Introduction to Computer Networks

CS640

Distance Vector Routing

https://pages.cs.wisc.edu/~mgliu/CS640/S25/index.html

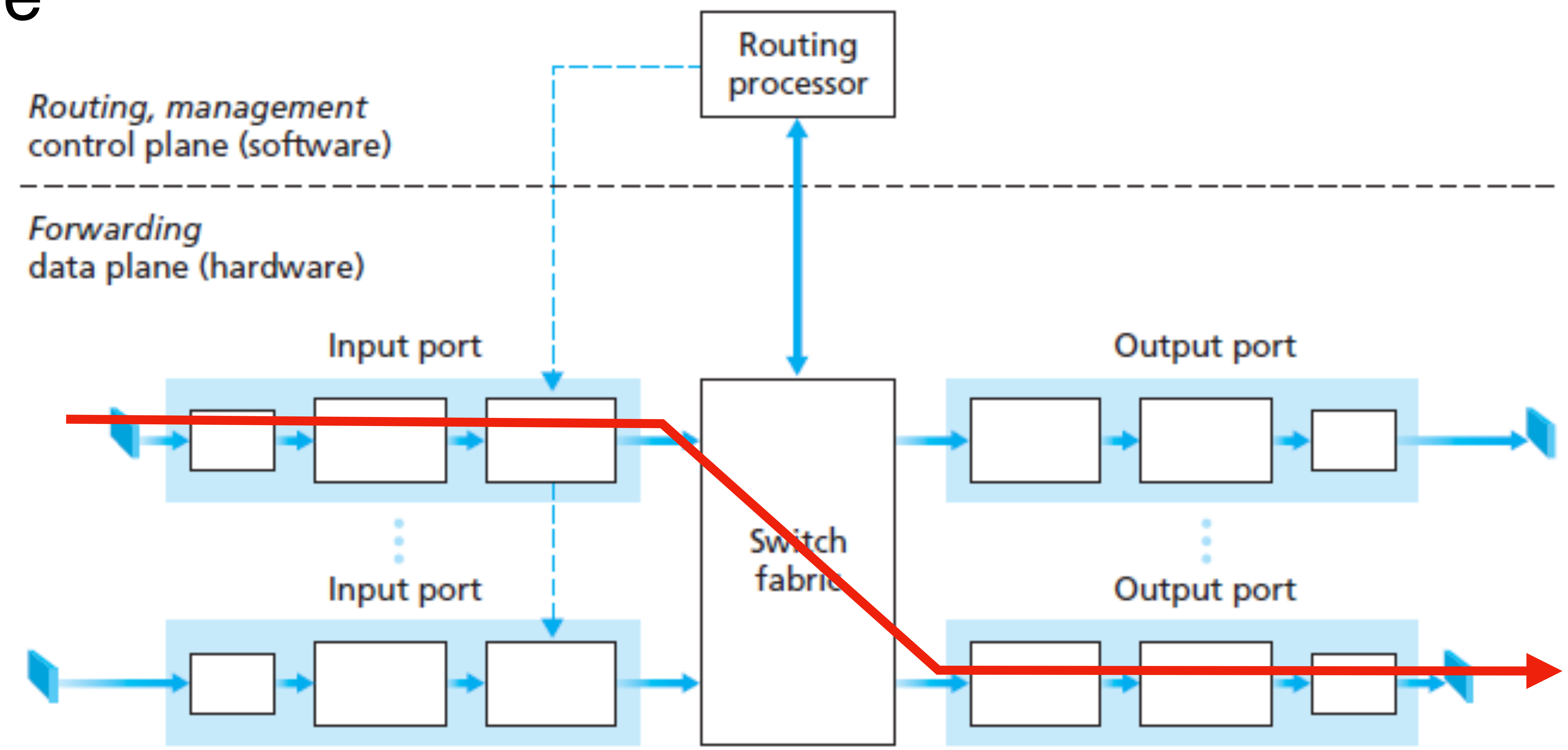
Ming Liu
mgliu@cs.wisc.edu

Outline

- Last
  - Efficient Addressing
- Today
  - Distance Vector Routing
- Announcements
  - Lab2 due on 03/04/2025 12:01PM
  - Quiz2 in-class next Thursday (03/06/2025)

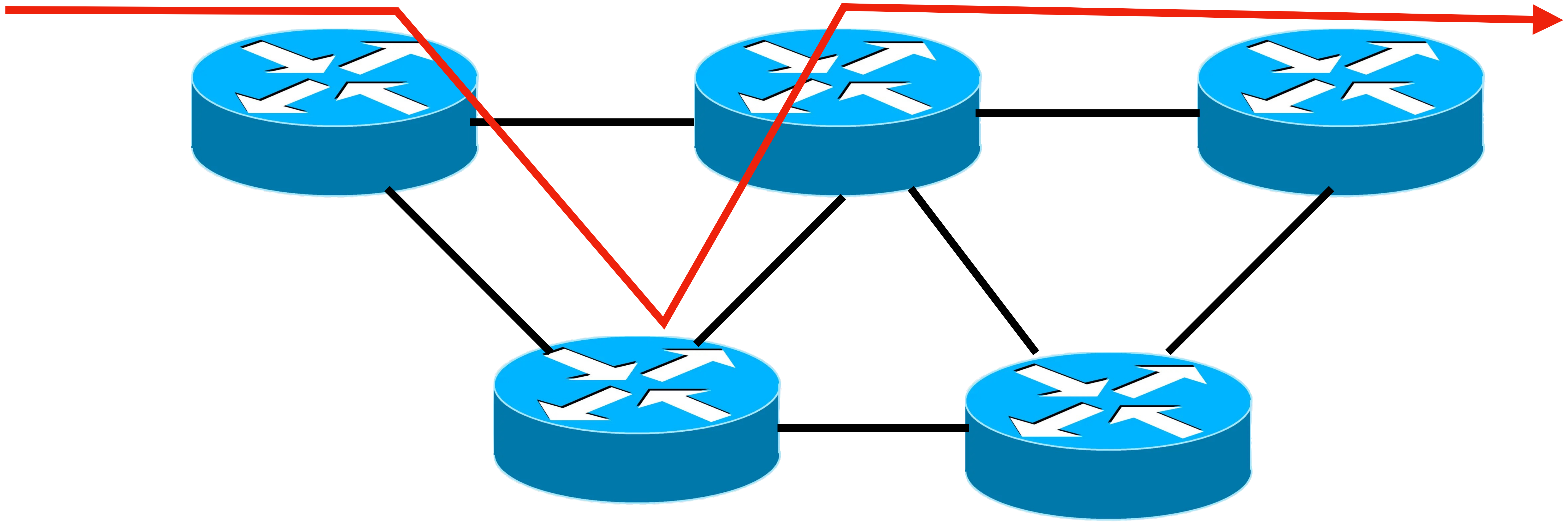
Recap: Forwarding v.s. Routing

- Forwarding refers to the router-local action of transferring a packet from an input interface to the appropriate output interface
  - Usually implemented in the hardware
  - O(nanosecond)
  - Data plane



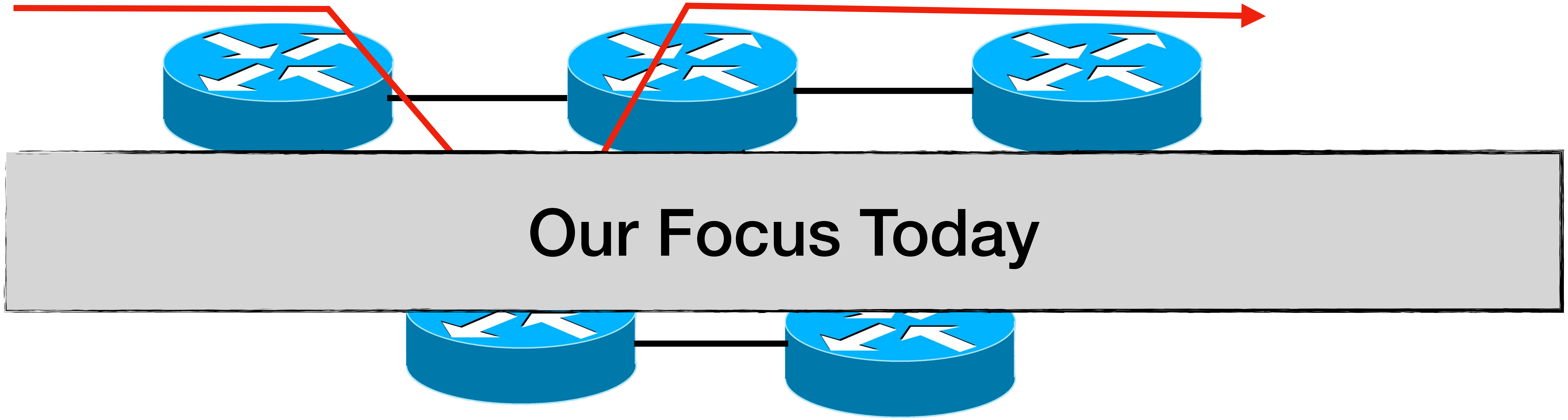
Recap: Forwarding v.s. Routing

- Routing refers to the network-wide process that determines the end-to-end paths that packets take from source to destination
  - Usually implemented in the software (and hardware)
  - O(second)
  - Control plane

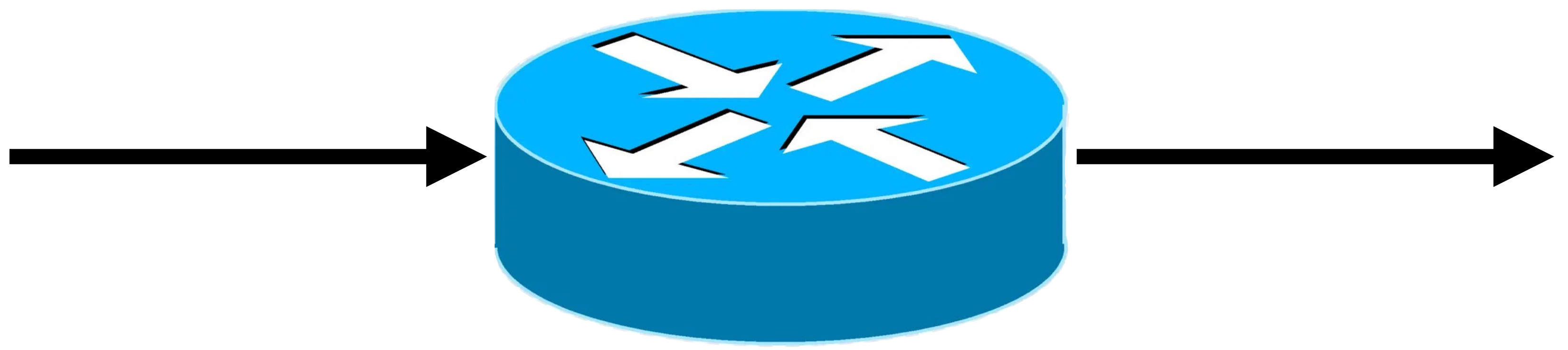


Recap: Forwarding v.s. Routing

- Routing refers to the network-wide process that determines the end-to-end paths that packets take from source to destination
  - Usually implemented in the software (and hardware)
  - O(second)
  - Control plane

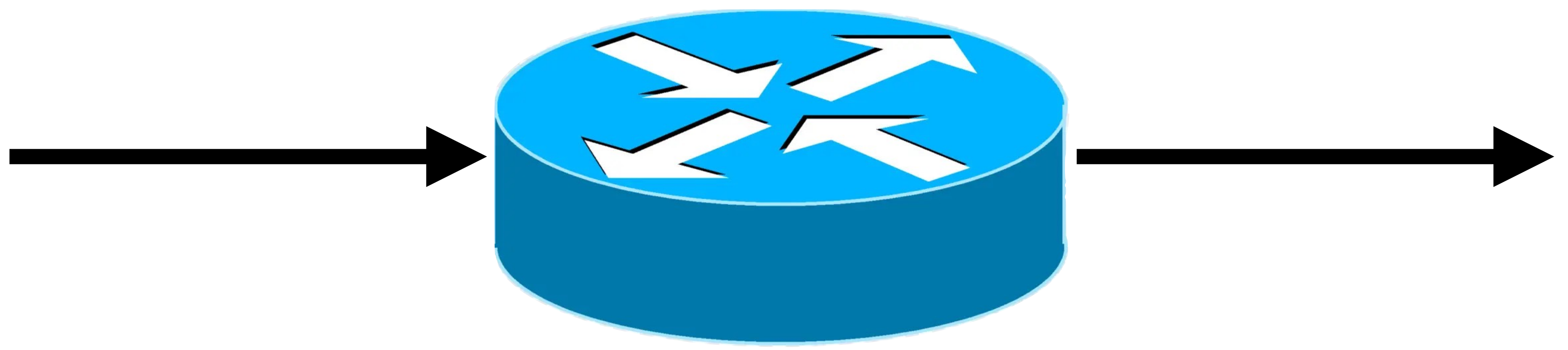


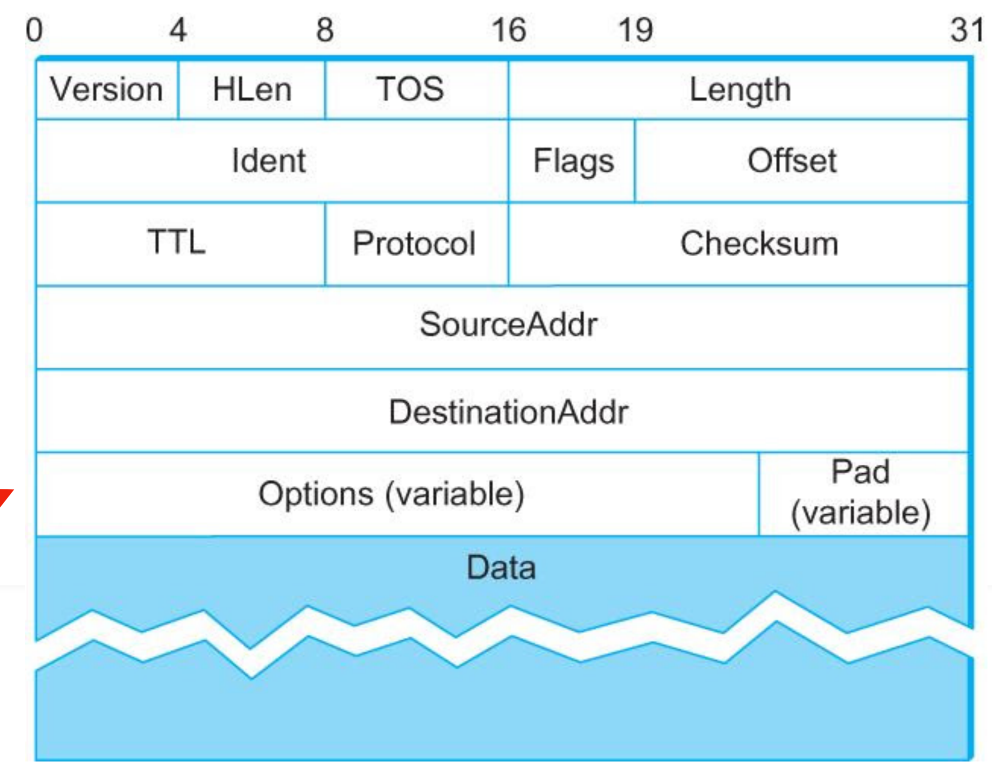
Routing Logics



```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

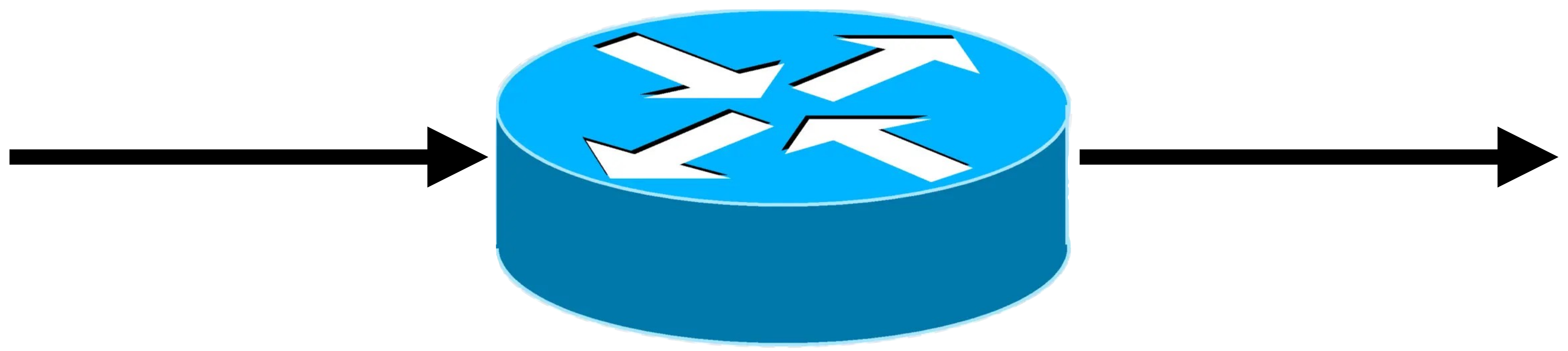
Routing Logics






```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

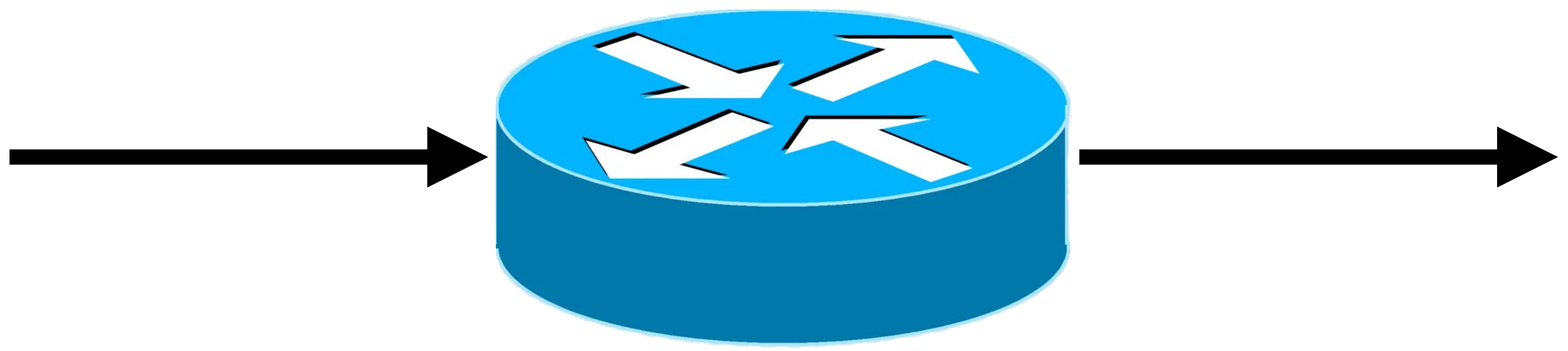
Routing Logics



Lookup Routing Table

```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

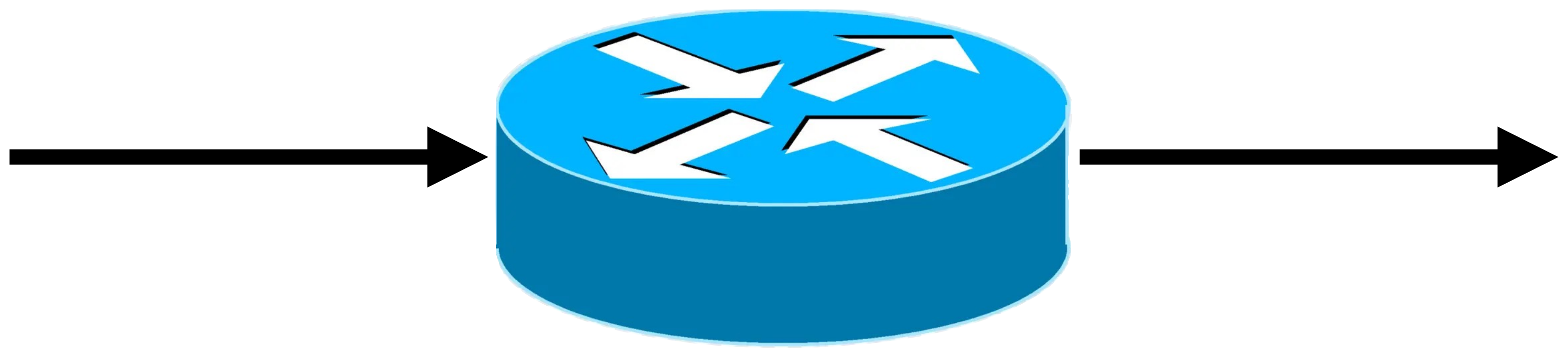
Routing Logics



```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

Compute Subnet Number

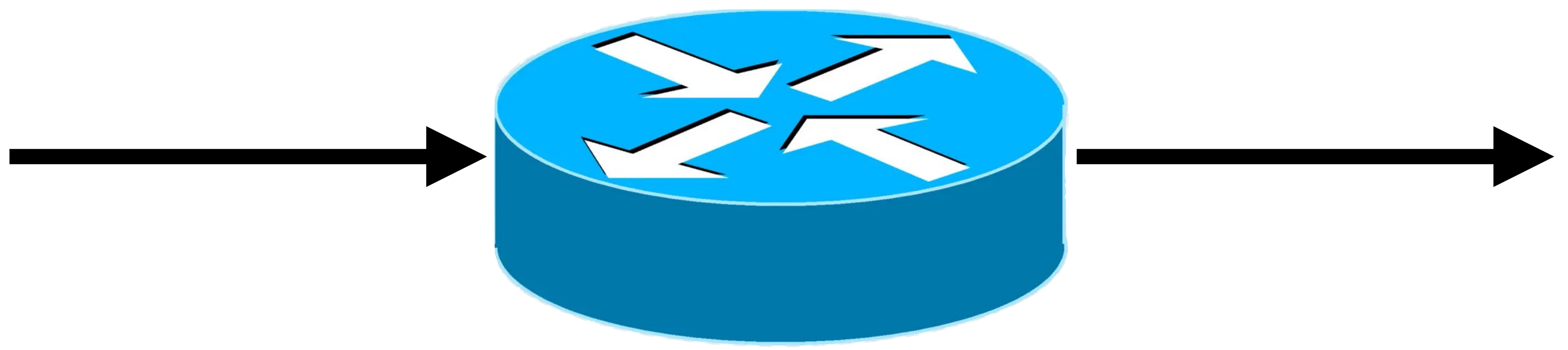
Routing Logics



A matched entry is found

```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

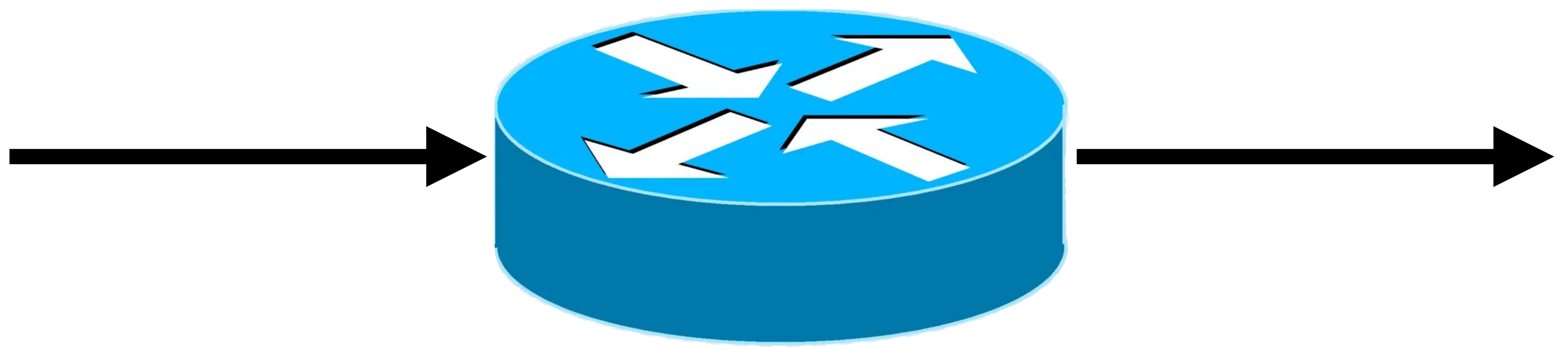
Routing Logics



```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

Hosts connects to the router directly

Routing Logics



```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

Send to the next router

How can we build the routing table?

```
D = destination IP address
for each entry (SubnetNum, SubnetMask, NextHop)
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to D
        else
            deliver datagram to NextHop
```

Routing Algorithm/Protocol

- Represent connected networks as a graph
  - Vertices in the graph are routers
  - Edges in the graph links

- Links (Edges) have communication cost, which can be quantized
  - E.g., physical distance, latency, throughput, etc.

Routing Challenges

- #1: Network hardware fabric is dynamic
  - Links and routers can fail
- #2: Network traffic is dynamic
  - A router or link can be overloaded
- #3: The communication cost is dynamic
  - The quantized value depends on both physical and runtime properties
- #4: There is no central view
  - Protocols must work in a distributed fashion

Naive Approach

- Static configuration
  - Calculate preferred paths mannually
  - Fill them into the routing table

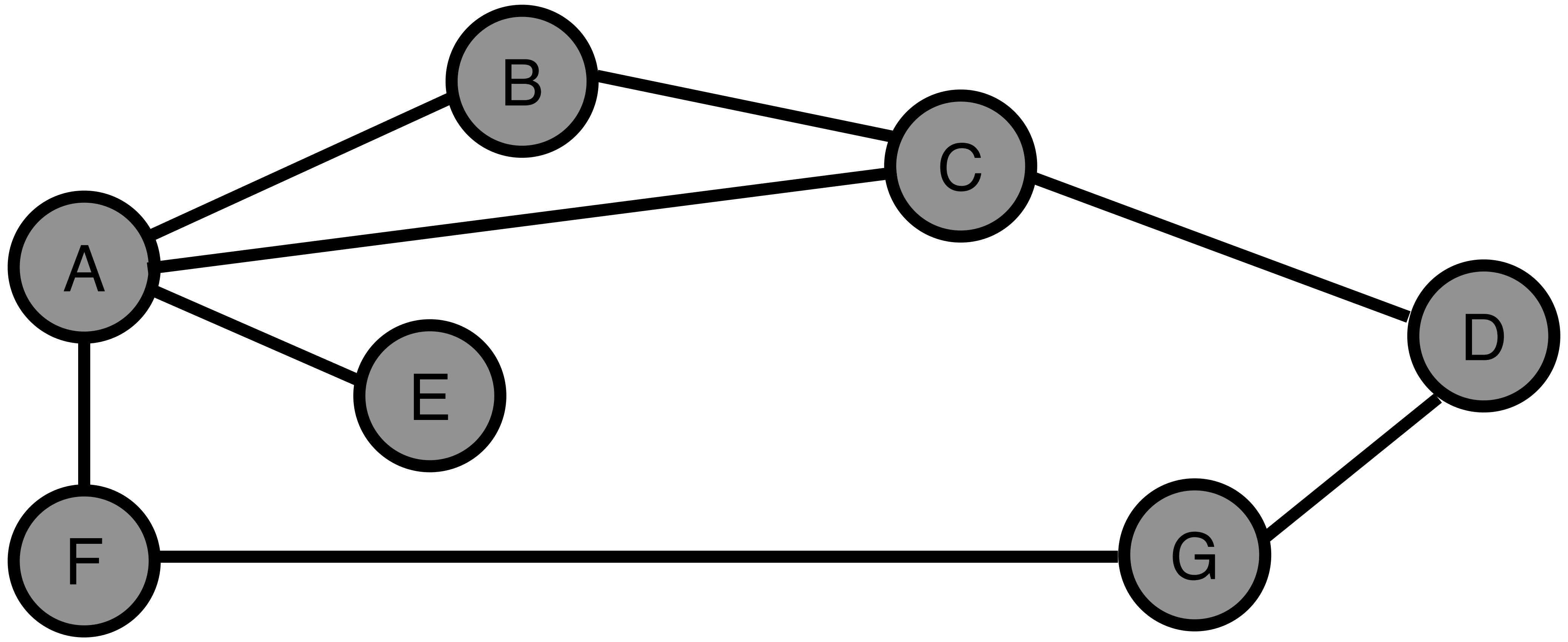
- Drawbacks
  - No adaptation
  - Unable to scale

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

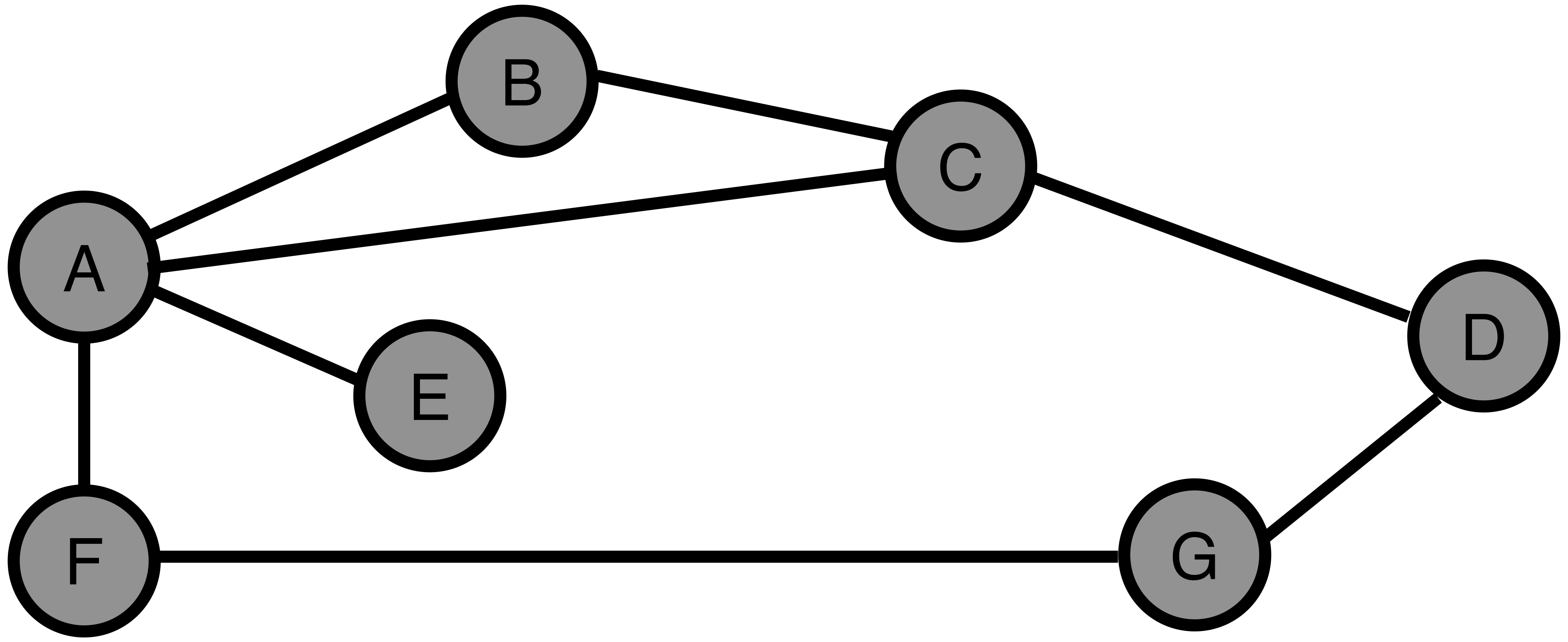
- Assumption
  - Each router knows the cost of the link to its directly connected neghbors

Distance Vector Protocol



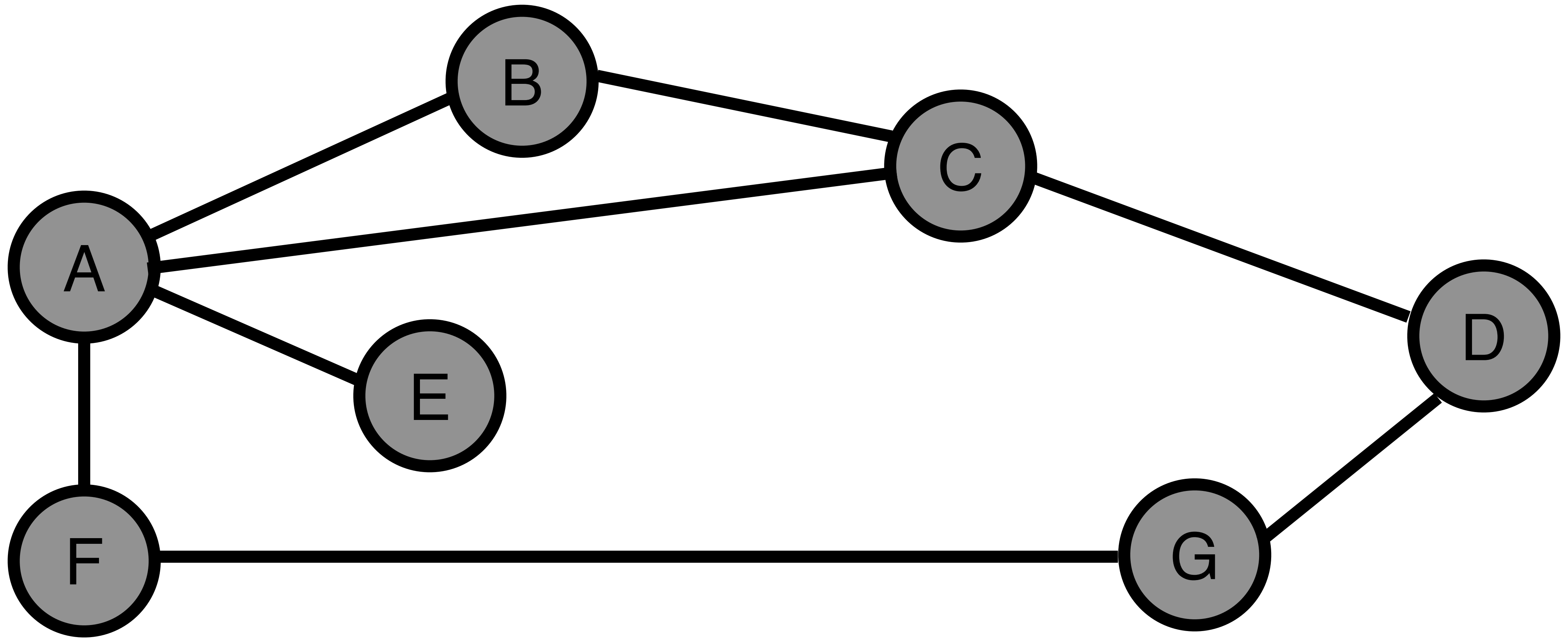
Distance is defined as the number of hops

Step 1: Figure Out Initial Distance



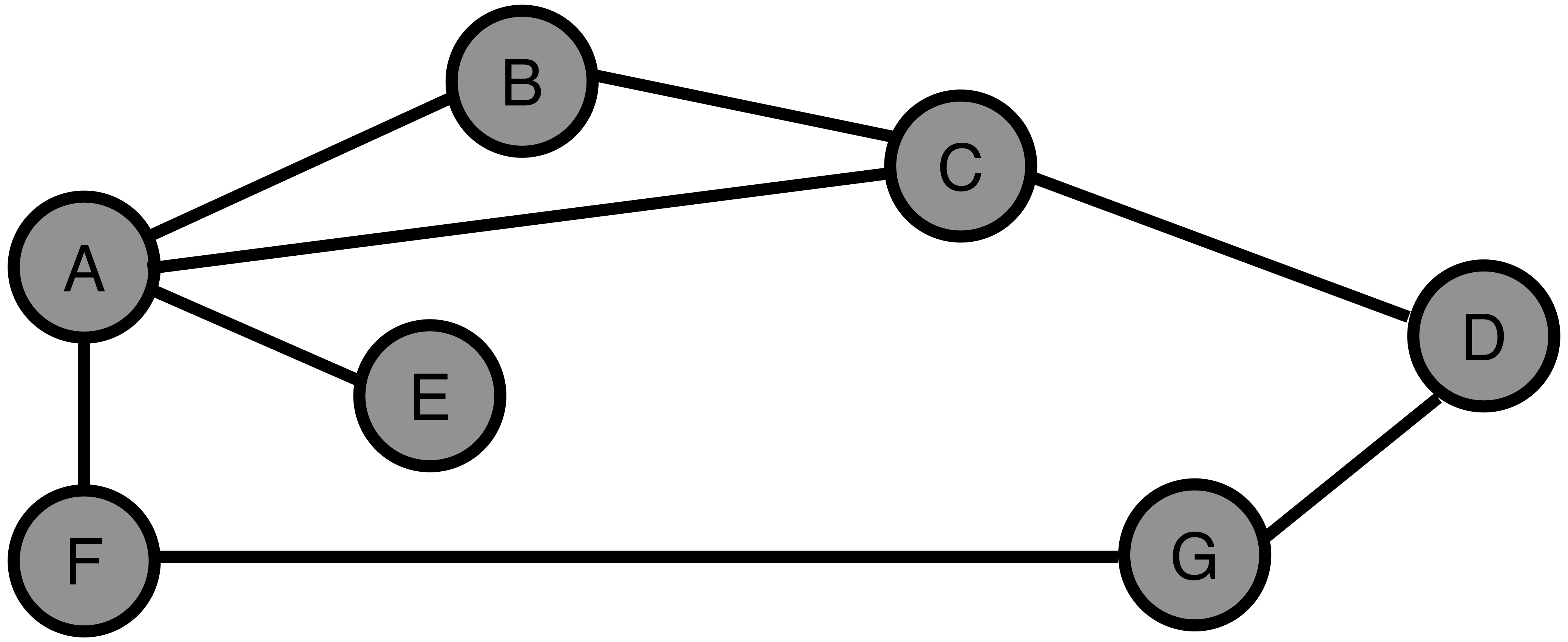
	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A							
B							
C							
D							
E							
F							
G							

Step 1: Figure Out Initial Distance



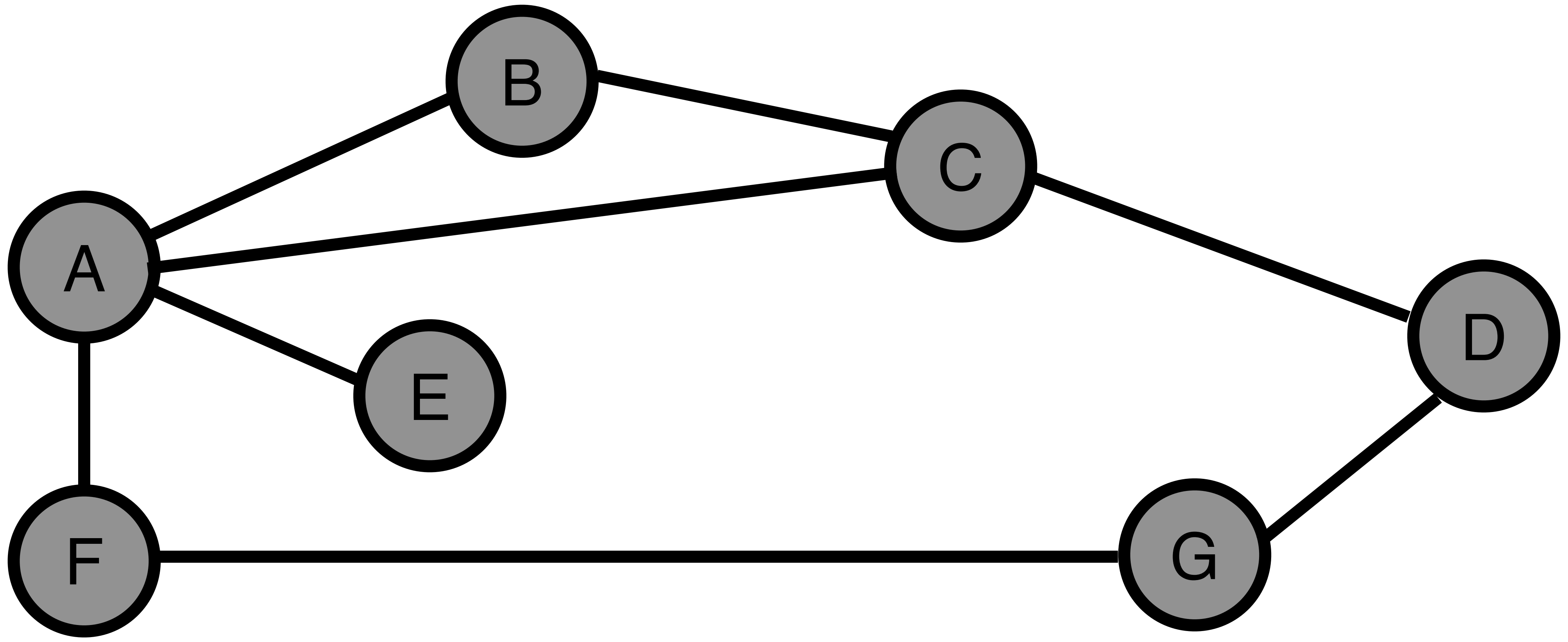
	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B							
C							
D							
E							
F							
G							

Step 1: Figure Out Initial Distance



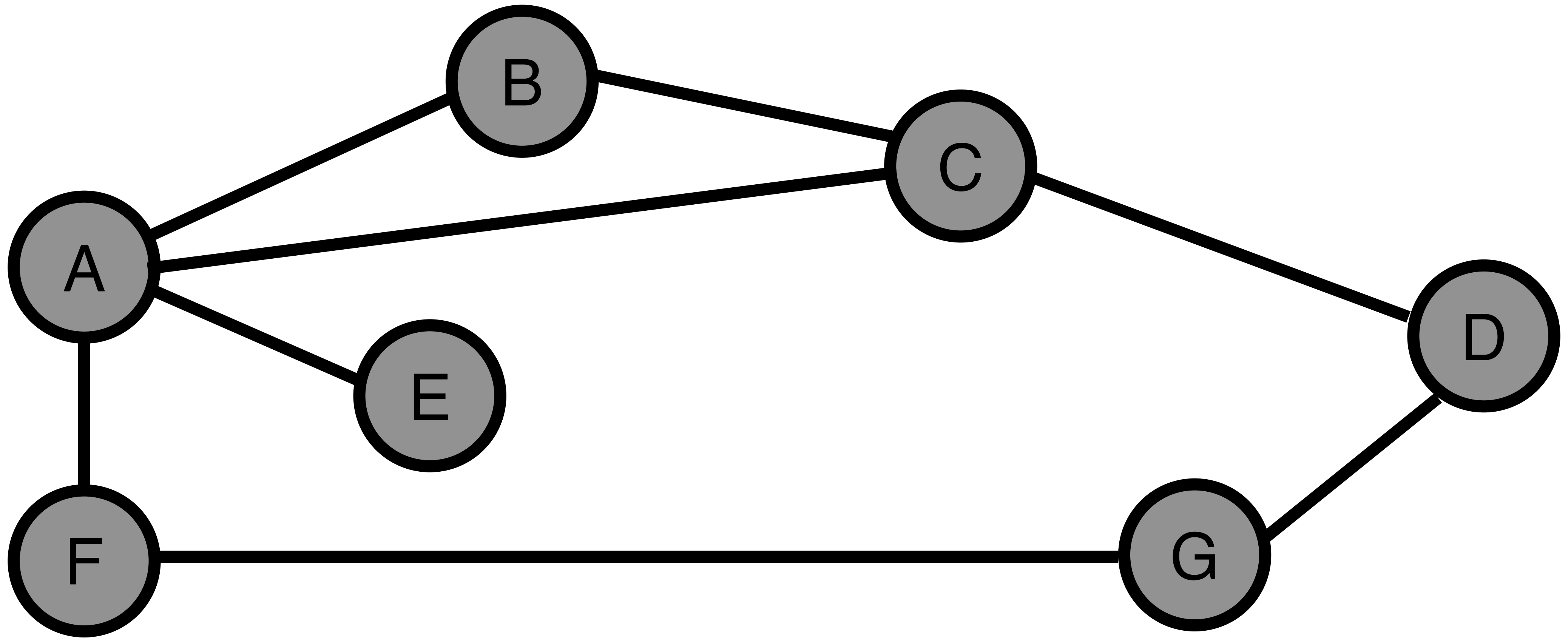
	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C							
D							
E							
F							
G							

Step 1: Figure Out Initial Distance



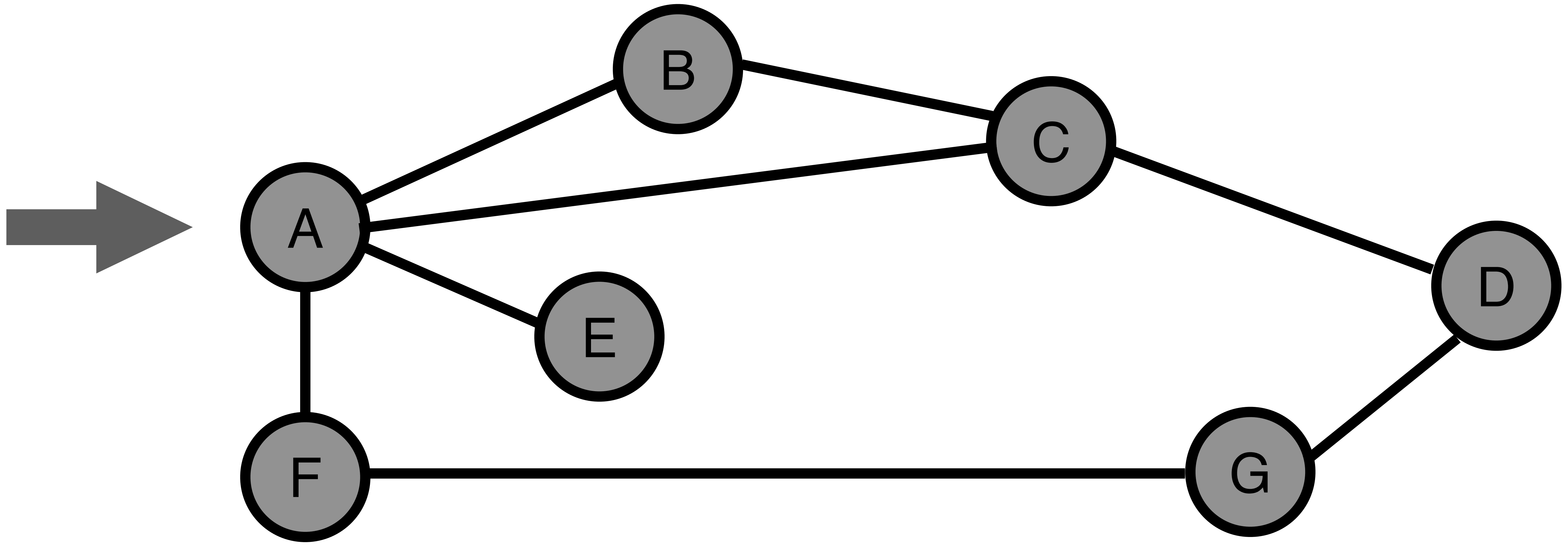
	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D							
E							
F							
G							

Step 1: Figure Out Initial Distance



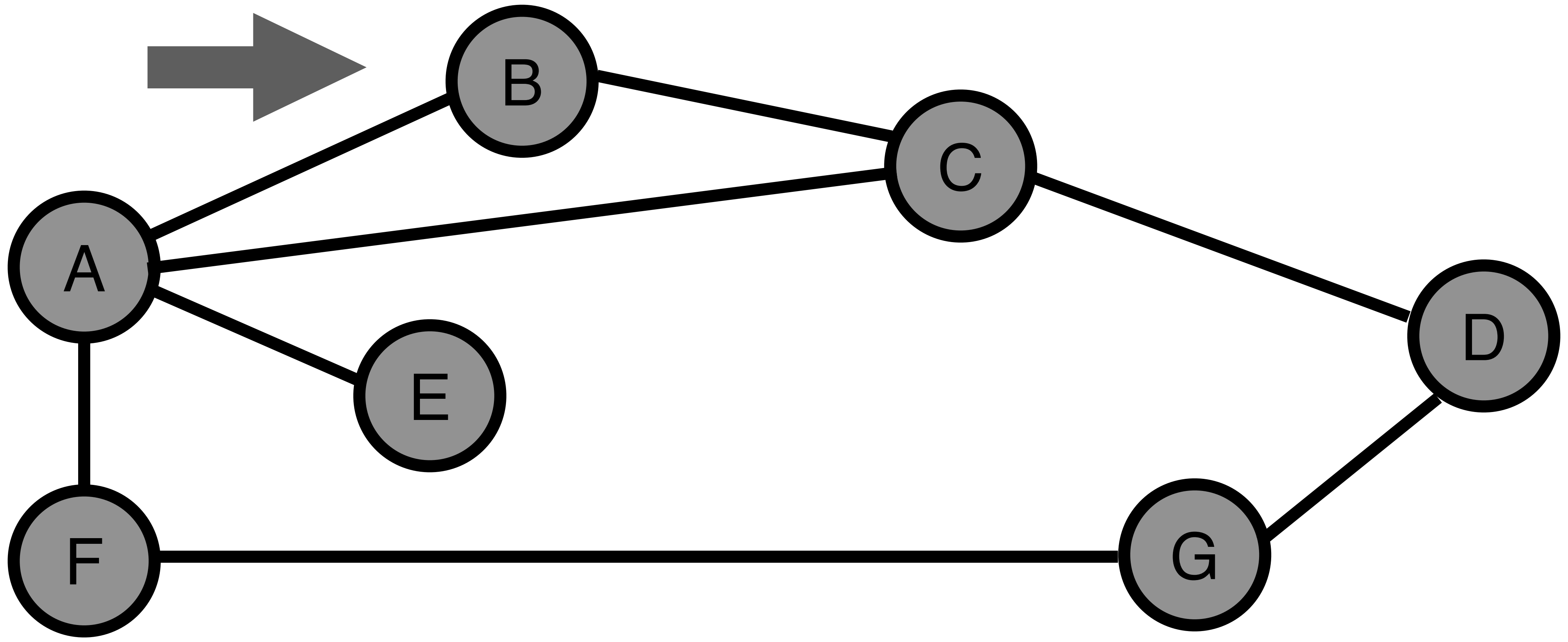
	Distance to Reach Node (Global View)						
	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Initial Routing Table



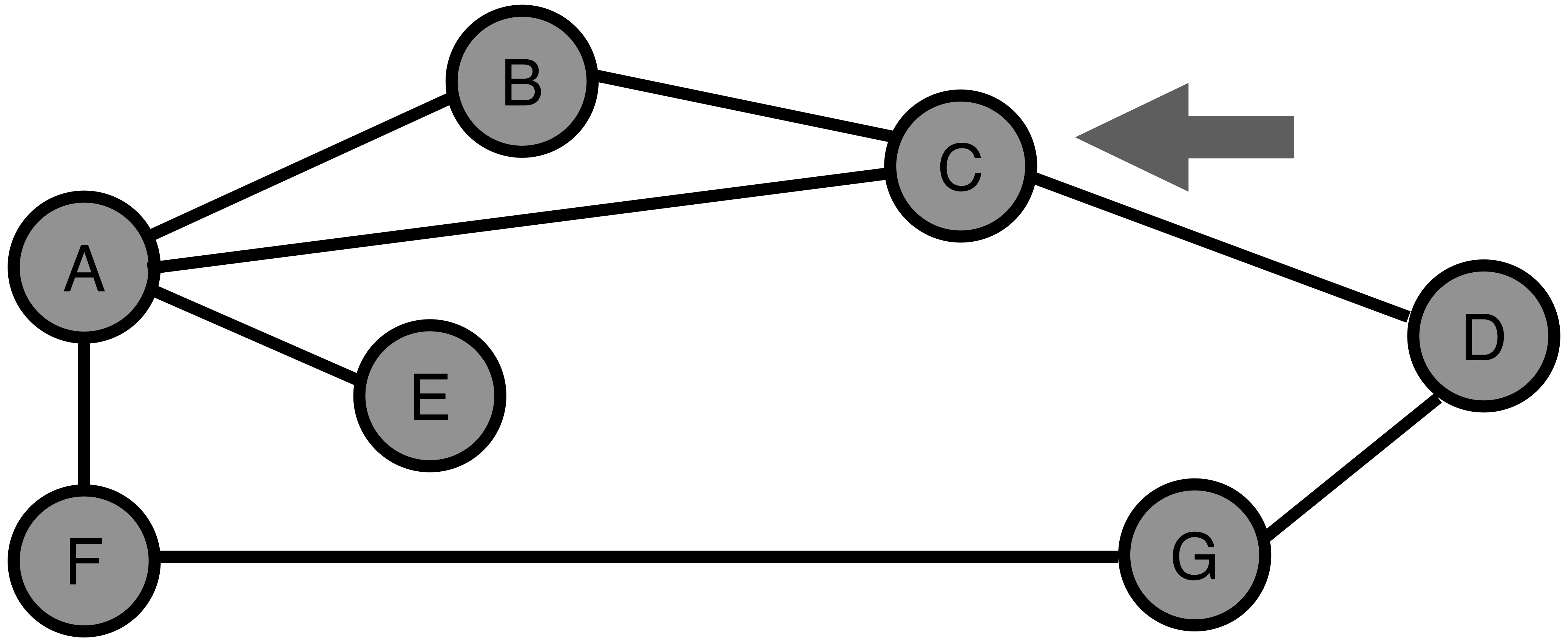
Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

Initial Routing Table



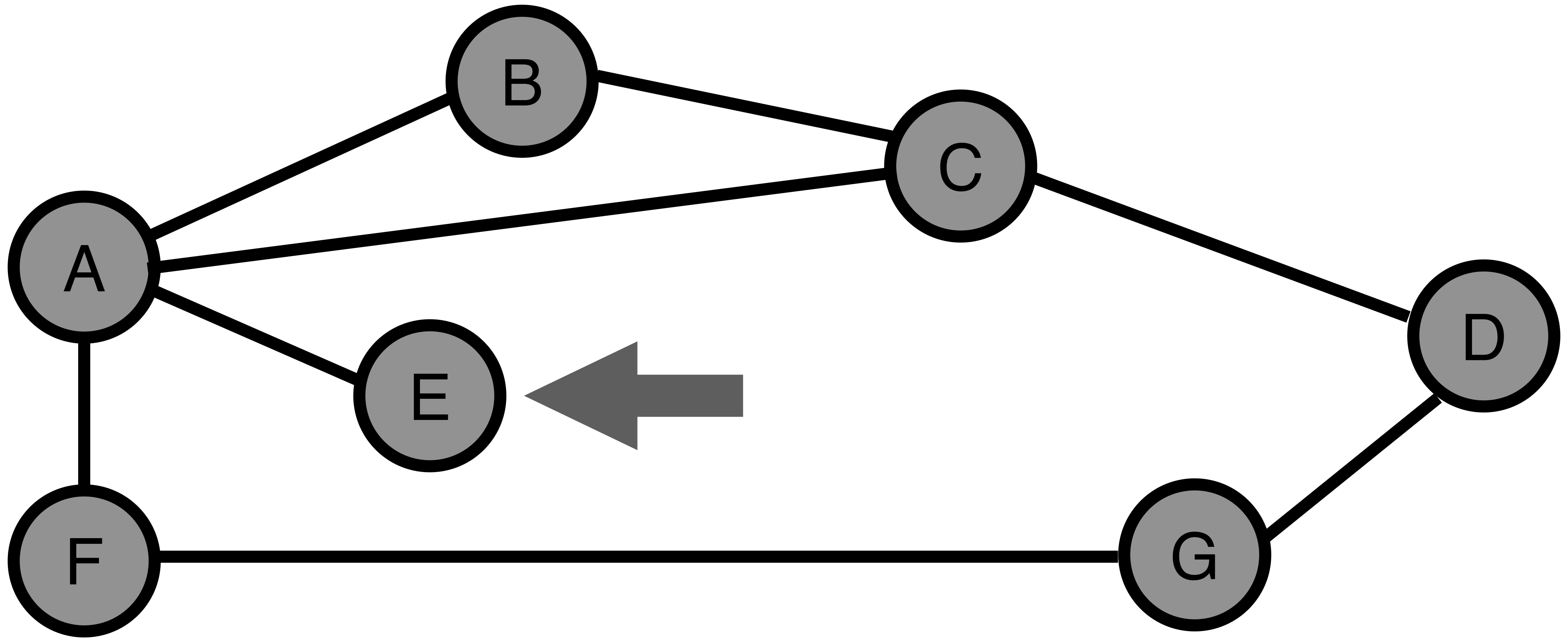
Destination	Cost	NextHop
A	1	A
C	1	C
D	∞	–
E	∞	–
F	∞	–
G	∞	–

Initial Routing Table



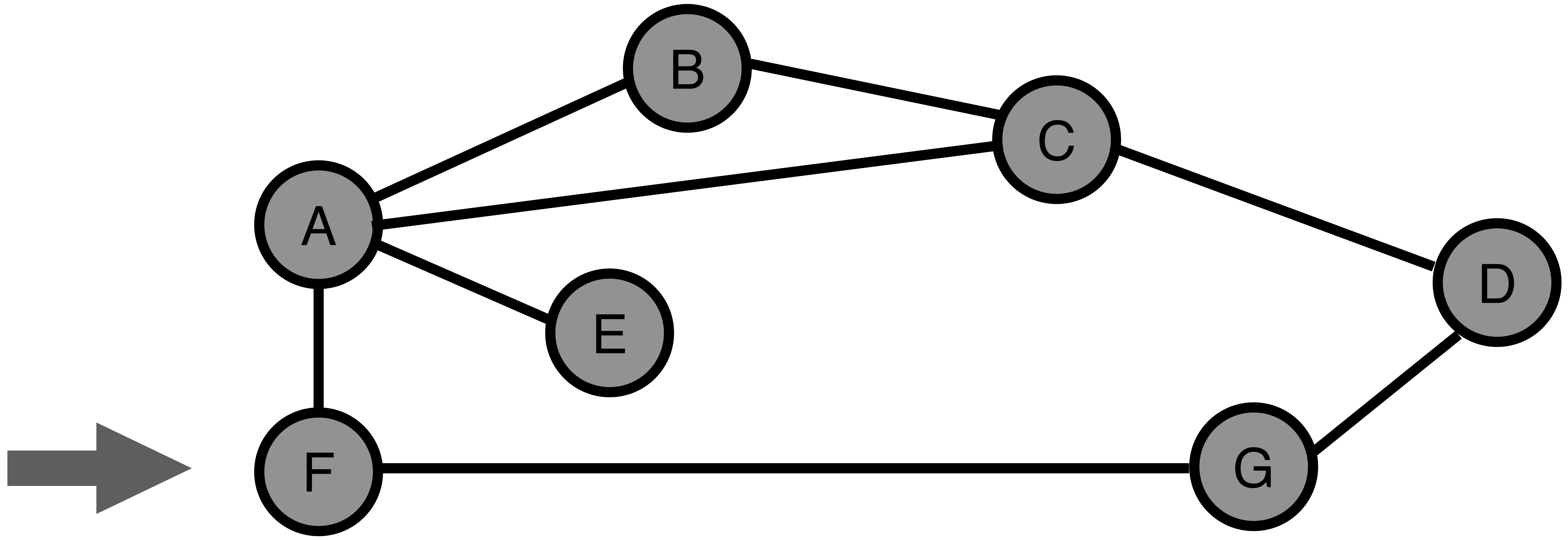
Destination	Cost	NextHop
A	1	A
B	1	B
D	1	D
E	∞	-
F	∞	-
G	∞	-

Initial Routing Table



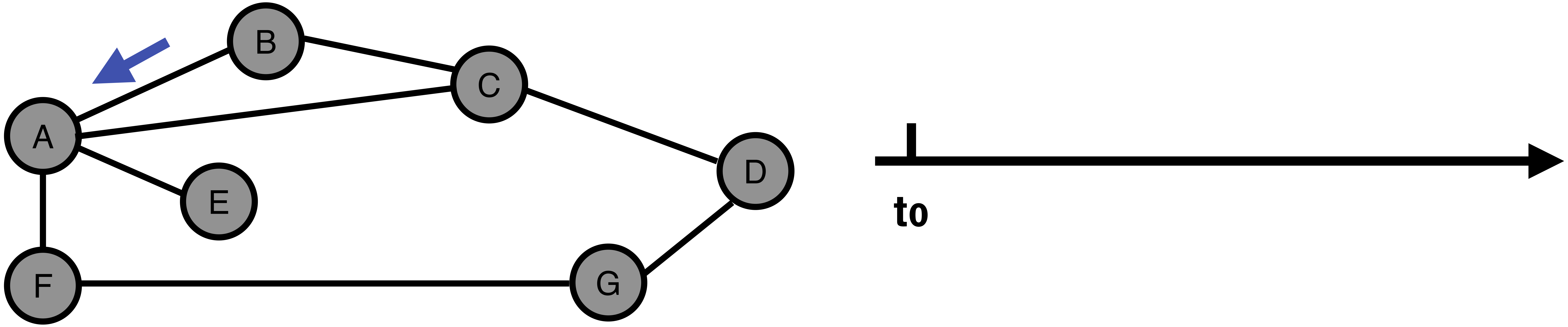
Destination	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
F	∞	–
G	∞	–

Initial Routing Table



Destination	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
E	∞	–
G	1	G

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

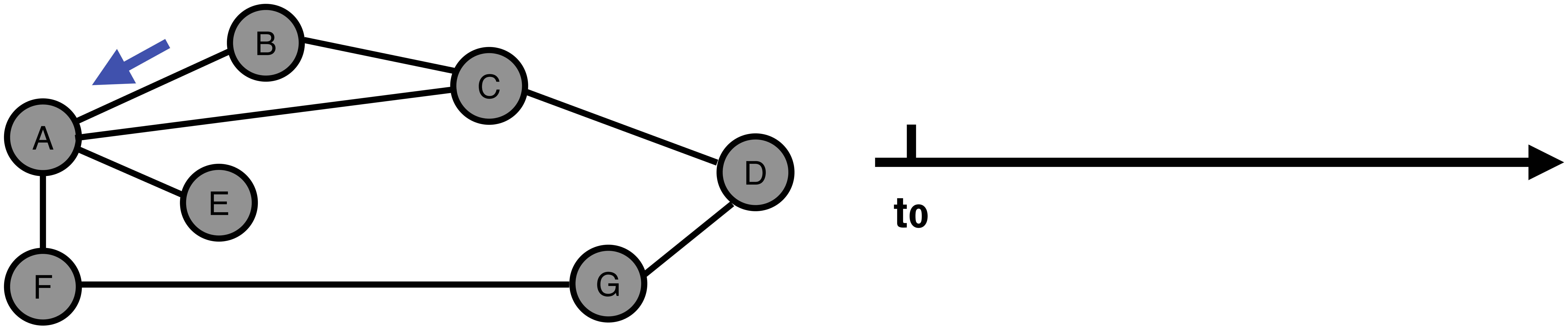
Dest.	Cost	NextHop
A	1	A
C	1	C
D	∞	–
E	∞	–
F	∞	–
G	∞	–

=

A

Dest.	Cost	NextHop
B		
C		
D		
E		
F		
G		

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

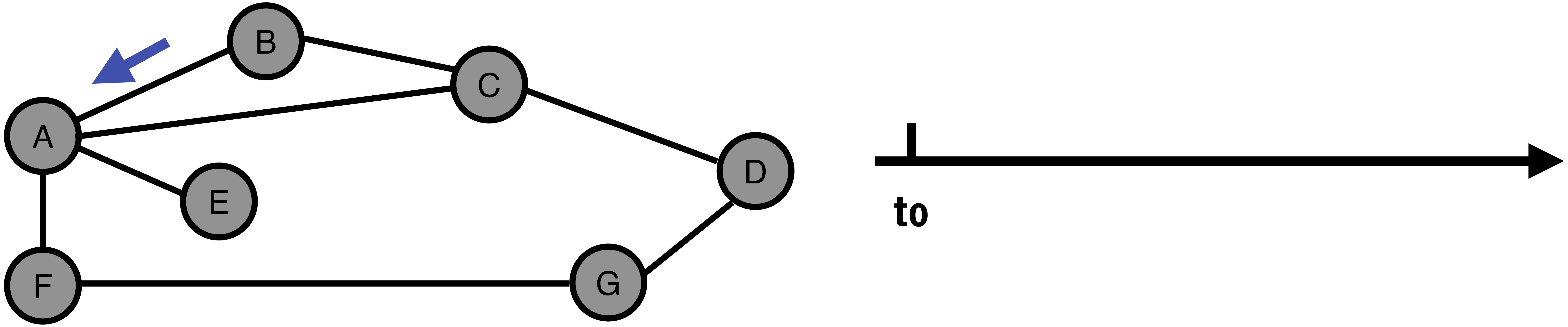
Dest.	Cost	NextHop
A	1	A
C	1	C
D	∞	–
E	∞	–
F	∞	–
G	∞	–

=

A

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

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

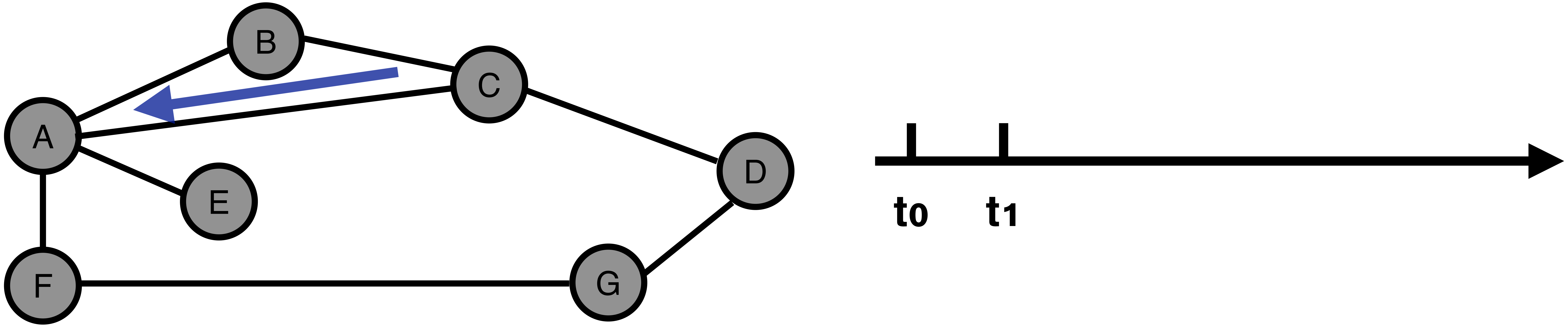
Dest.	Cost	NextHop
A	1	A
C	1	C
D	∞	-
E	∞	-
F	∞	-
G	∞	-

=

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	∞	–
E	1	E
F	1	F
G	∞	–

+

C

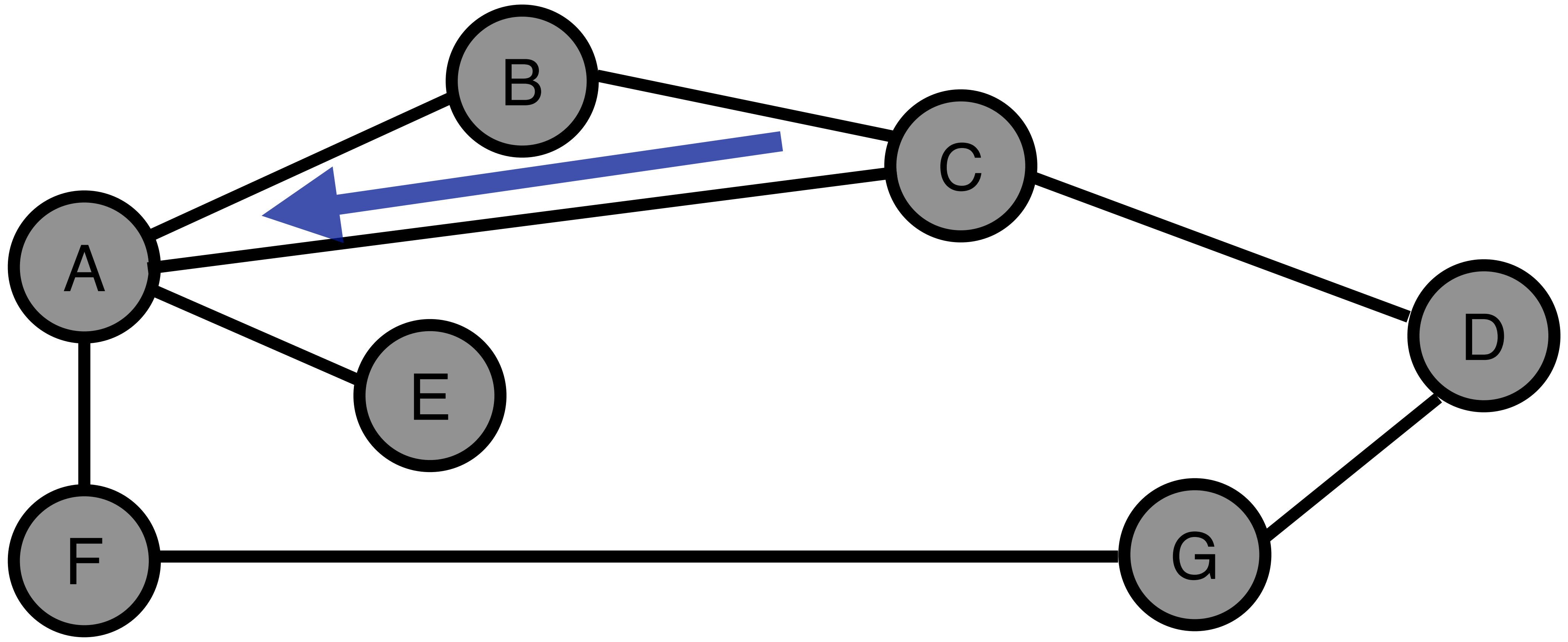
Dest.	Cost	NextHop
A	1	A
B	1	B
D	1	D
E	∞	–
F	∞	–
G	∞	–

=

A

Dest.	Cost	NextHop
B		
C		
D		
E		
F		
G		

Step 3+: Keep Exchange Vectors Until Stable



A

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

+

C

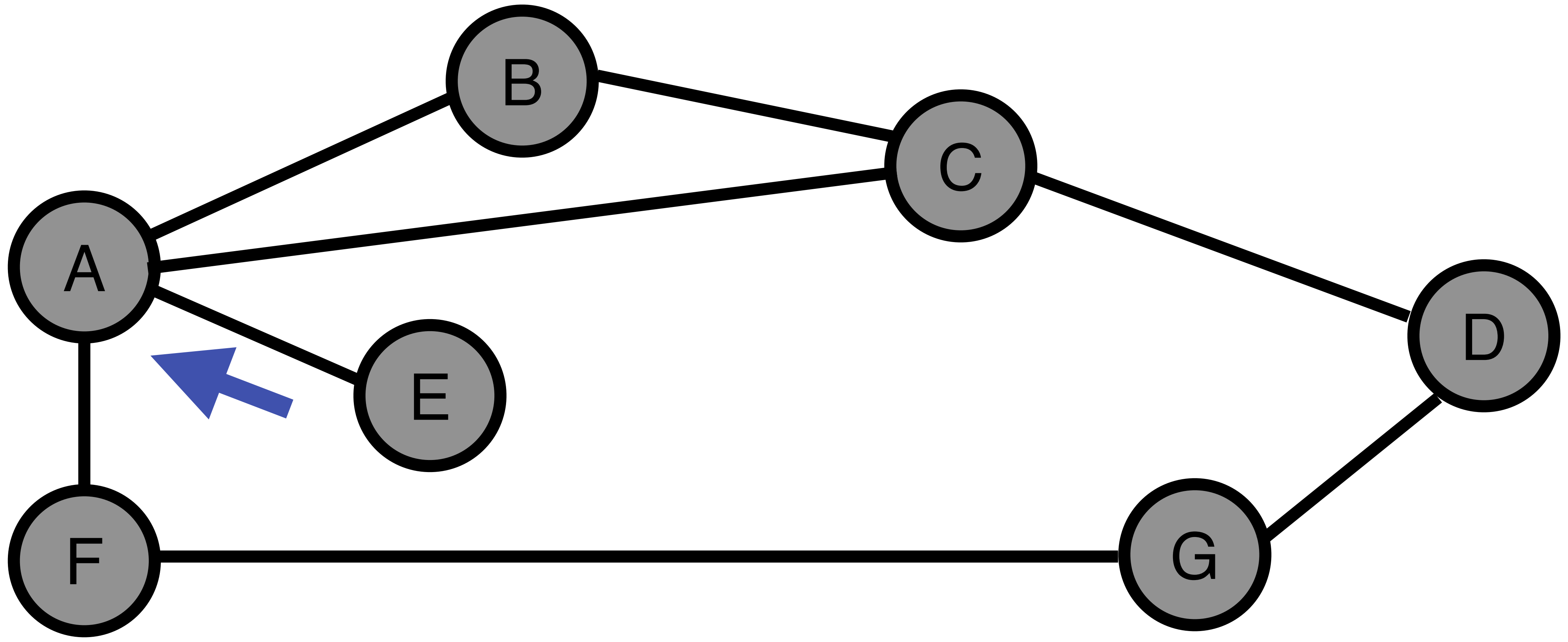
Dest.	Cost	NextHop
A	1	A
B	1	B
D	1	D
E	∞	–
F	∞	–
G	∞	–

=

A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
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	∞	–

+

E

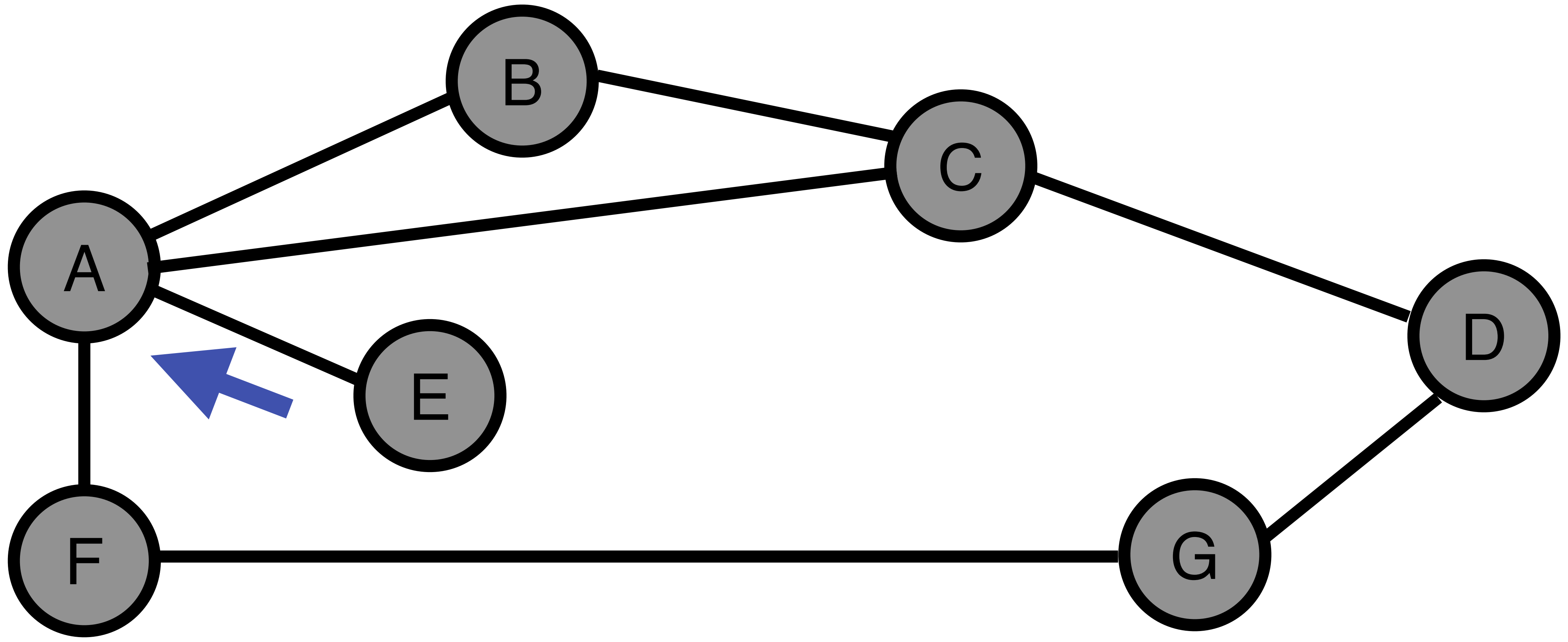
Dest.	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
F	∞	–
G	∞	–

=

A

Dest.	Cost	NextHop
B		
C		
D		
E		
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	∞	–

\+

E

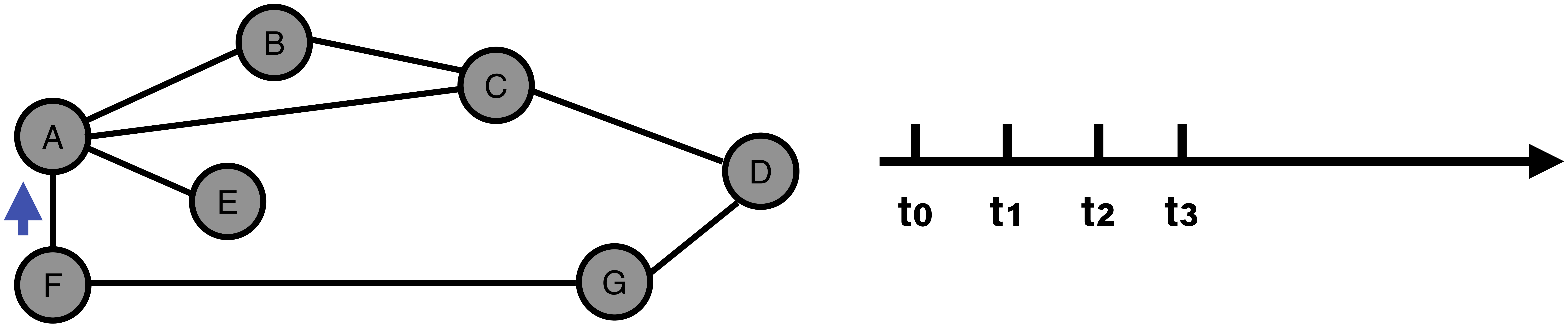
Dest.	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
F	∞	–
G	∞	–

=

A

Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
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

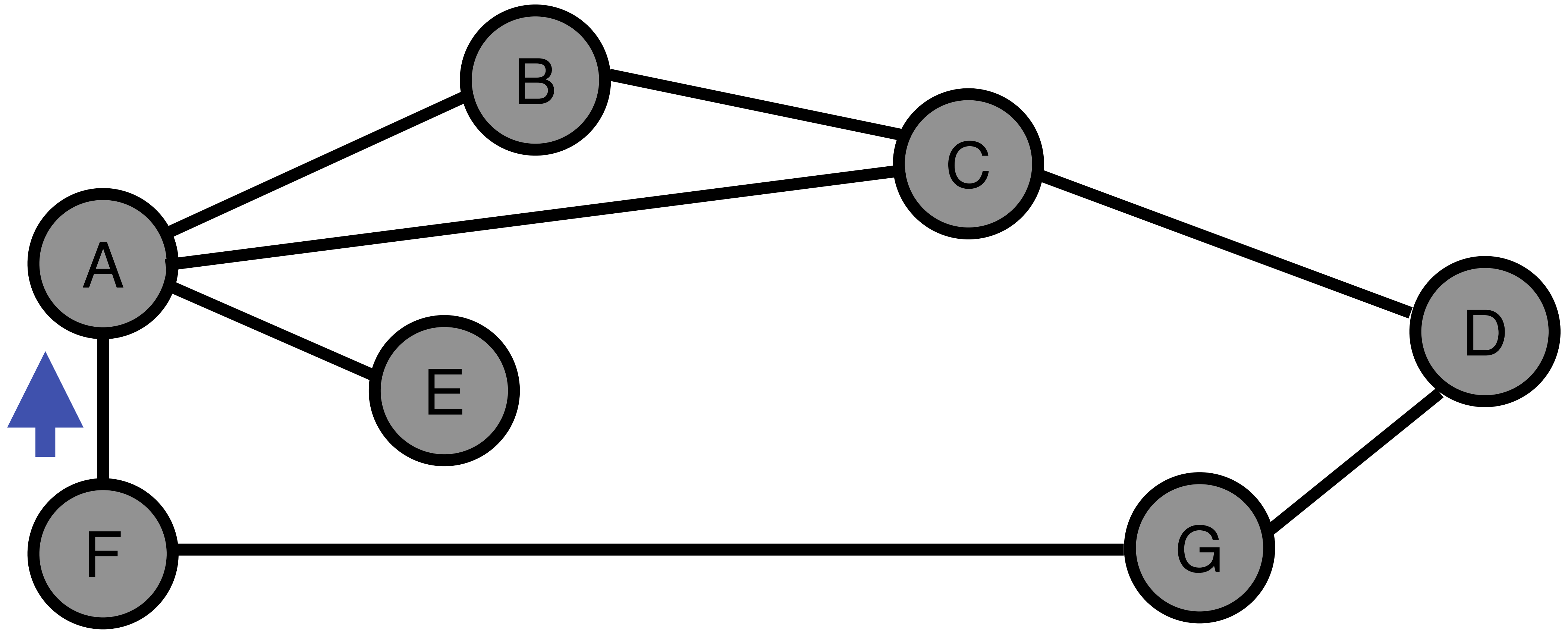
Dest.	Cost	NextHop
A	1	A
B	∞	–
C	∞	–
D	∞	–
F	0	F
G	1	G

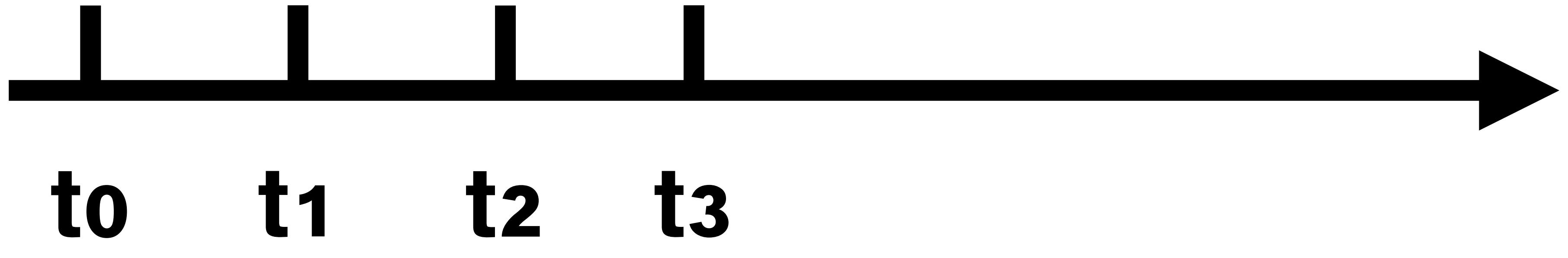
=

A

Dest.	Cost	NextHop
B		
C		
D		
E		
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

Route Selection

- New_Cost (node) = Cost(node) (from neighbor) + Cost (node-neighbor)
  - Cost (node-neighbor) = 1 in the above discussion
  - If next_hop (new) == next_hop (old)
    - Cost = New_Cost
  - Else
    - Cost = Min (New_Cost, Old_Cost)
    - Update the next hop to the neighbor node

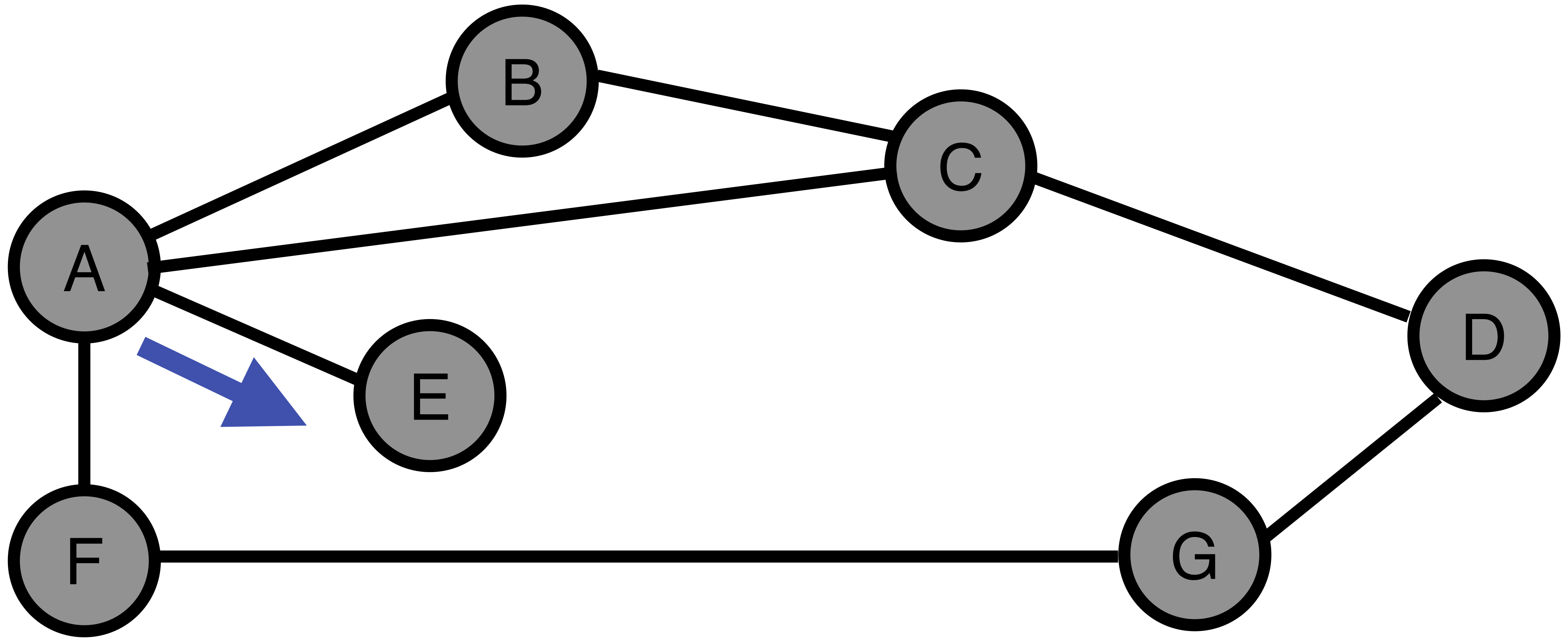
- The routing table is evolving
  - Based on the event sequence

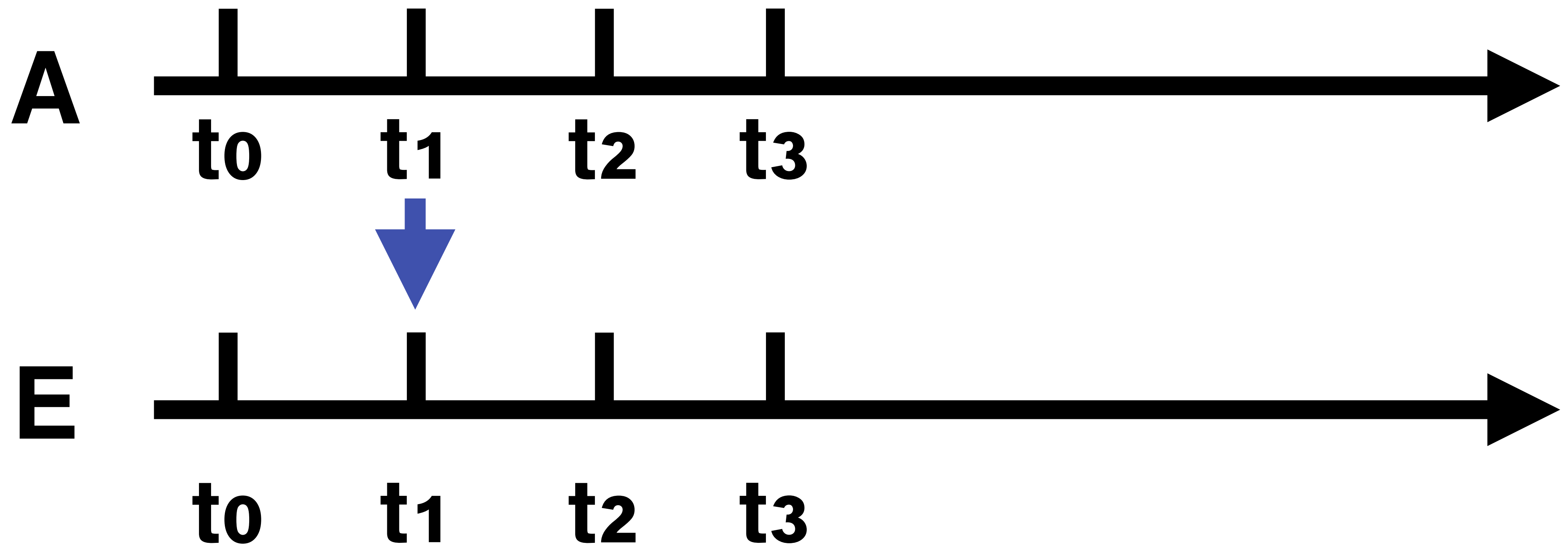
Dest		
B		
C		
D		
E		
F	1	F
G	∞	-

F	0	F
G	1	G

F	1	F
G	2	F

Routing Table Keeps Evolving





E

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

\+

A (t1)

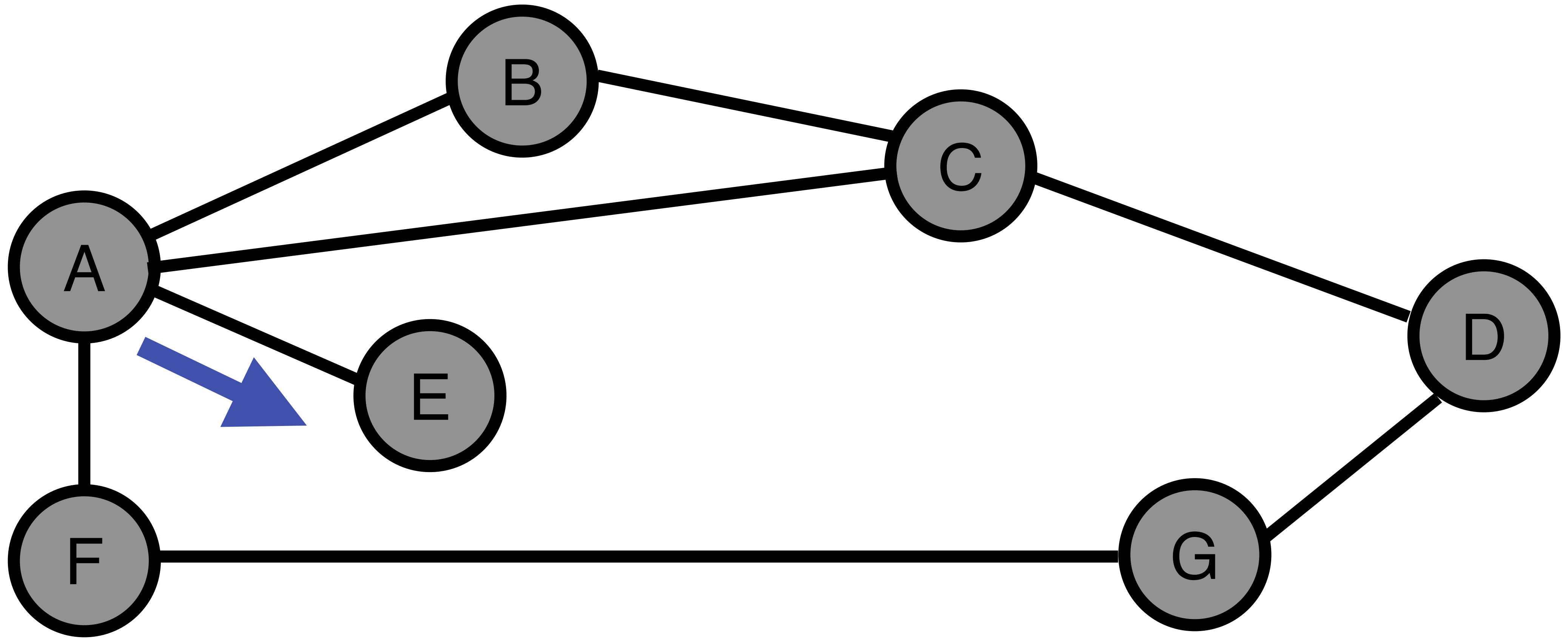
Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	∞	–

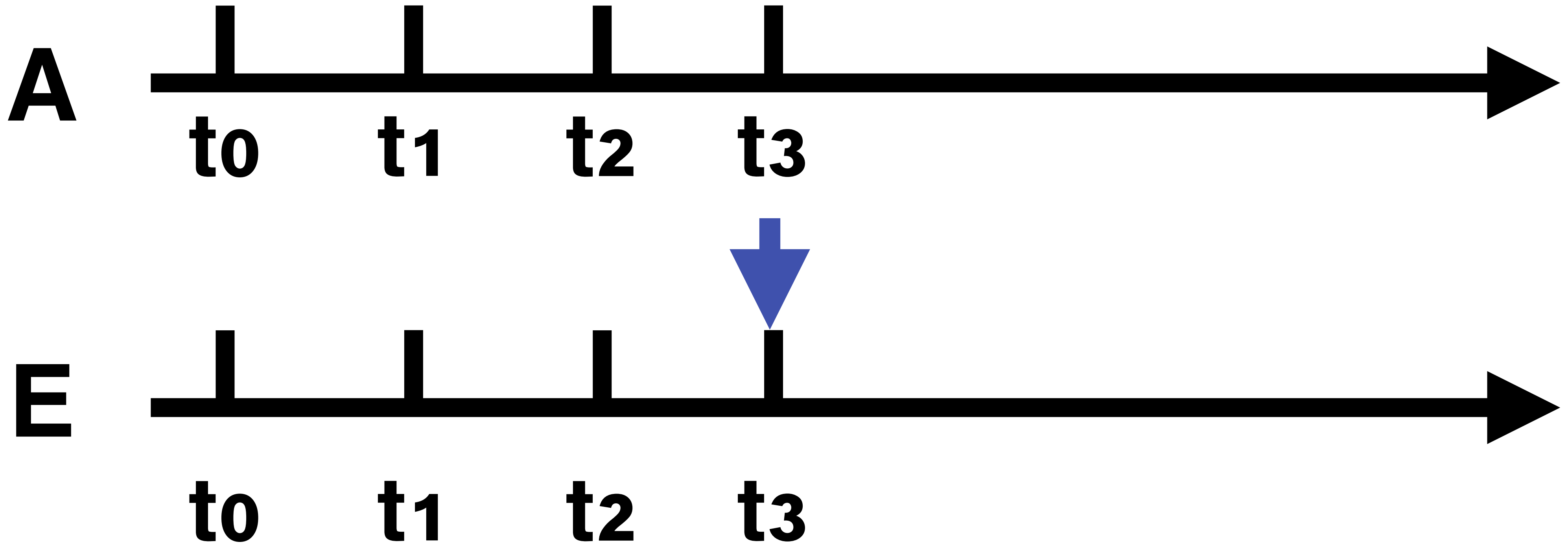
=

E

Dest.	Cost	NextHop
A	1	A
B	2	A
C	2	A
D	3	A
F	2	A
G	∞	–

Routing Table Keeps Evolving





E

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

+

A (t_3)

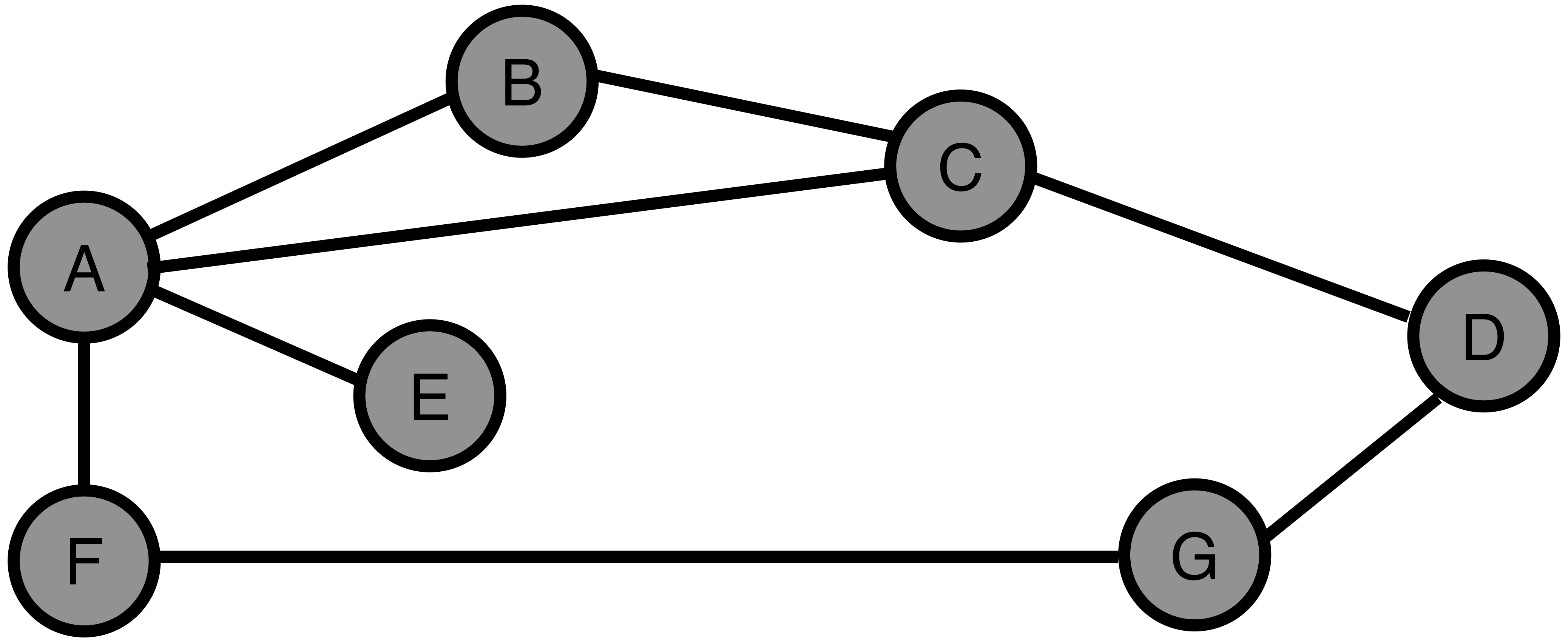
Dest.	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	E

=

E

Dest.	Cost	NextHop
A	1	A
B	2	A
C	2	A
D	3	A
F	2	A
G	2	A

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

ON A ROUTING PROBLEM*

By RICHARD BELLMAN (*The RAND Corporation*)

Summary. Given a set of N cities, with every two linked by a road, and the times required to traverse these roads, we wish to determine the path from one given city to another given city which minimizes the travel time. The times are not directly proportional to the distances due to varying quality of roads and varying quantities of traffic.

The functional equation technique of dynamic programming, combined with approximation in policy space, yields an iterative algorithm which converges after at most $(N - 1)$ iterations.

1. Introduction. The problem we wish to treat is a combinatorial one involving the determination of an optimal route from one point to another. These problems are usually difficult when we allow a continuum, and when we admit only a discrete set of paths, as we shall do below, they are notoriously so.

The purpose of this paper is to show that the functional equation technique of dynamic programming, [1, 2], combined with the concept of approximation in policy space, yields a method of successive approximations which is readily accessible to either hand or machine computation for problems of realistic magnitude. The method is distinguished by the fact that it is a method of exhaustion, i.e. it converges after a finite number of iterations, bounded in advance.

2. Formulation. Consider a set of N cities, numbered in some arbitrary fashion from 1 to N, with every two linked by a direct road. The time required to travel from i to j is not directly proportional to the distance between i and j, due to road conditions and traffic. Given the matrix $T = (t_{ij})$, not necessarily symmetric, where t_{ij} is the time required to travel from i to j, we wish to trace a path between 1 and N which consumes minimum time.

Since there are only a finite number of paths available, the problem reduces to choosing the smallest from a finite set of numbers. This direct, or enumerative, approach is impossible to execute, however, for values of N of the order of magnitude of 20.

We shall construct a search technique which greatly reduces the time required to find minimal paths.

3. Functional equation approach. Let us now introduce a dynamic programming approach. Let

$$f_i = \text{the time required to travel from } i \text{ to } N,\ i = 1, 2, \cdots, N-1, \text{ using an optimal policy,} \quad (3.1)$$

with $f_N = 0$.

Employing the principle of optimality, we see that the f_i satisfy the nonlinear system of equations

$$f_i = \underset{j \neq i}{\text{Min}} [t_{ij} + f_j], \quad i = 1, 2, \cdots, N-1, \qquad f_N = 0. \quad (3.2)$$

*Received January 30, 1957.

- **Worst-case: O(|V|.|E|)**
- **Best-case: O(l.|E|), where l is the maximum length of a shortest path**

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

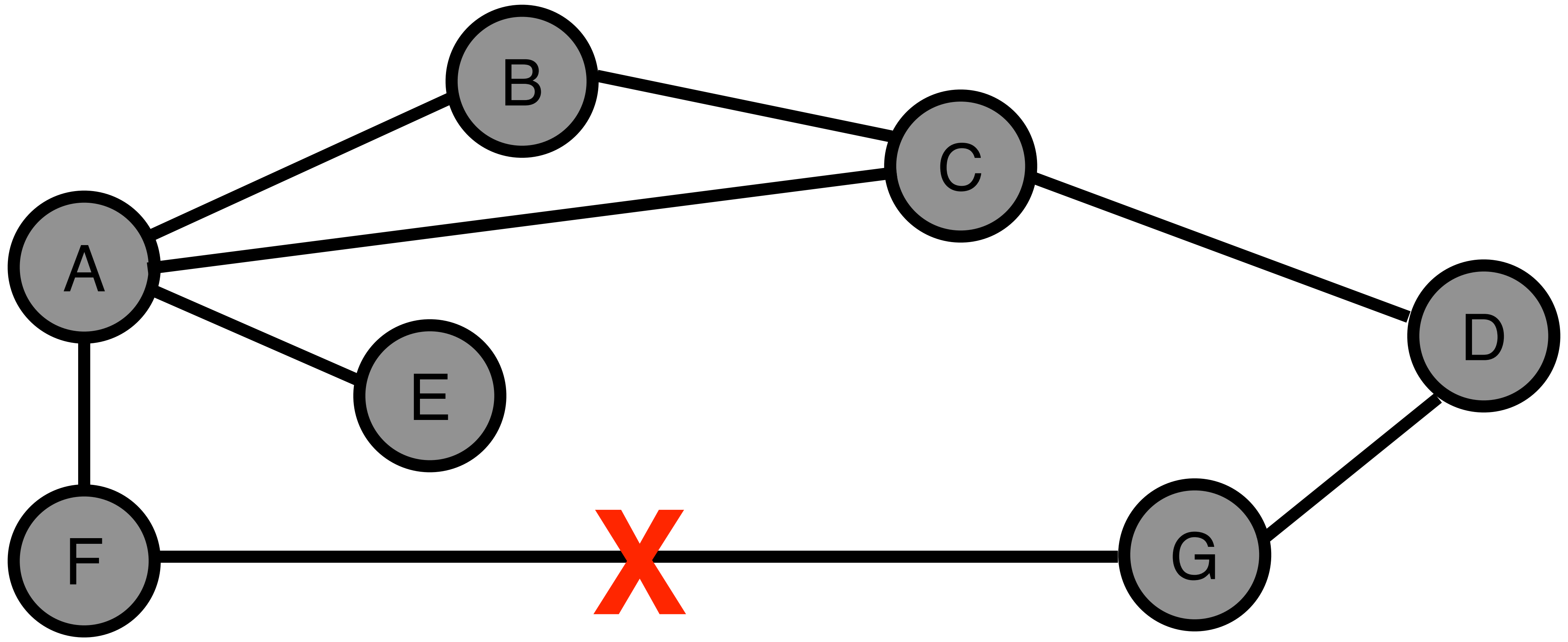
Advantage

- Fast response to the good news

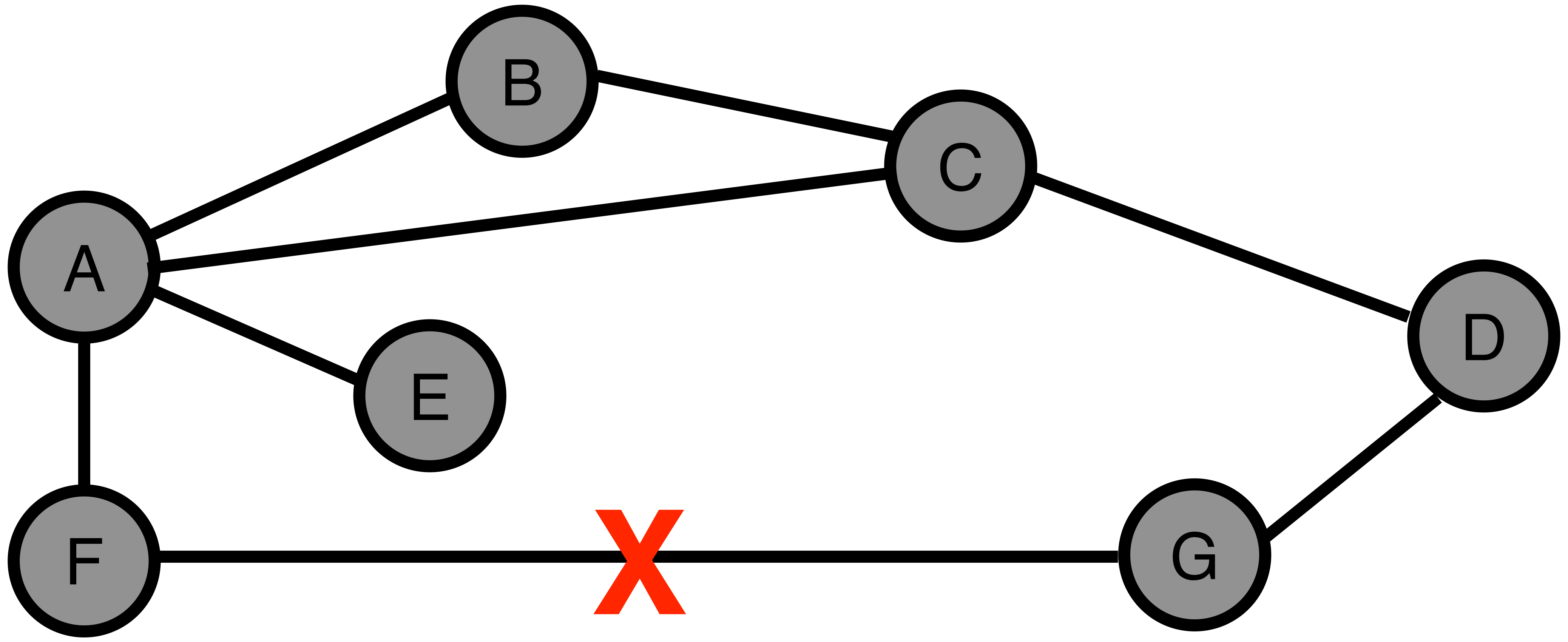
Disadvantage

- Slow response to the bad news

Distance Vector Under Link Failures

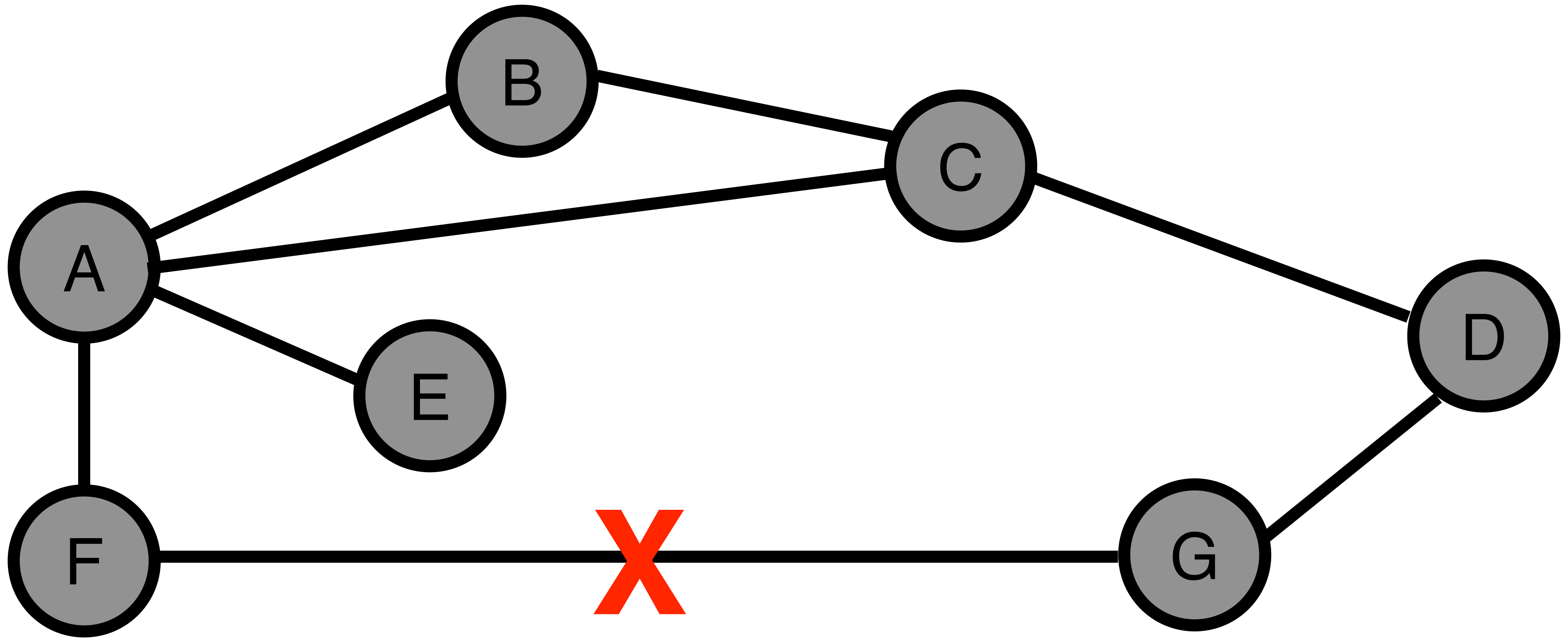


Distance Vector Under Link Failures



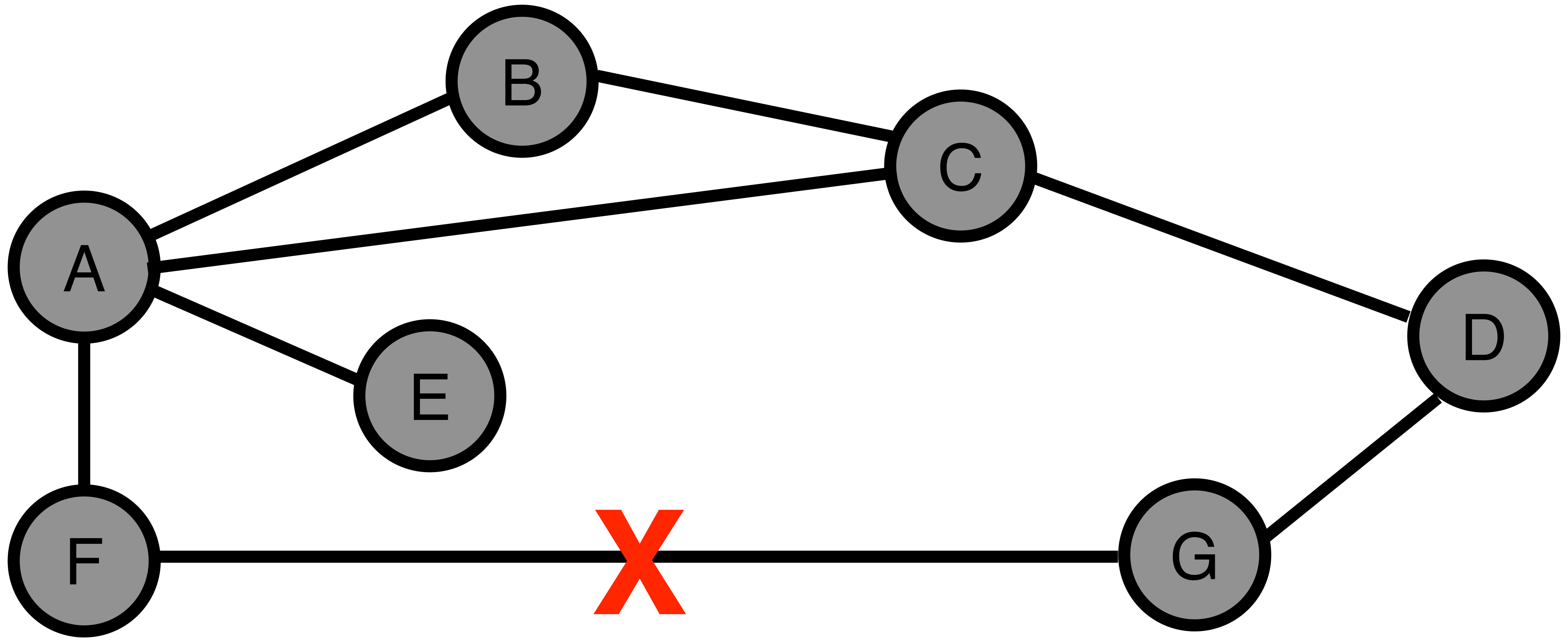
- F detects that the link to G has failed

Distance Vector Under Link Failures



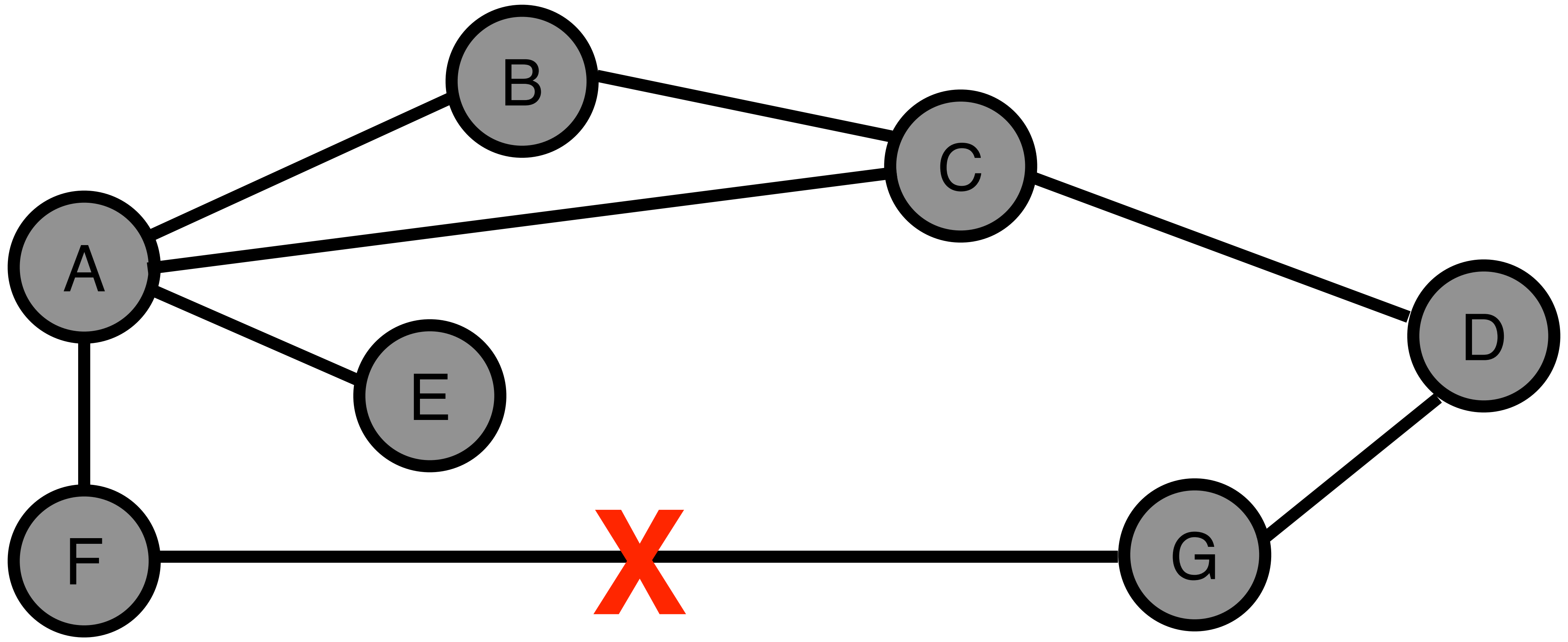
- F detects that the link to G has failed
- F sets the distance to G as infinity and sends updates to A

Distance Vector Under Link Failures



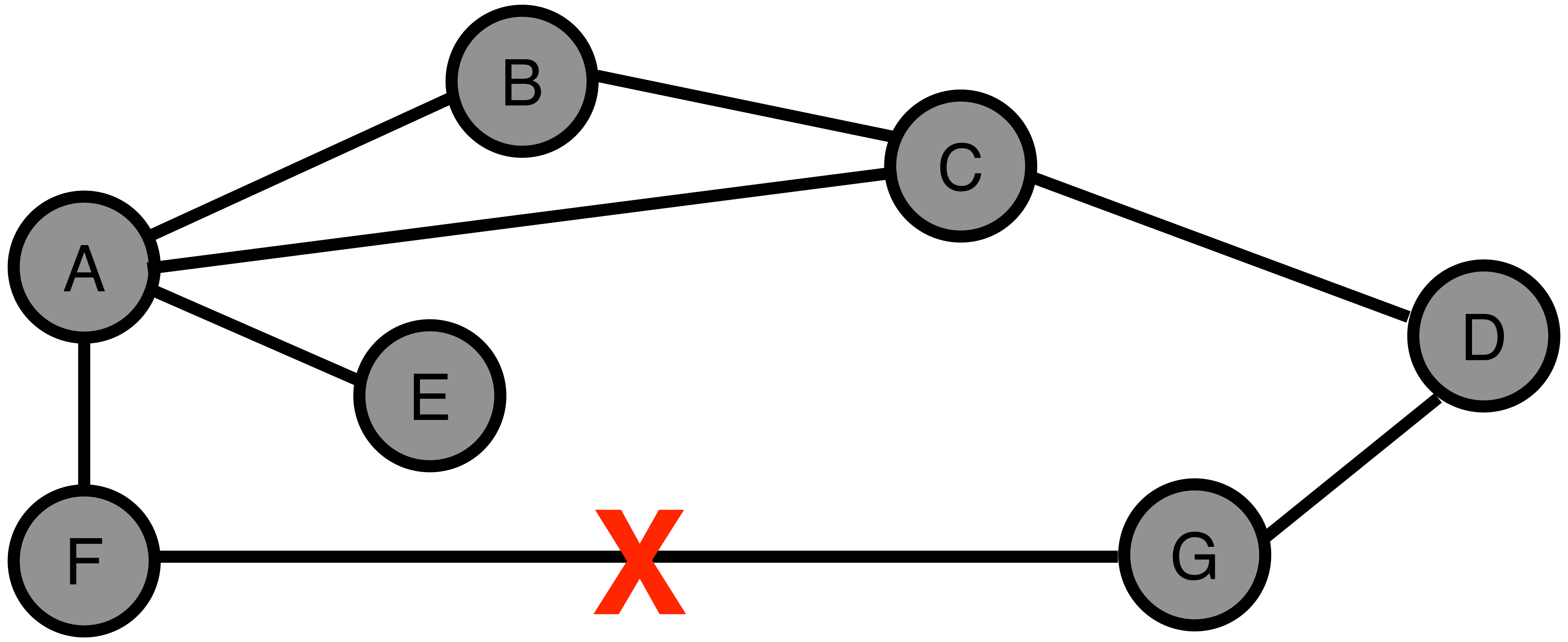
- F detects that the link to G has failed
- F sets the distance to G as infinity and sends updates to A
- A sets the distance to G to infinity since it uses F to reach G

Distance Vector Under Link Failures



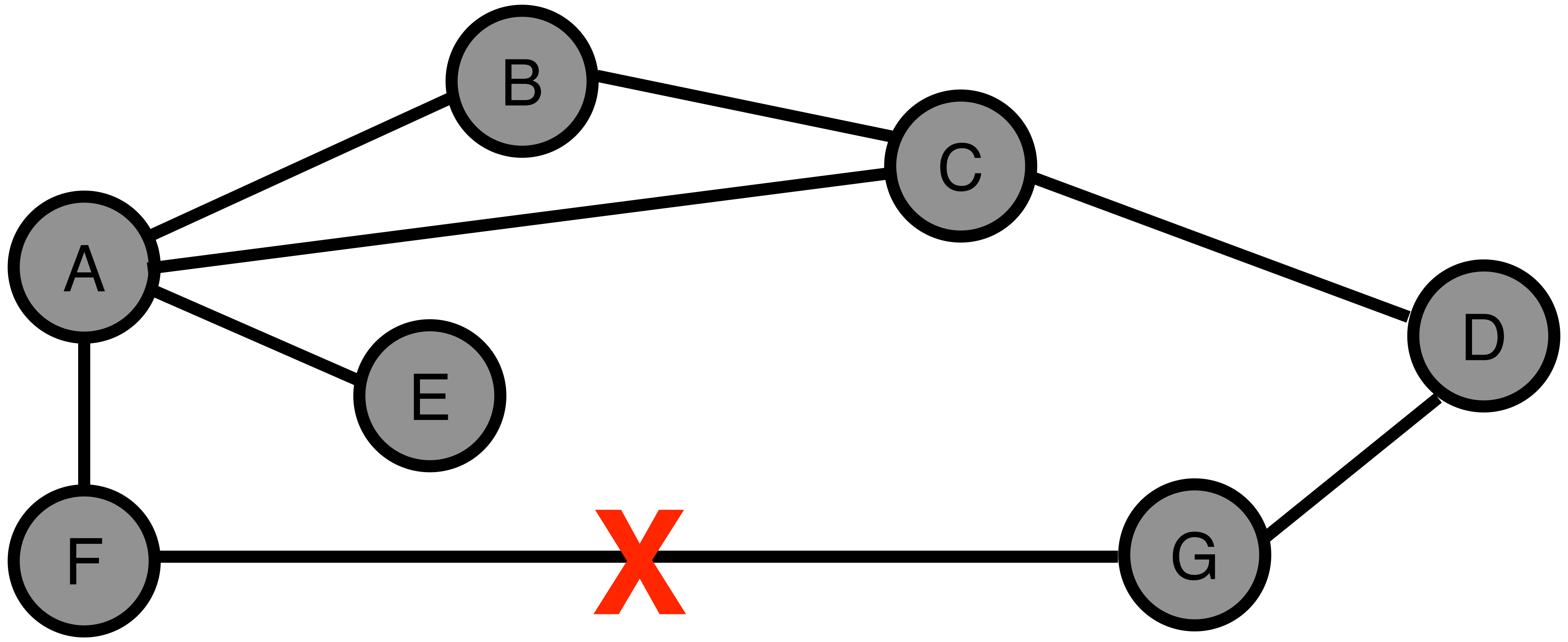
- F detects that the link to G has failed
- F sets the distance to G as infinity and sends updates to A
- A sets the distance to G to infinity since it uses F to reach G
- **A receives a periodic update from C with a 2-hop path to G**

Distance Vector Under Link Failures



- F detects that the link to G has failed
- F sets the distance to G as infinity and sends updates to A
- A sets the distance to G to infinity since it uses F to reach G
- **A receives a periodic update from C with a 2-hop path to G**
- **A sets the distance to G to 3 and sends an update to F**

Distance Vector Under Link Failures



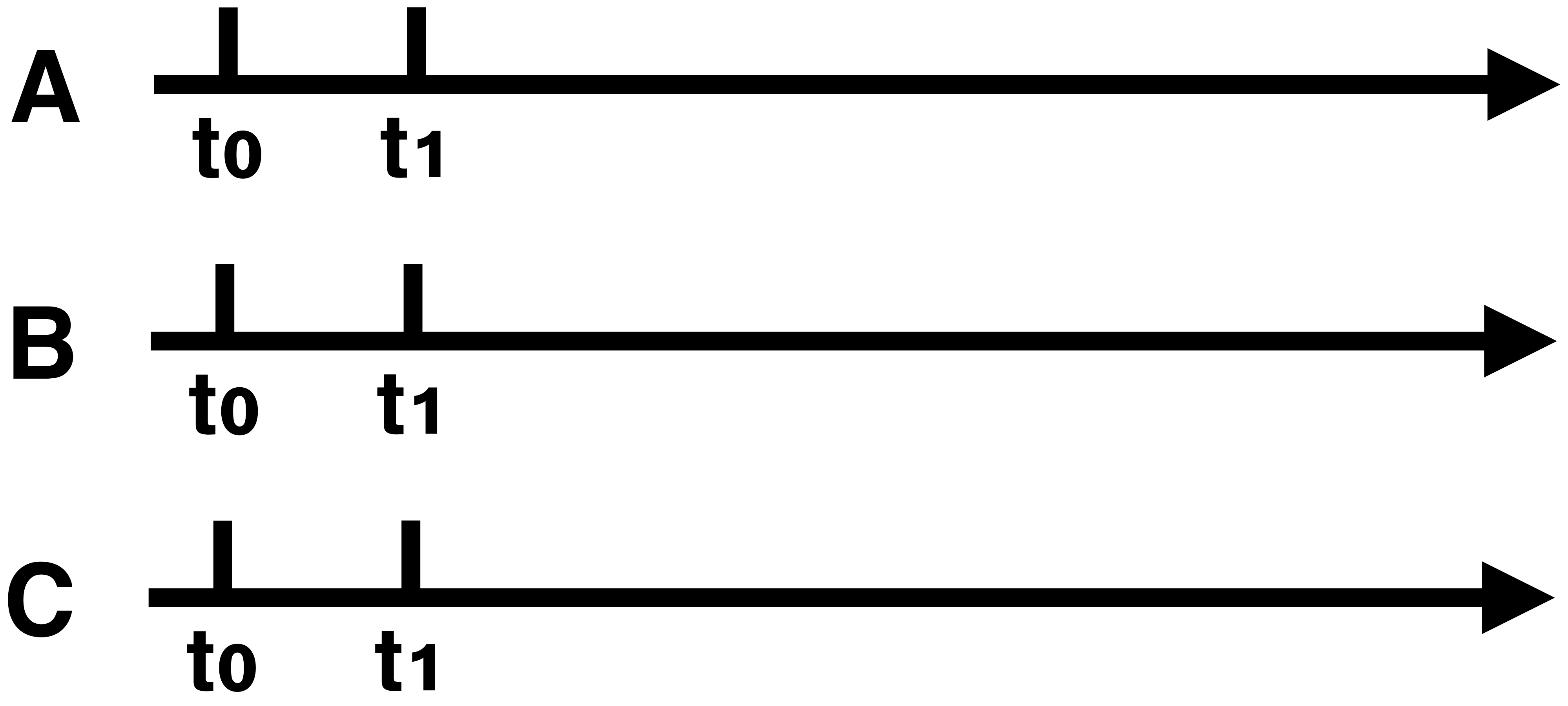
- F detects that the link to G has failed
- F sets the distance to G as infinity and sends updates to A
- A sets the distance to G to infinity since it uses F to reach G
- **A receives a periodic update from C with a 2-hop path to G**
- **A sets the distance to G to 3 and sends an update to F**
- **F decides it can reach G in 4 hops via A**

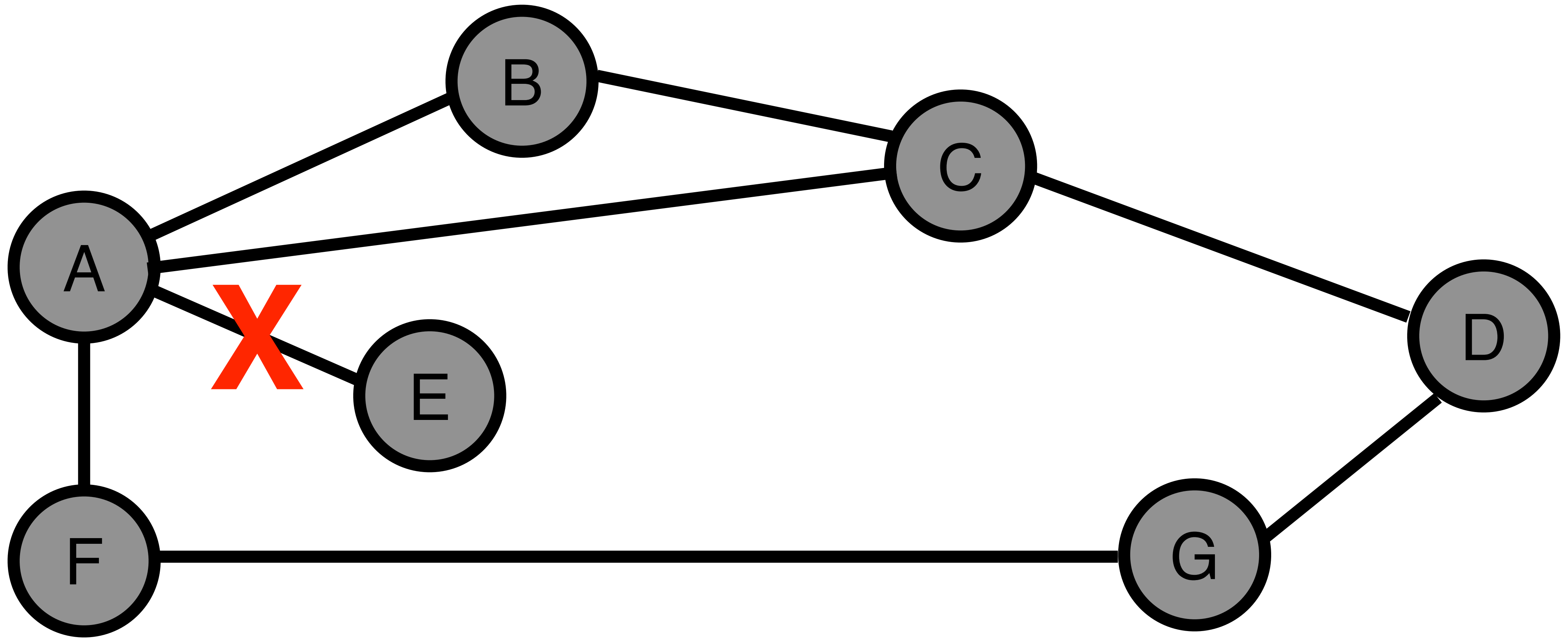
Distance Vector Converges Slowly

- Converge: the process of getting consistent routing information to all the routers

- Slightly different circumstances can prevent the network from stabilizing

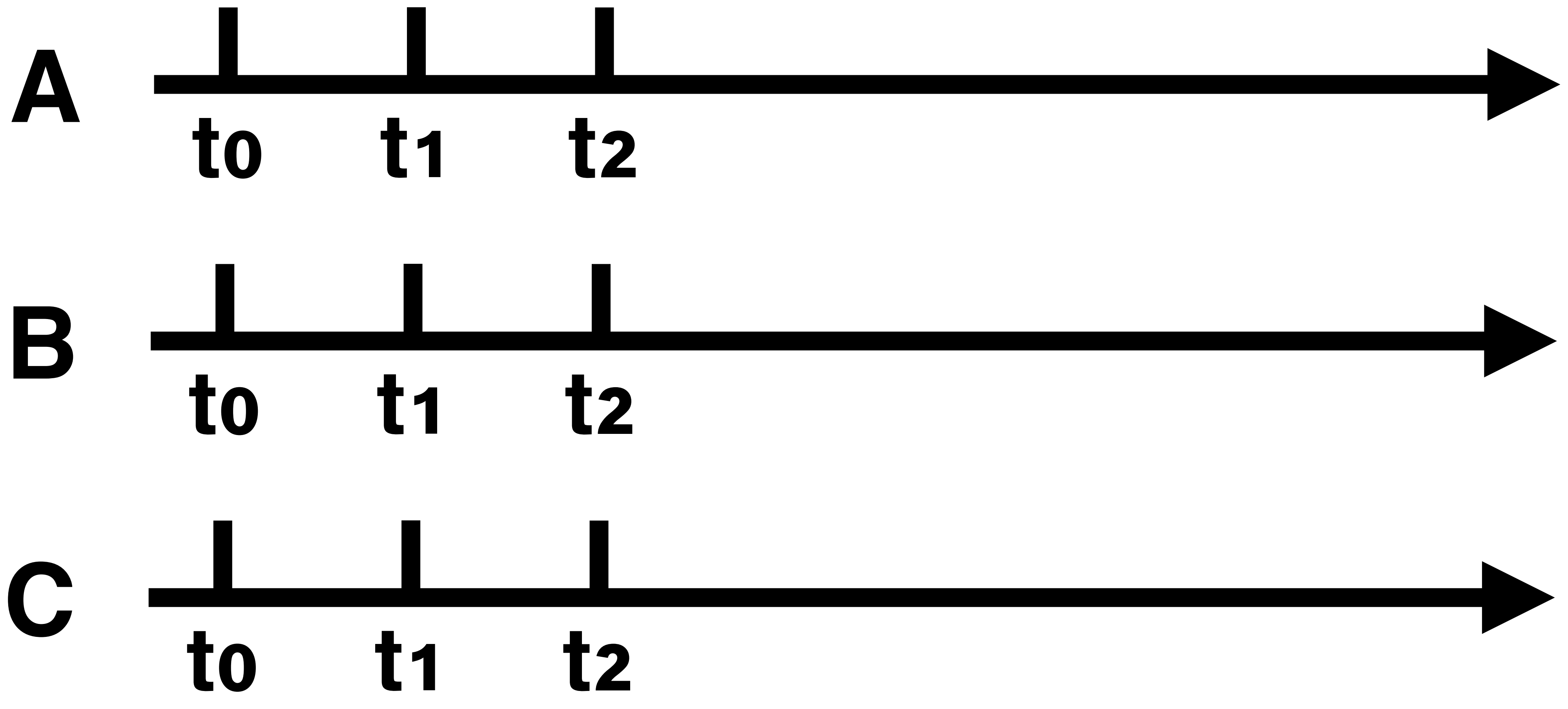
A Slow Converging Example

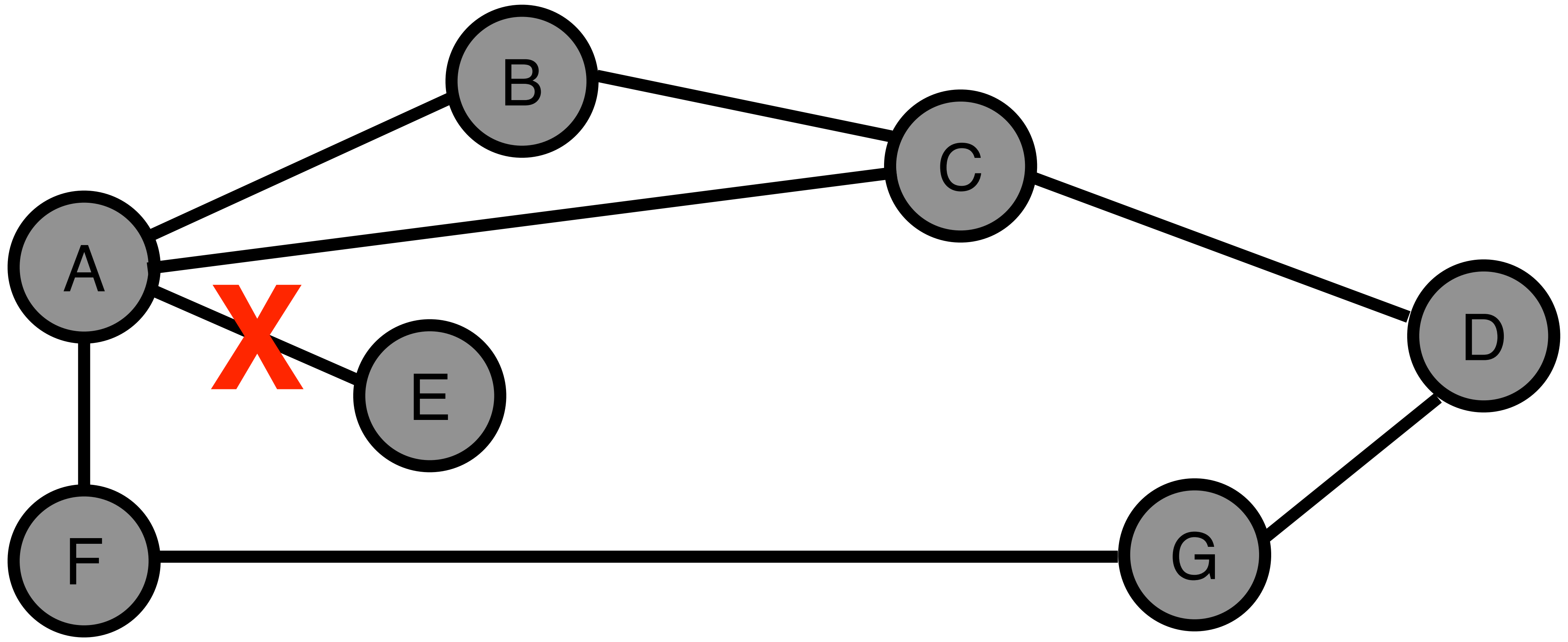




- At **t_0**, A detects the link failure and advertises **a distance of infinity to E**
- At **t_1**, B and C receive the message, and update the routing table accordingly

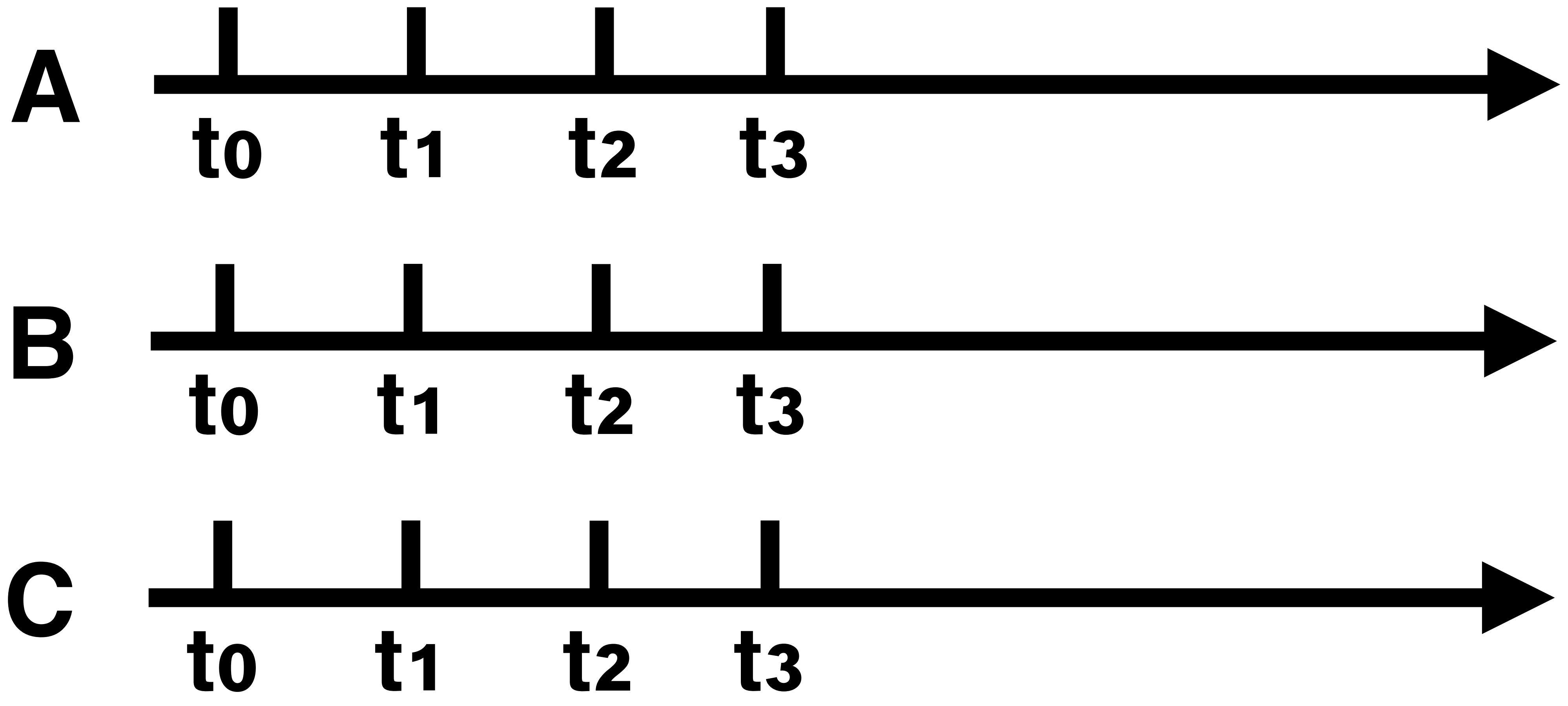
A Slow Converging Example

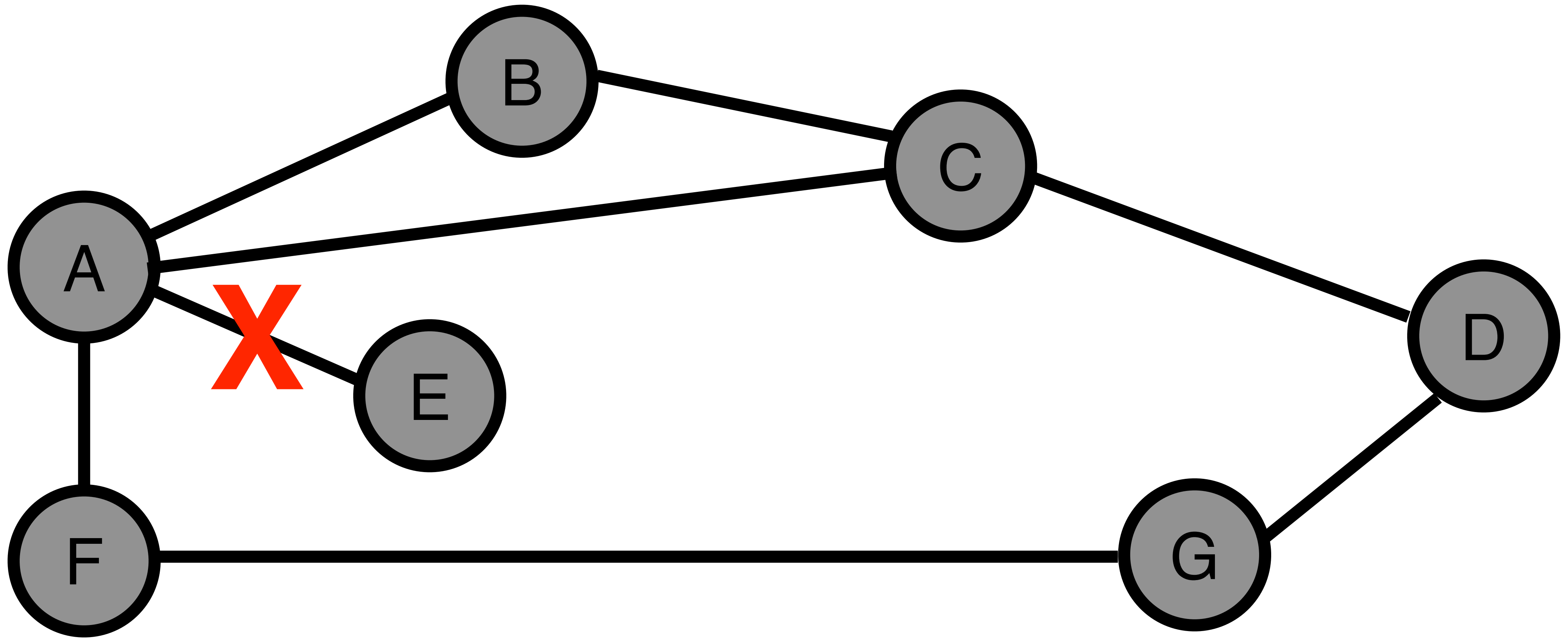




- At **t_0**, A detects the link failure and advertises **a distance of infinity to E**
- At **t_1**, B receives the message from A and updates the routing table as **<E, Infinity>**
- At **t_2**, B receives the message from C (saying the distance to E is 2), and updates the routing table as **<E, 3>**

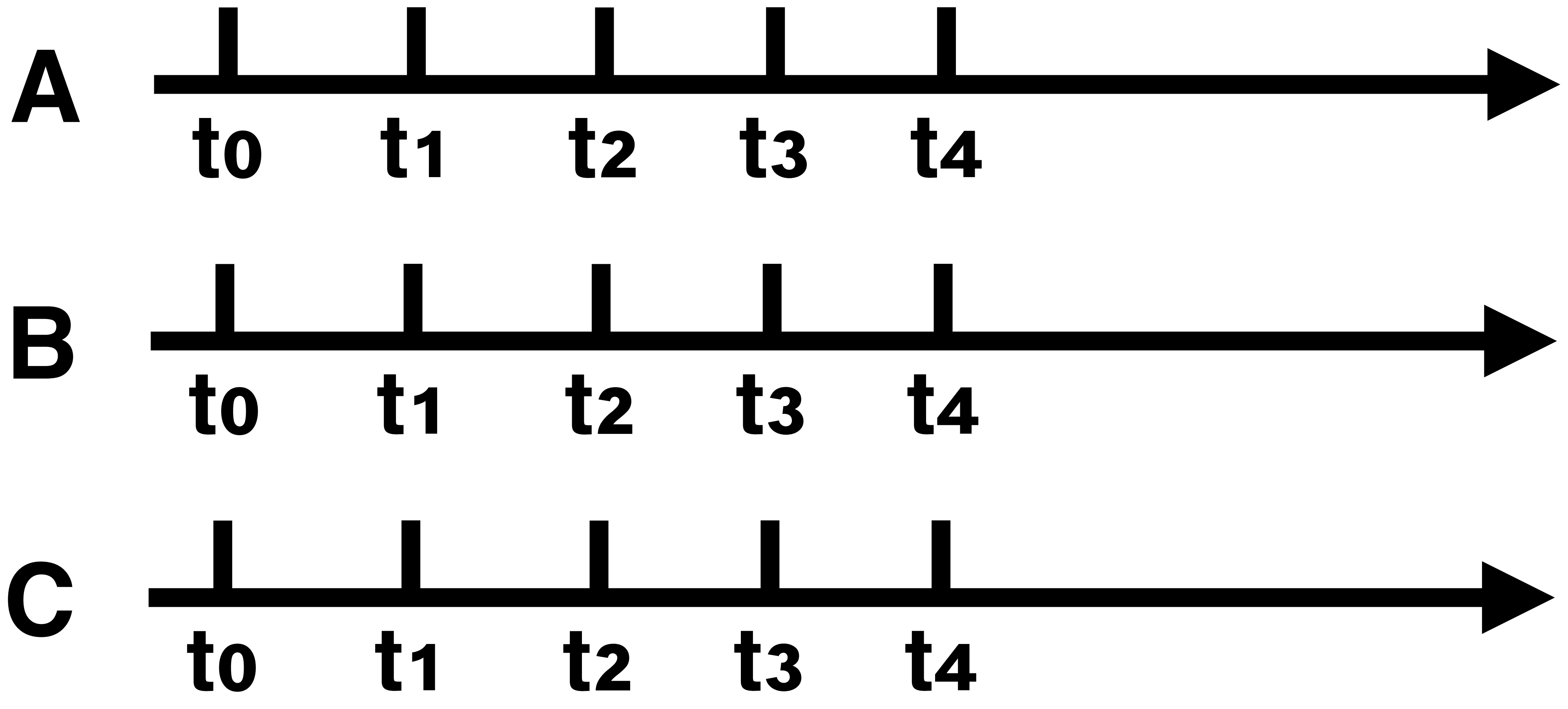
A Slow Converging Example

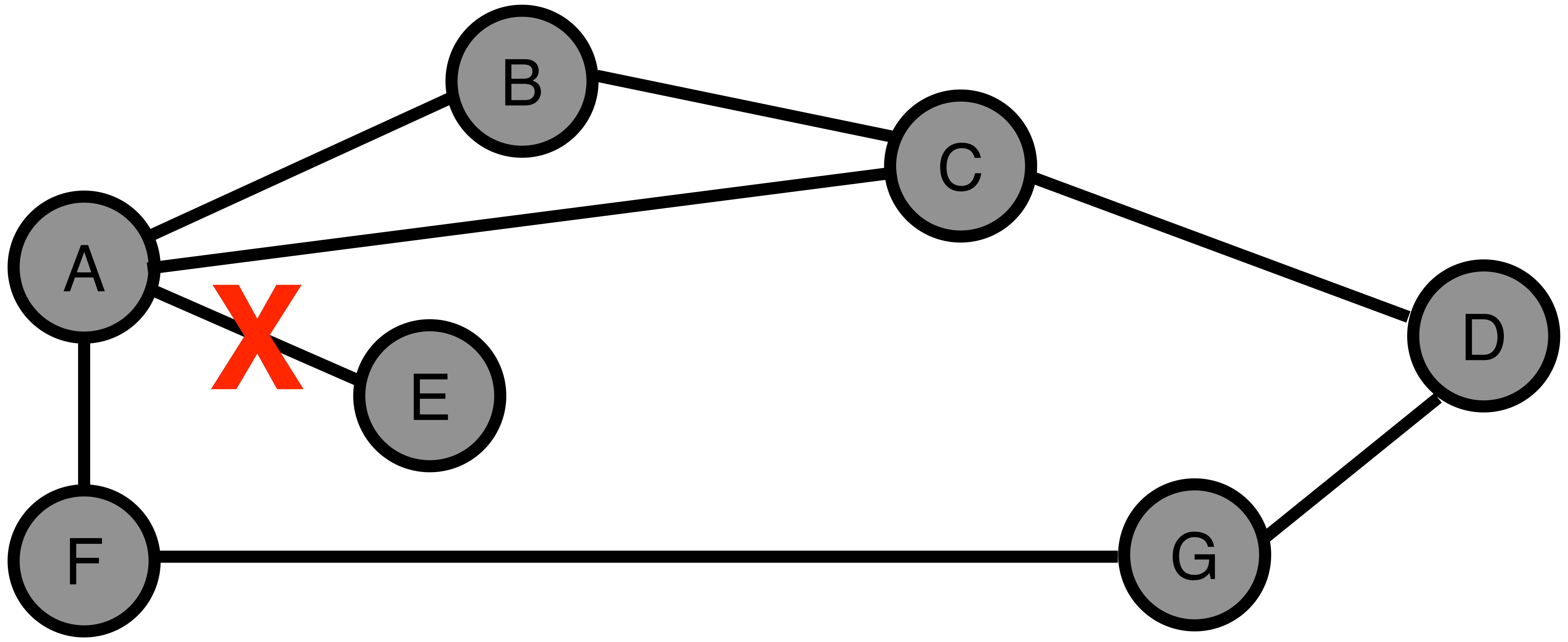




- At t_0, A detects the link failure and advertises **a distance of infinity to E**
- At t_1, B receives the message from A and updates the routing table as **<E, Infinity>**
- At t_2, B receives the message from C (saying the distance to E is 2), and updates the routing table as **<E, 3>**
- At t_3, C receives the message from A and updates the routing table as **<E, Infinity>**

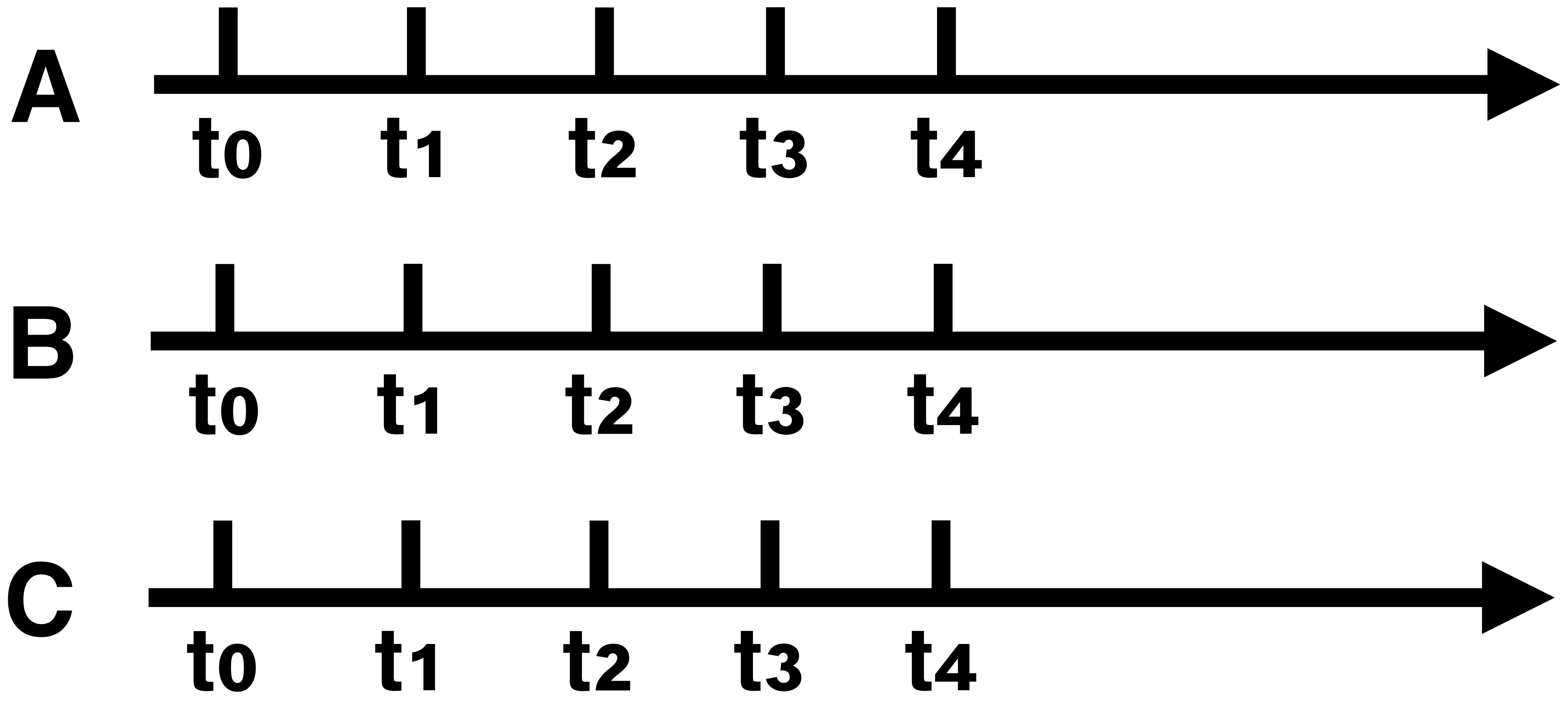
A Slow Converging Example

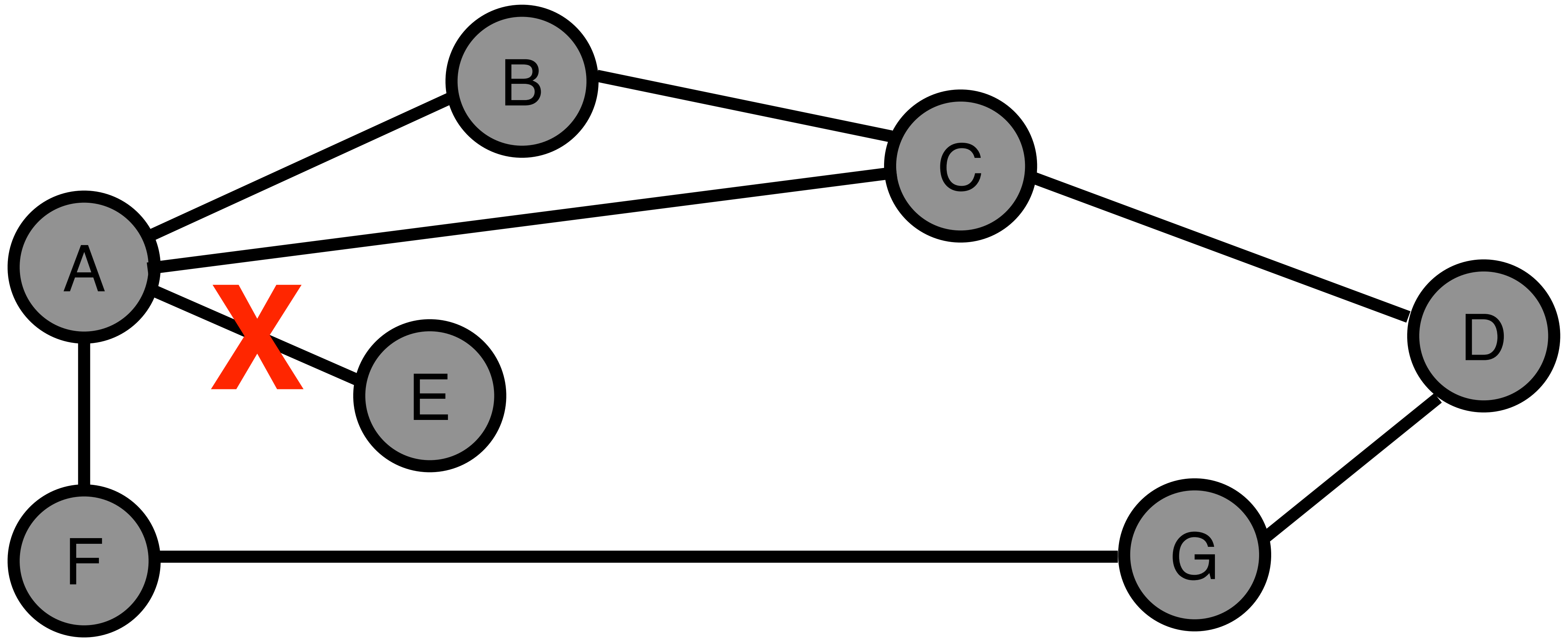




- At t4, C receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**
- At t4, A receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**

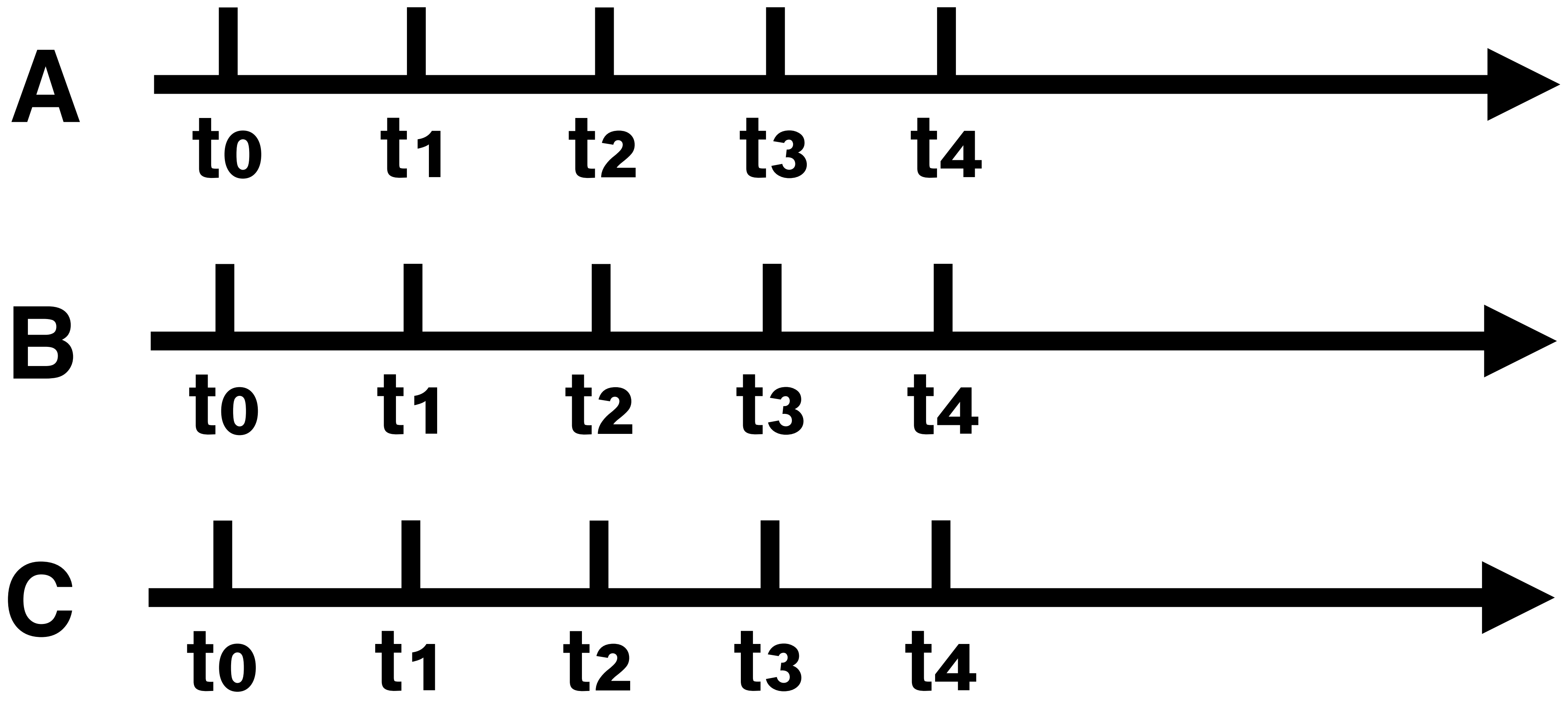
A Slow Converging Example

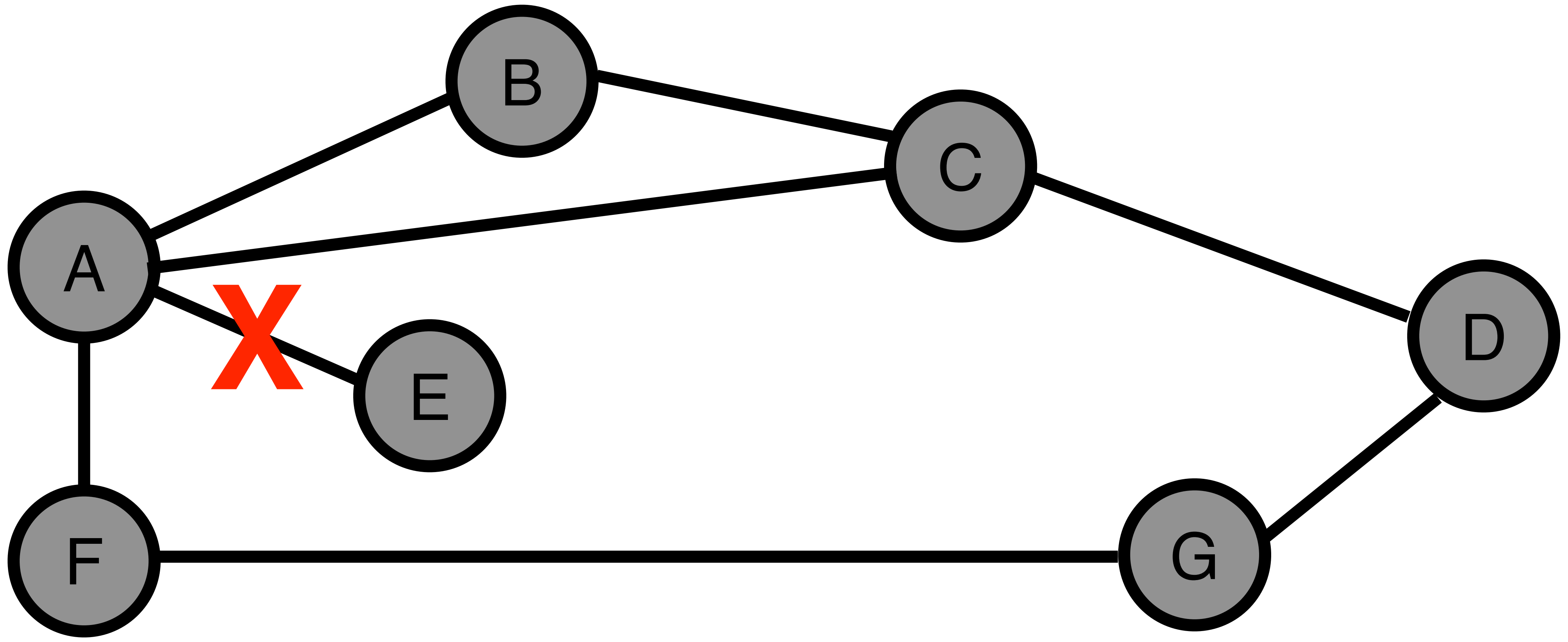




- At **t_4**, C receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**
- At **t_4**, A receives the message from B (saying the distance to E is 3), and updates the routing table as **<E, 4>**
- **A will advertise this new changes to C, then C advertises B, B advertises A, …**

A Slow Converging Example



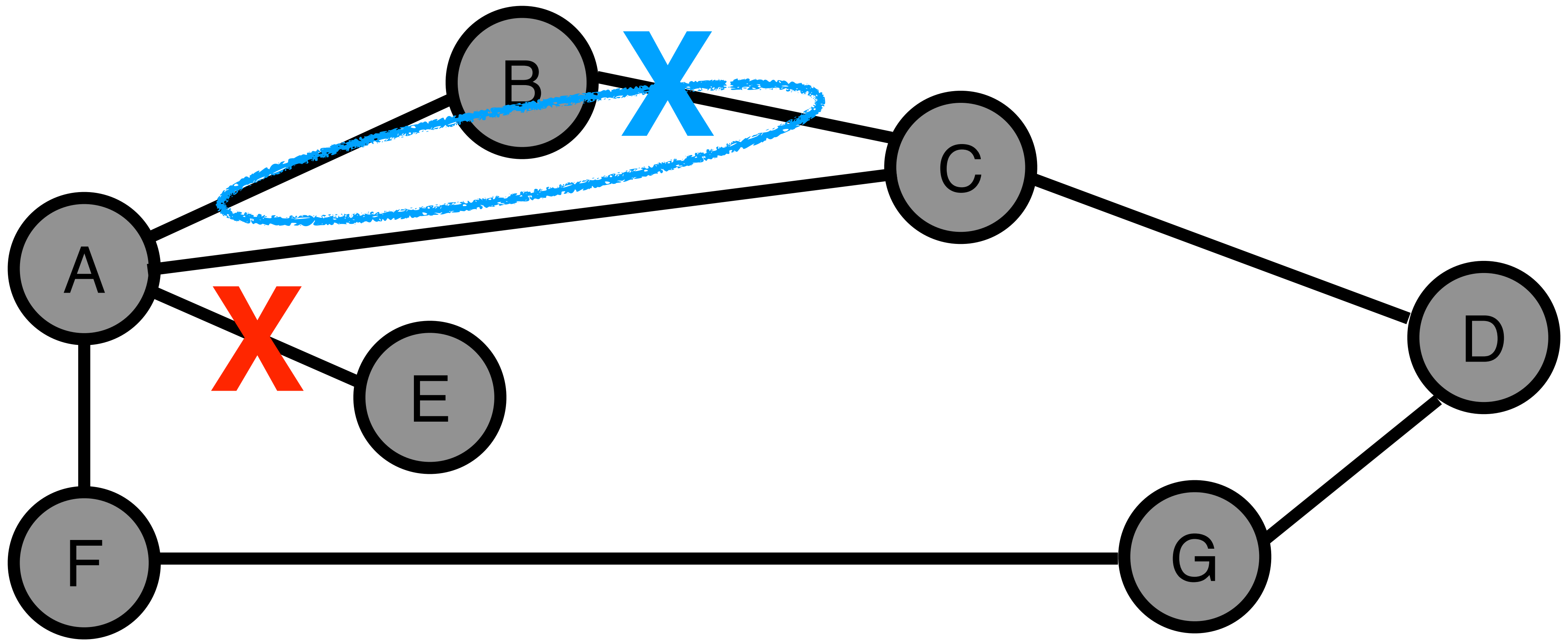


This cycle stops only when the distances reach some threshold that is large enough to be considered infinite

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

Count-to-Infinity Problem: A Simple Fix

- Use a relatively small number as an approximation of infinity
  - The maximum number of hops to traverse a network never exceeds 16

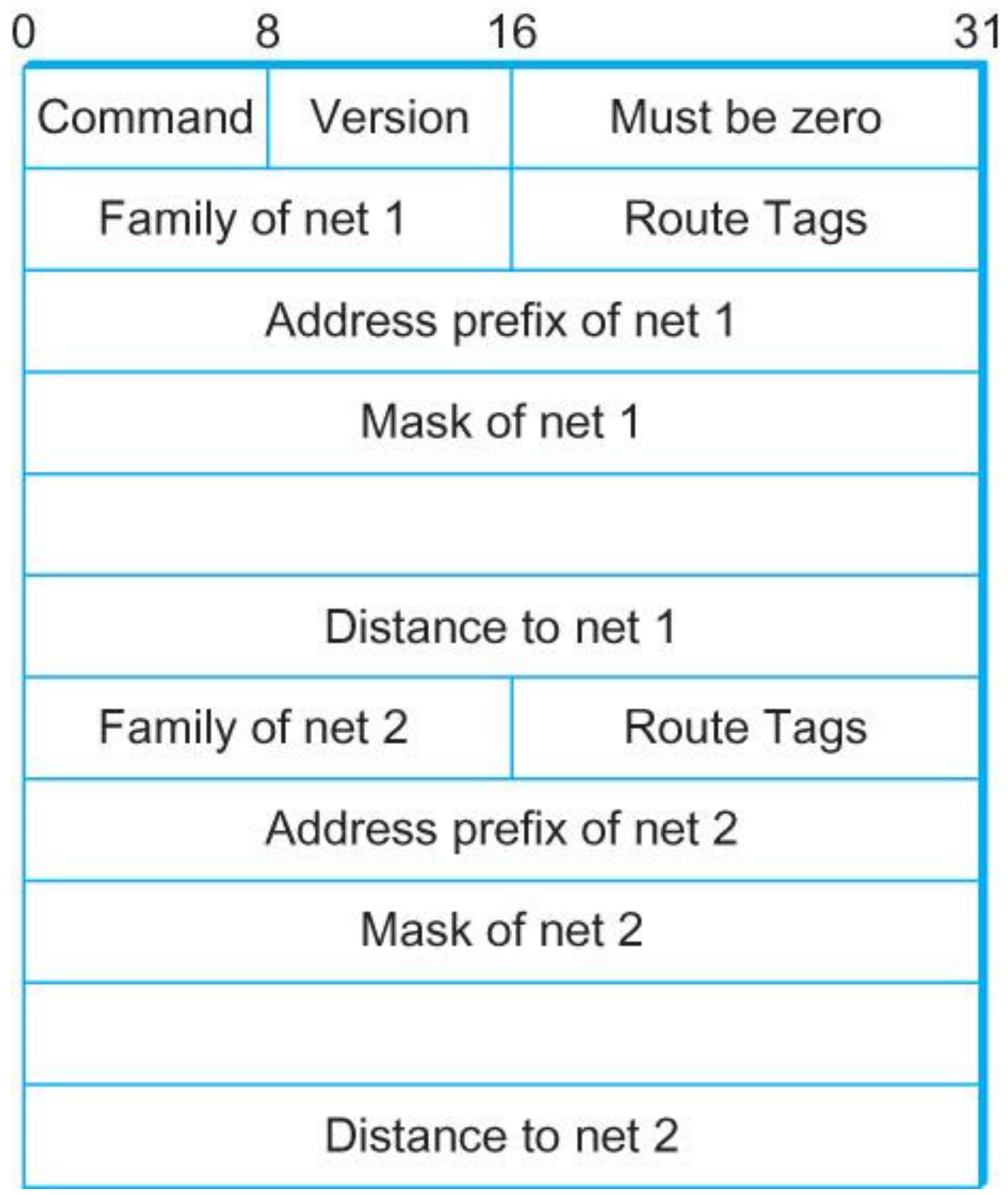


Routing Information Protocol (RIP)

- Earliest IP routing protocol
  - 1982 BSD of Unix
  - The current standard is version 2 (RFC 1723)

- Features
  - Cost: the number of hops
  - "Infinity" = 16

- Sending updates
  - Routers listen for updates on the UDP port 520
  - Frequency: 30 seconds
  - Triggered when an entry is changed



Summary

- Today
  - Distance vector routing

- Next lecture
  - Link state routing