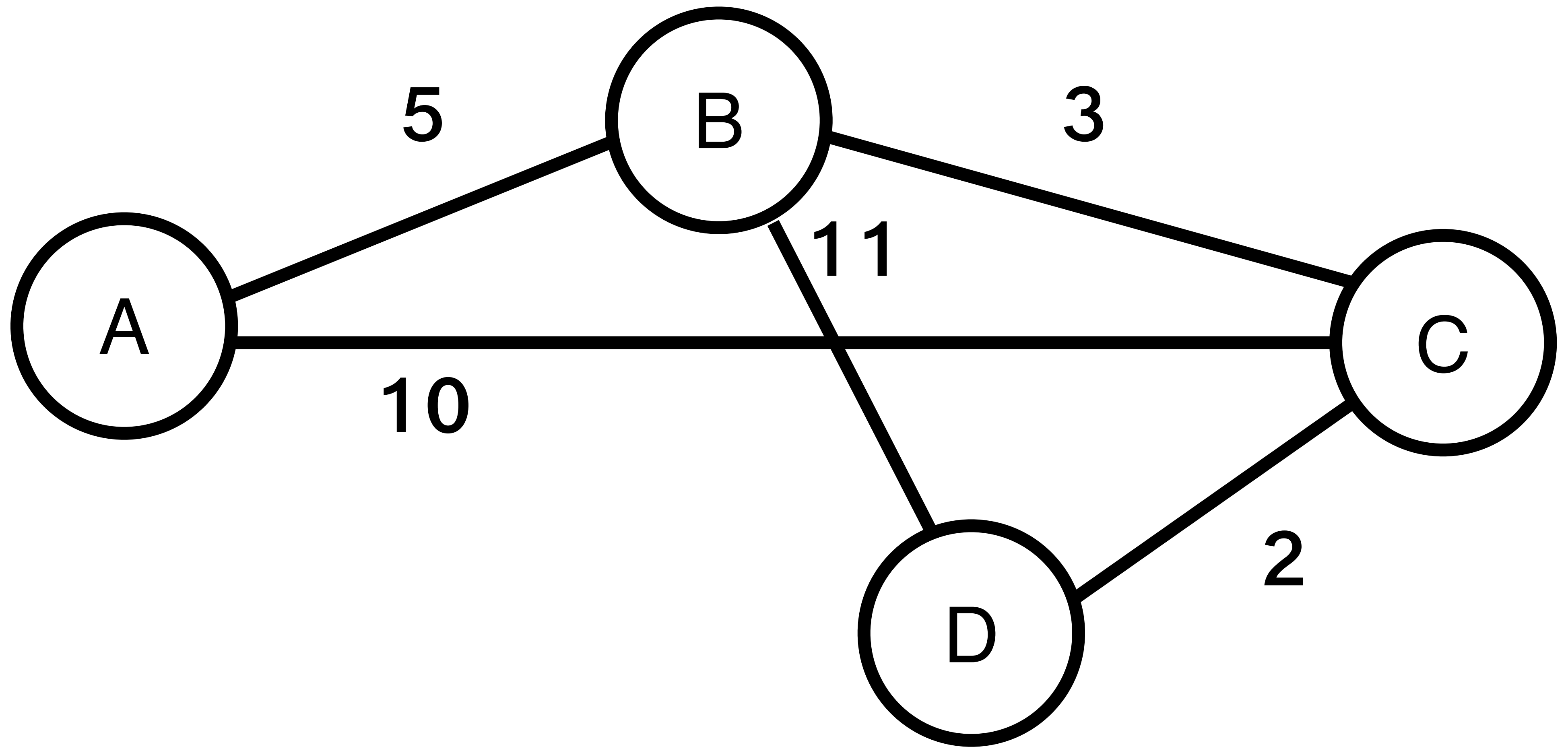
Building Routing Table for Node A



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

Link State Routing: Two Steps

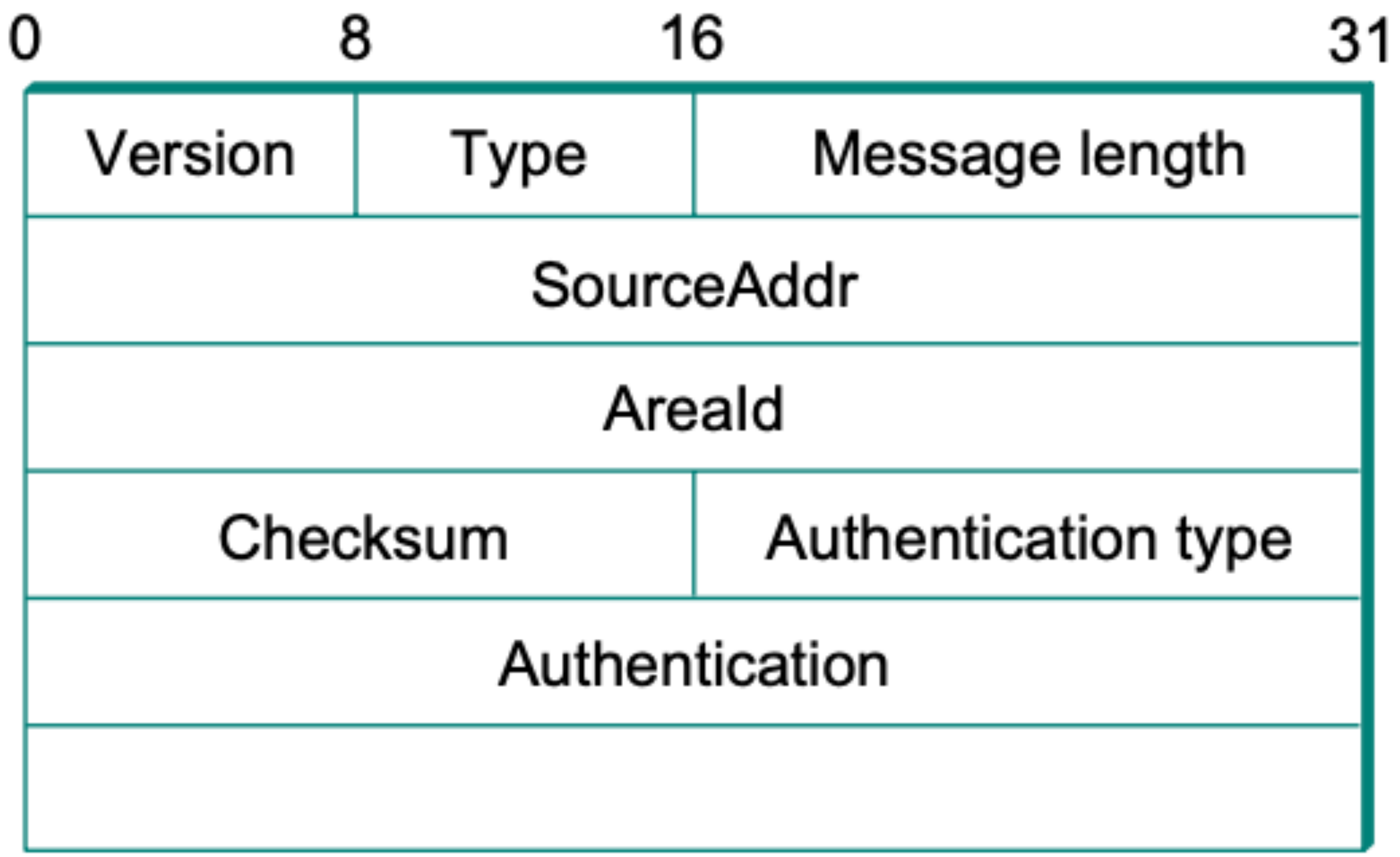
- Step #1: Reliable flooding
  - Each node maintains a global view of the network

- **Step #2: Route calculation**
  - **Use the Dijkstra algorithm to figure out the shortest path**

Open Shortest Path First (OSPF)

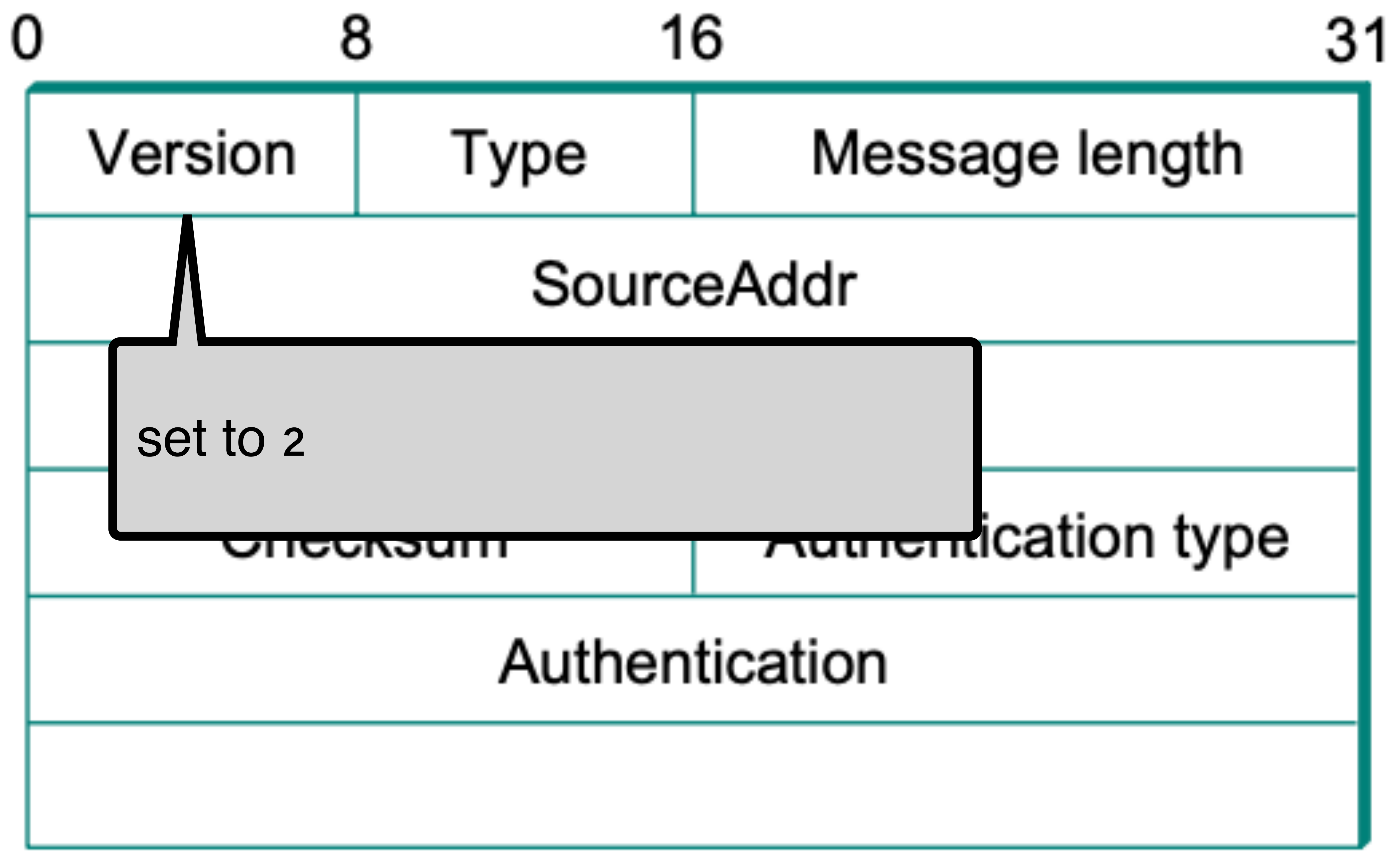
- OSPF Distance vector routing in practice
  - Originally designed in the 1980s
  - V2 is defined in RFC 2328 (1998)
  - V3 is defined in RFC 5340 (2008)

Open Shortest Path First (OSPF)

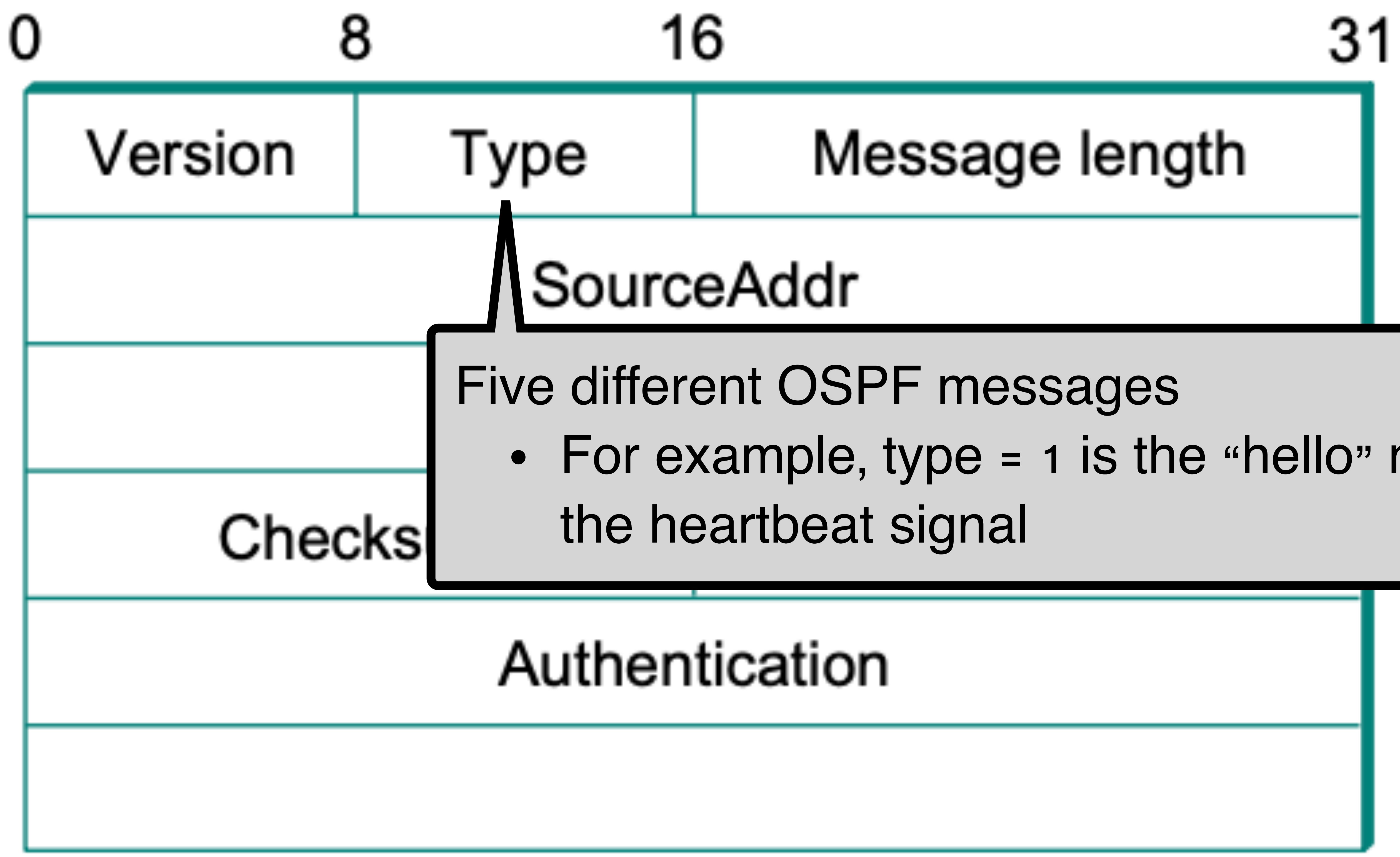


OSPF header format

Open Shortest Path First (OSPF)



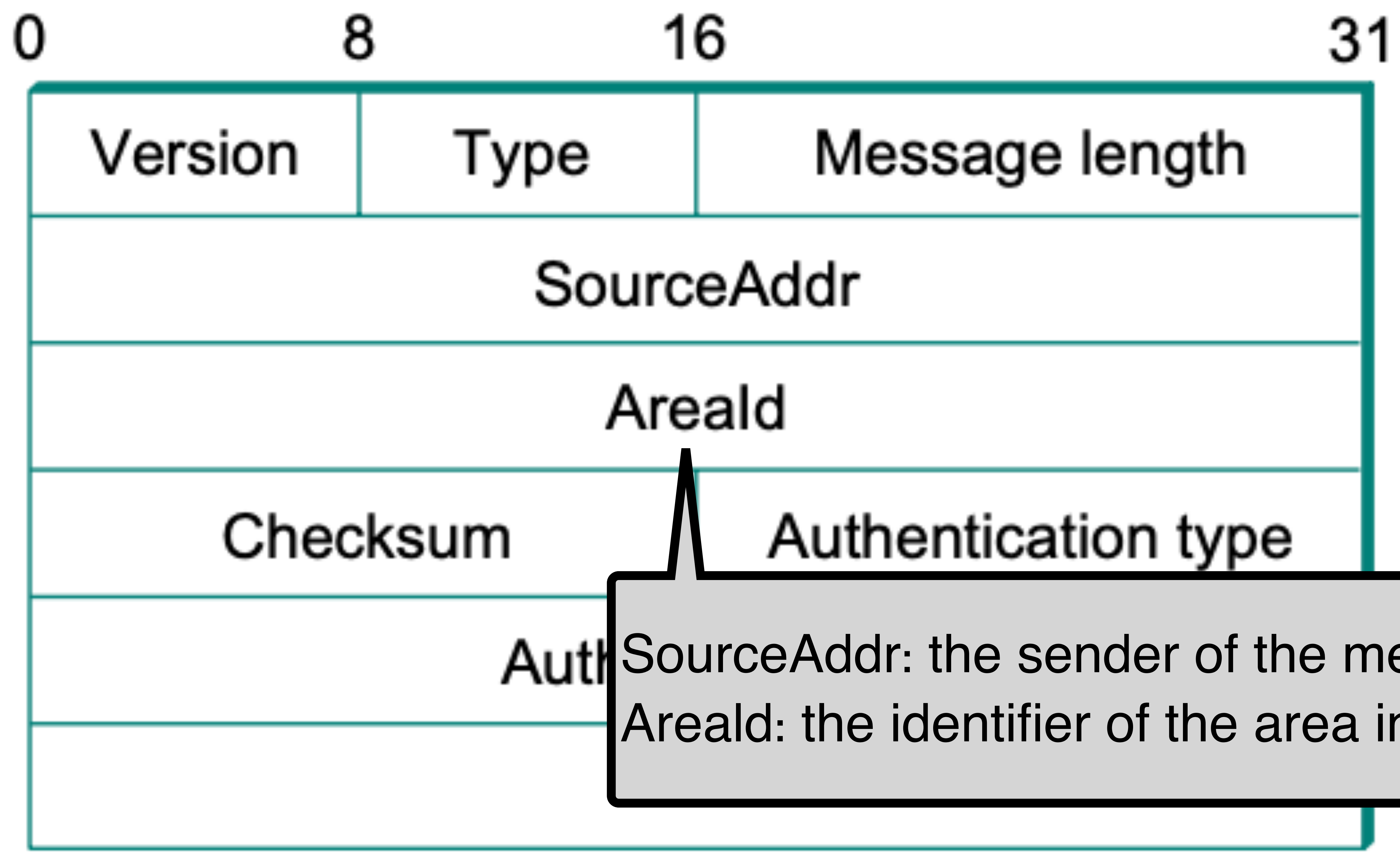
Open Shortest Path First (OSPF)



Five different OSPF messages

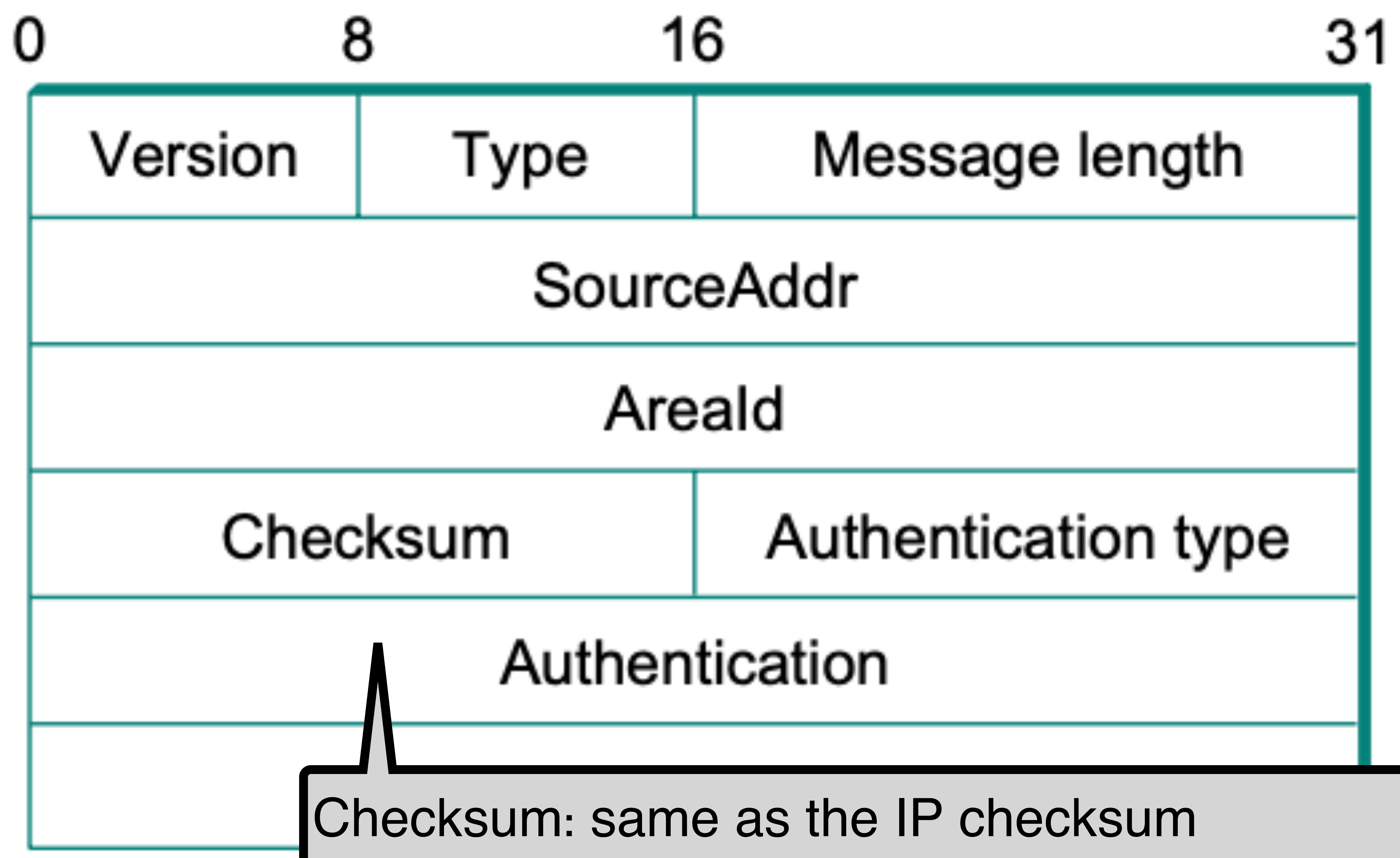
- For example, type = 1 is the "hello" message as the heartbeat signal

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)

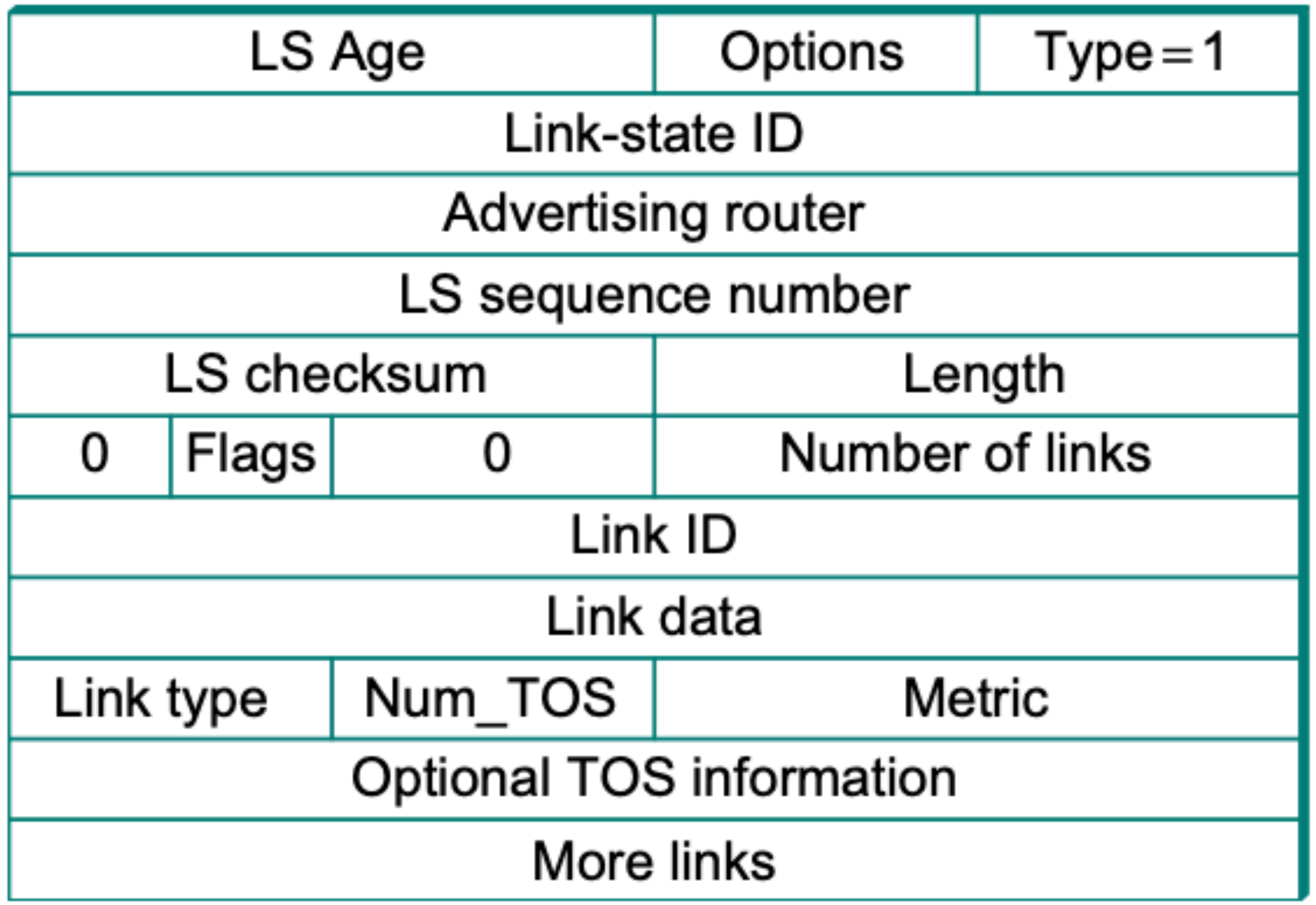


Checksum: same as the IP checksum

Authentication:

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

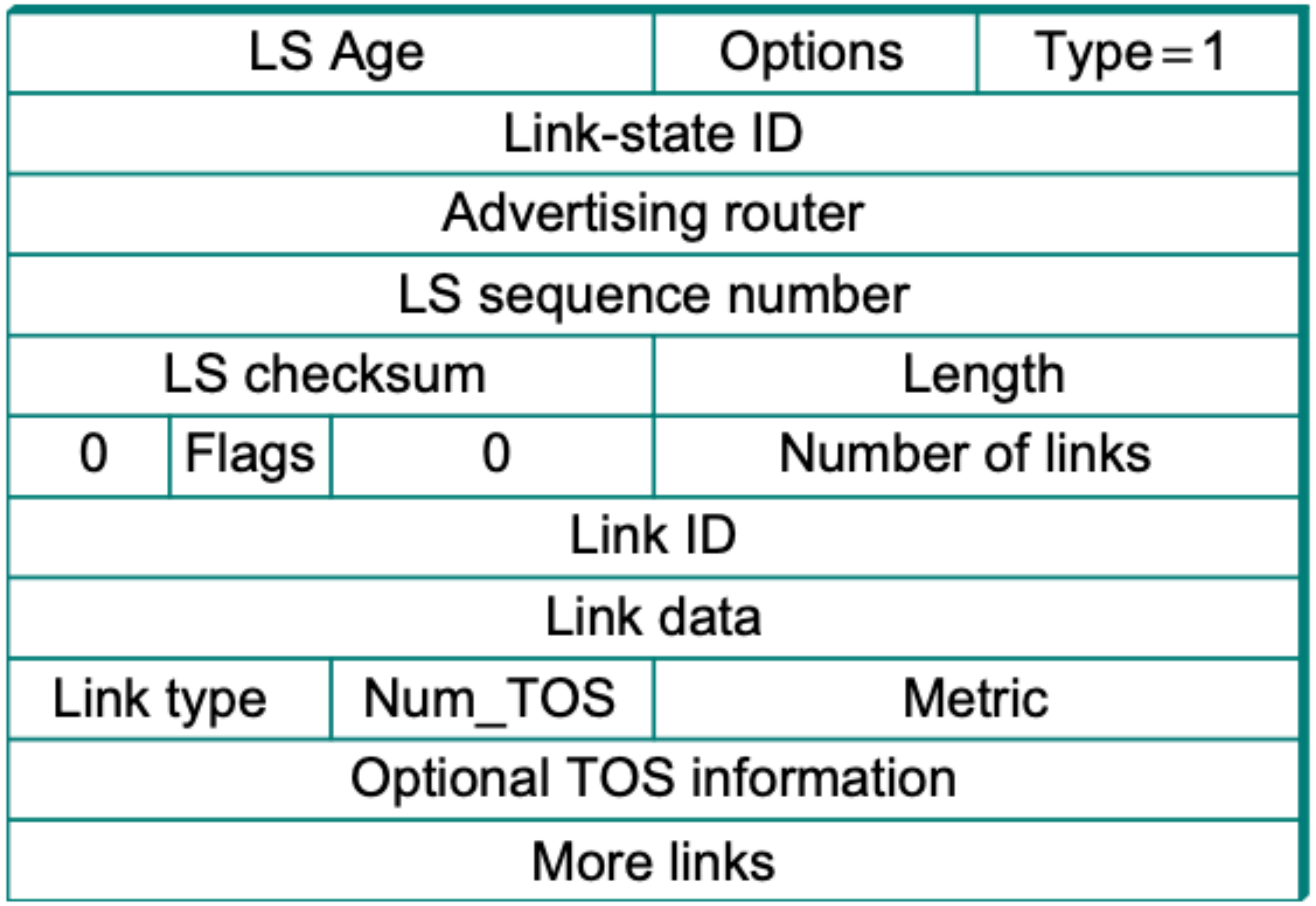
Open Shortest Path First (OSPF)



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

OSPF: Link State Packet



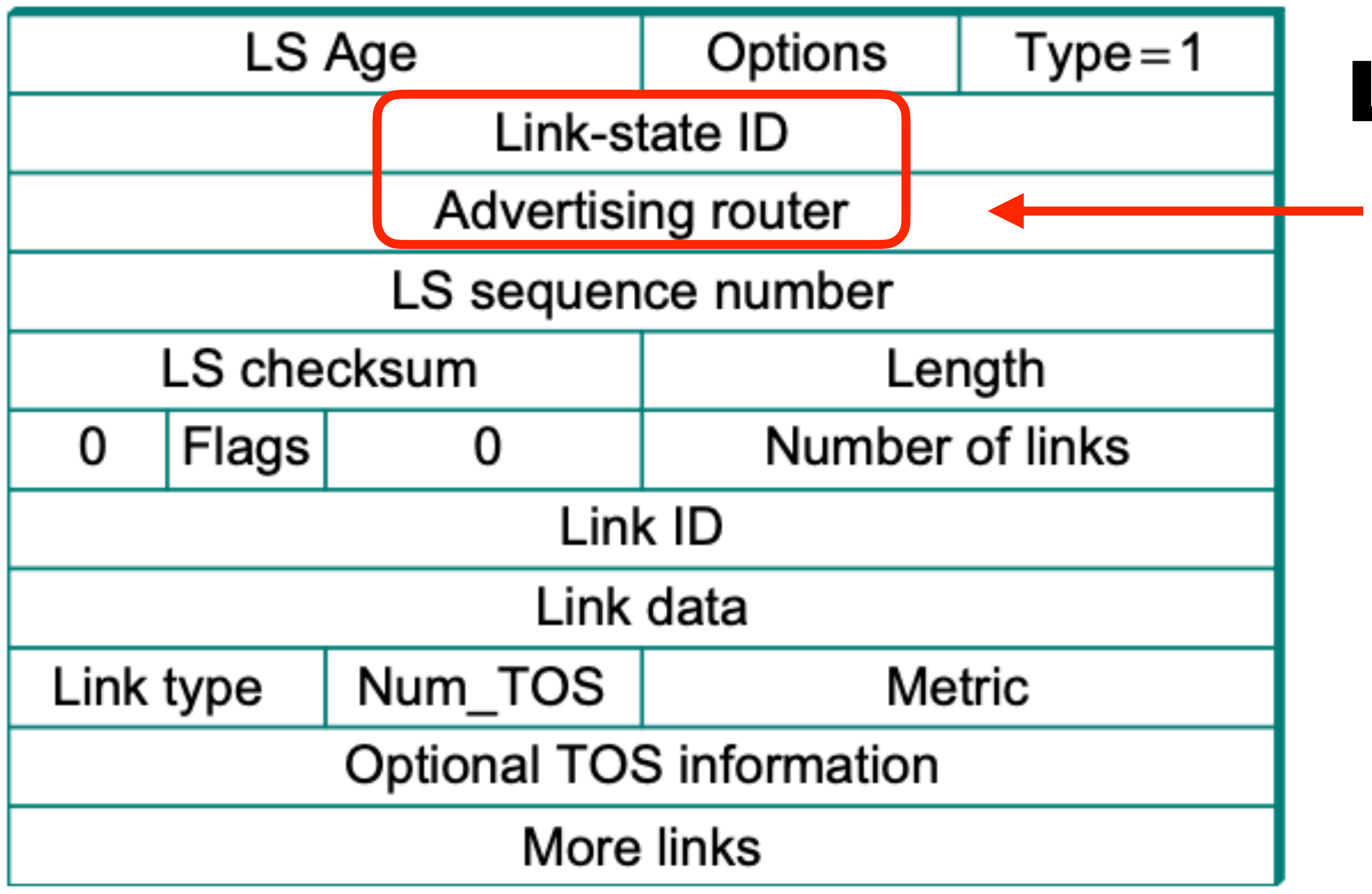
LS Age			Options	Type=1
Link-state ID				
Advertising router				
LS sequence number				
LS checksum			Length	
0	Flags	0	Number of links	
Link ID				
Link data				
Link type		Num_TOS	Metric	
Optional TOS information				
More links				

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

OSPF: Link State Packet

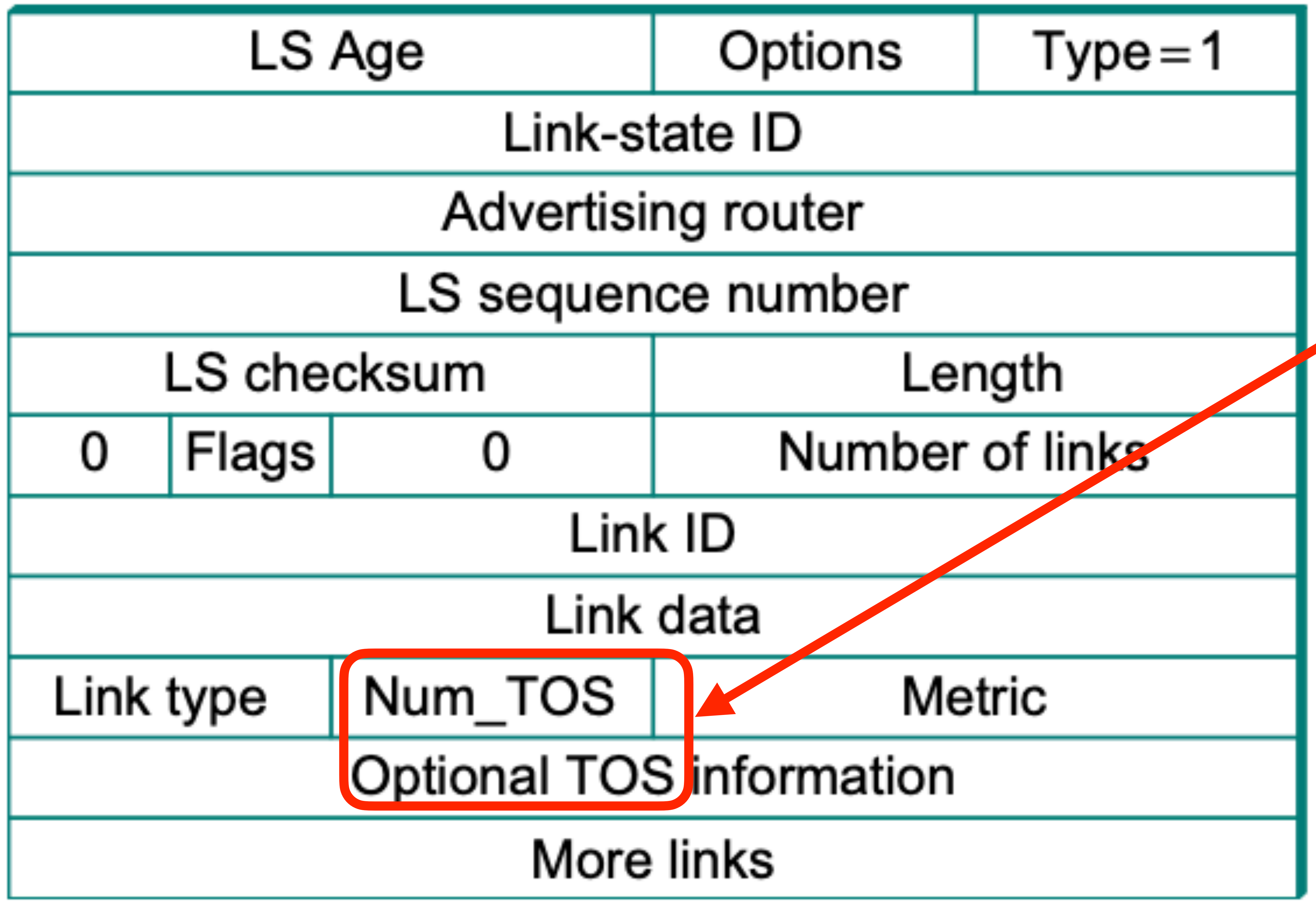


OSPF link-state advertisement

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

OSPF: Link State Packet

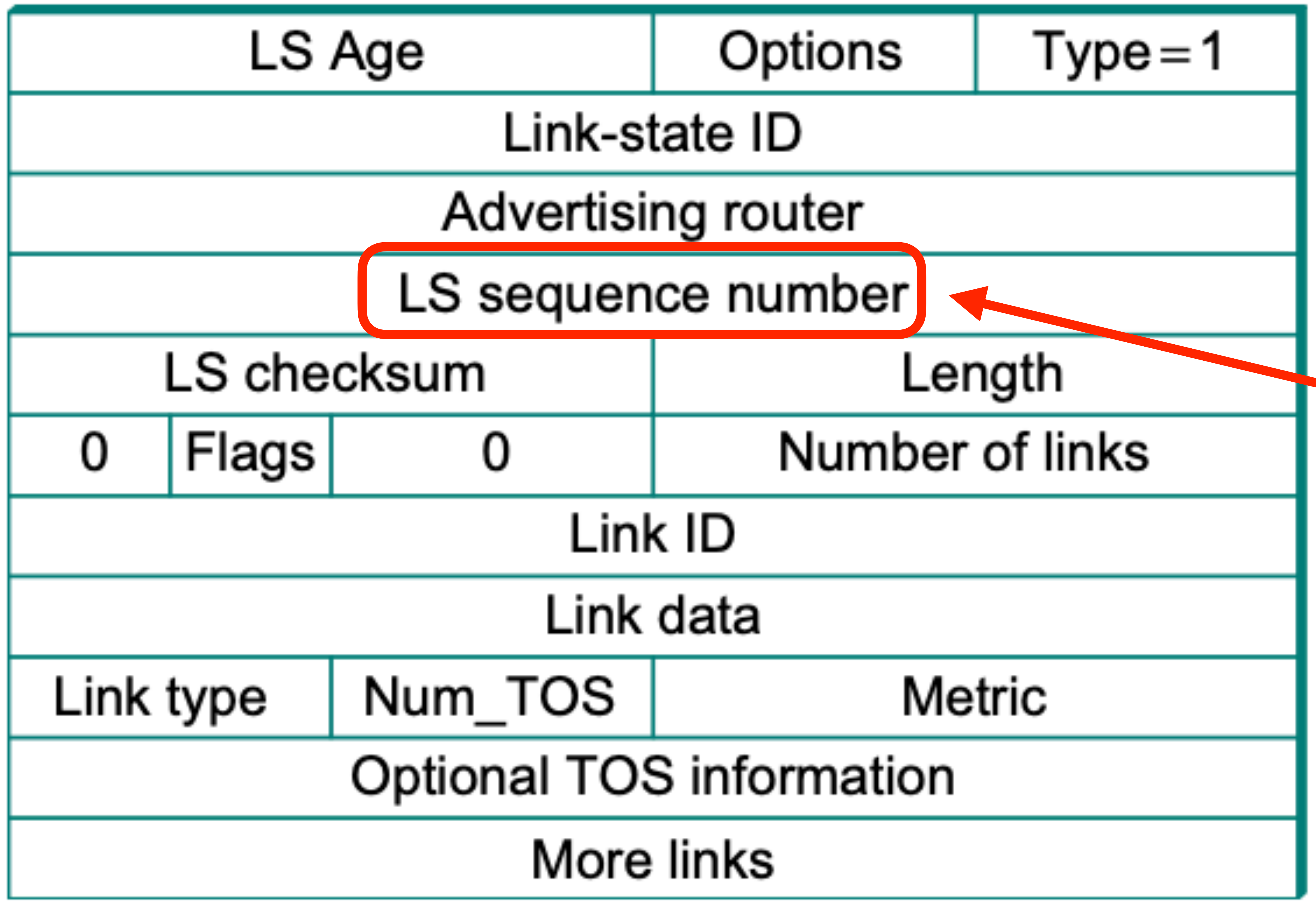


OSPF link-state advertisement

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

OSPF: Link State Packet

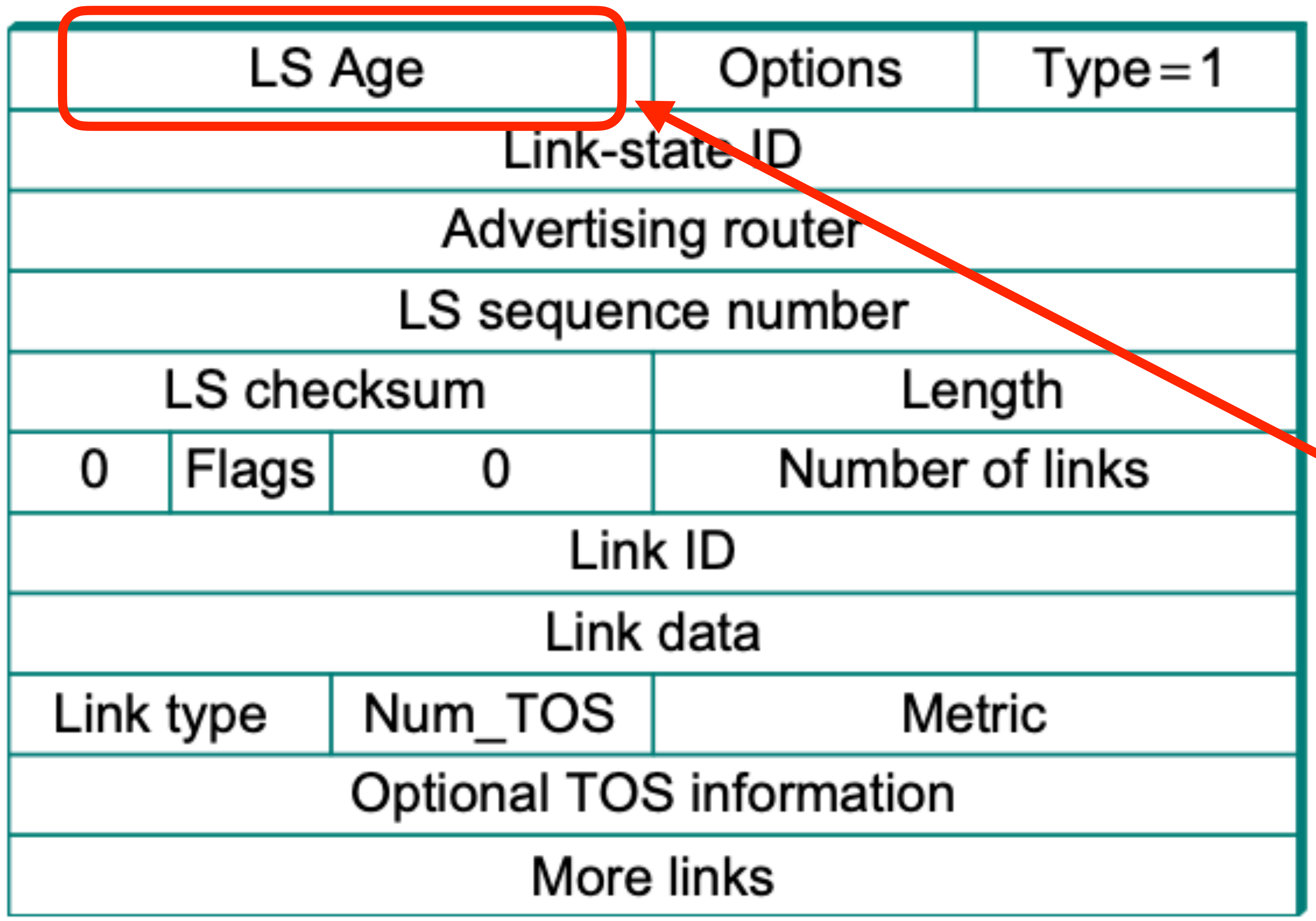


OSPF link-state advertisement

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

OSPF: Link State Packet



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

Link State v.s. Distance Vector

Link State v.s. Distance Vector

- Link state routing
  - High messaging overhead
  - Computing complexity

- Distance vector
  - Slow convergence
  - Race conditions

Communication Cost: A Non-trivial Metric

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: the number of hops

- #2: original ARPANET metric
  - link cost == the number of packets enqueued on each link
  - Take latency and bandwidth into consideration

Metrics for Link Cost

- #1: the number of hops

- #2: original ARPANET metric
  - link cost == the number of packets enqueued on each link
  - Take latency and bandwidth into consideration

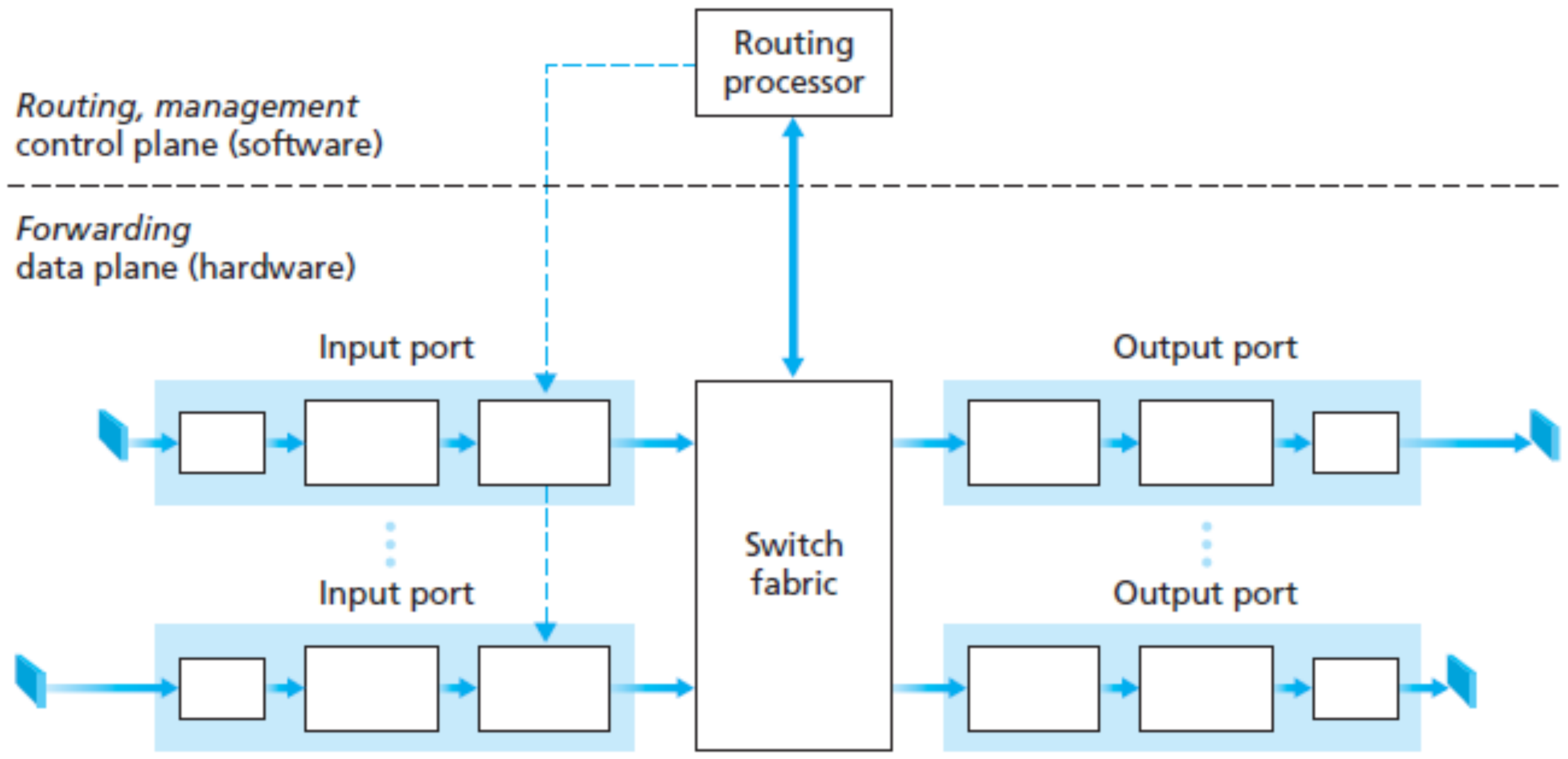
This (#2) moves packets towards the shortest queue, not the destination!!

Metrics for Link Cost (Cont’d)

- #3: new ARPANET metric
  - link cost == the average delay over some time period
  - Sample each incoming packet with its arrival time (**AT**)
  - Record the 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 the departure time for retransmission

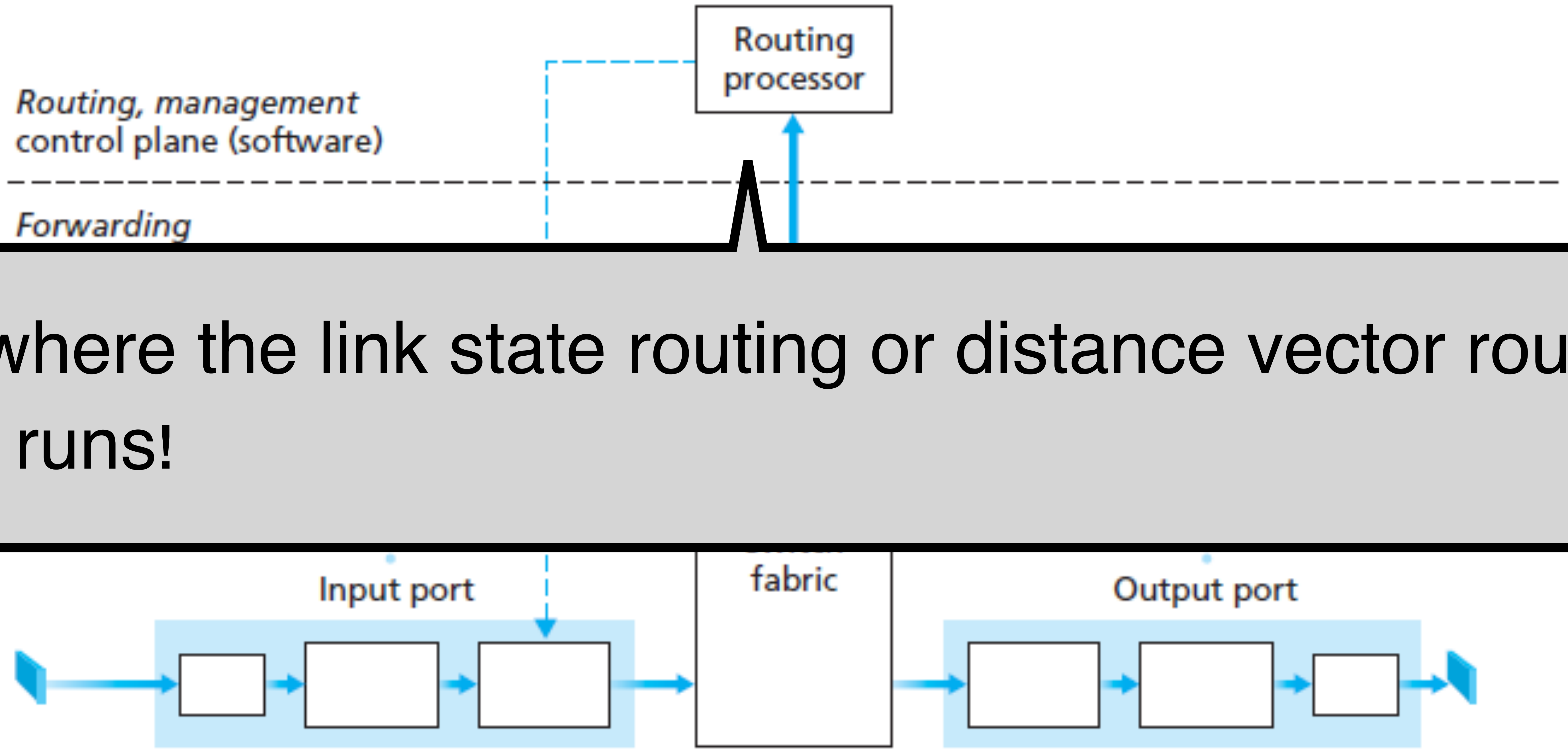
Recap: The Router Architecture—Routing Processor

- Routing Processor:
  - Execute the routing protocols
  - Maintain routable tables and attached link state information
  - Compute the forwarding table for the router



Recap: The Router Architecture—Routing Processor

- Routing Processor:
  - Execute the routing protocols
  - Maintain routable tables and attached link state information
  - Compute the forwarding table for the router



There is where the link state routing or distance vector routing algorithm runs!

Summary

- Today
  - Link state routing
- Next lecture
  - Software-Defined Networking