DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 6
Synchronization

Clock Synchronization

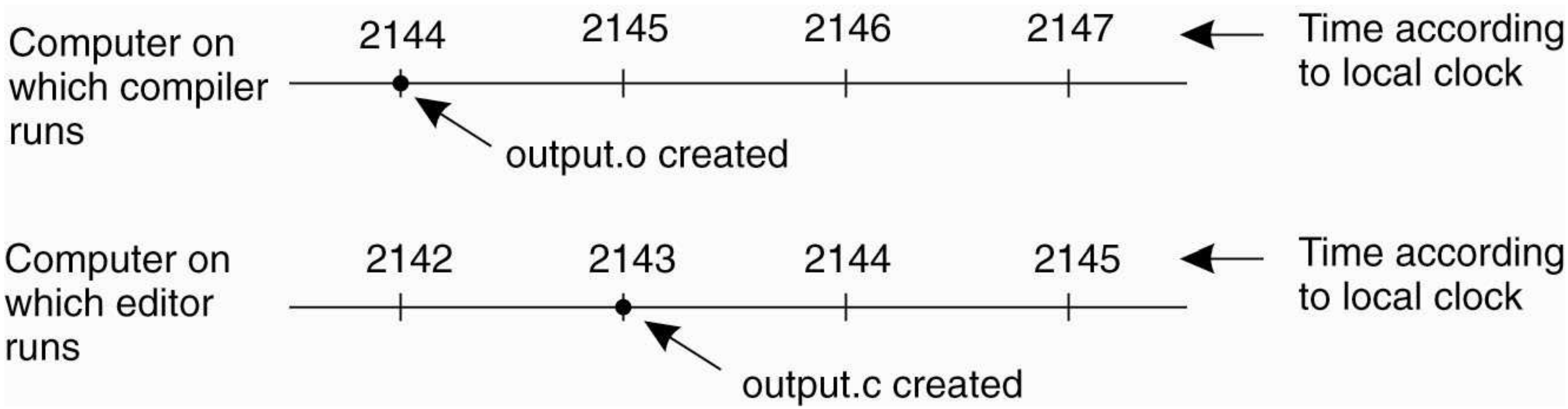


Figure 6-1. When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical Clocks (1)

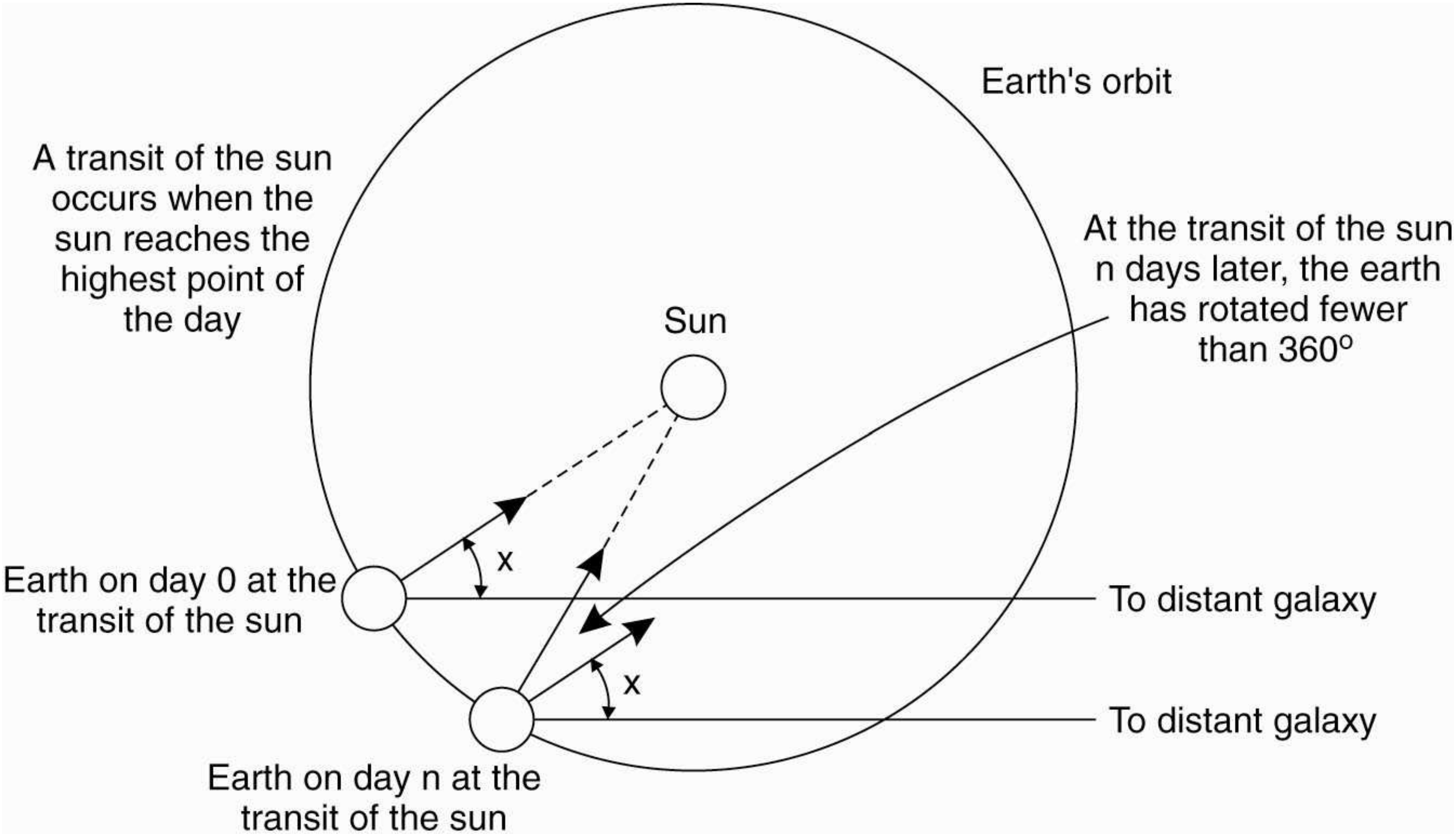


Figure 6-2. Computation of the mean solar day.

Physical Clocks (2)

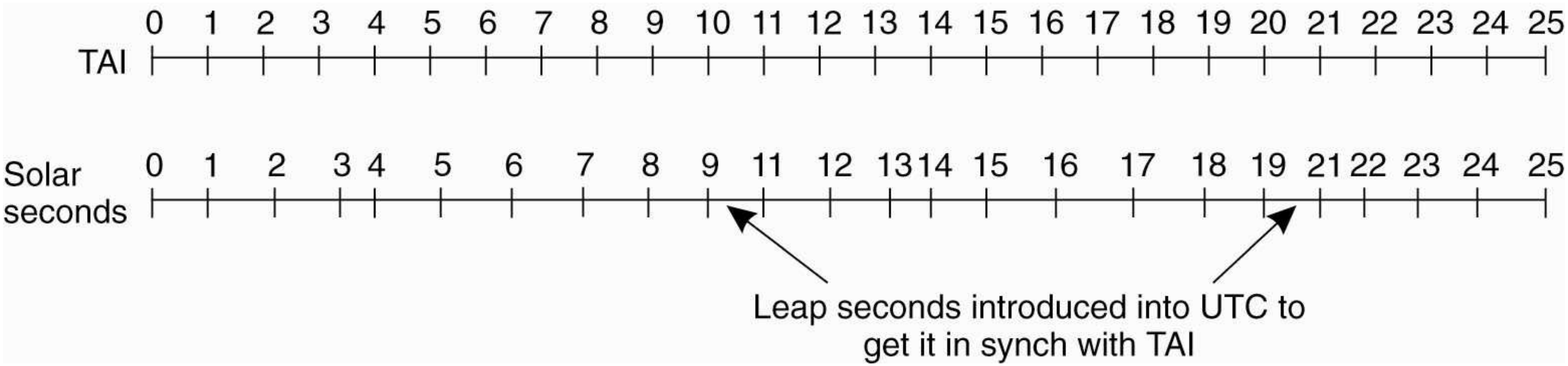


Figure 6-3. TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

Global Positioning System

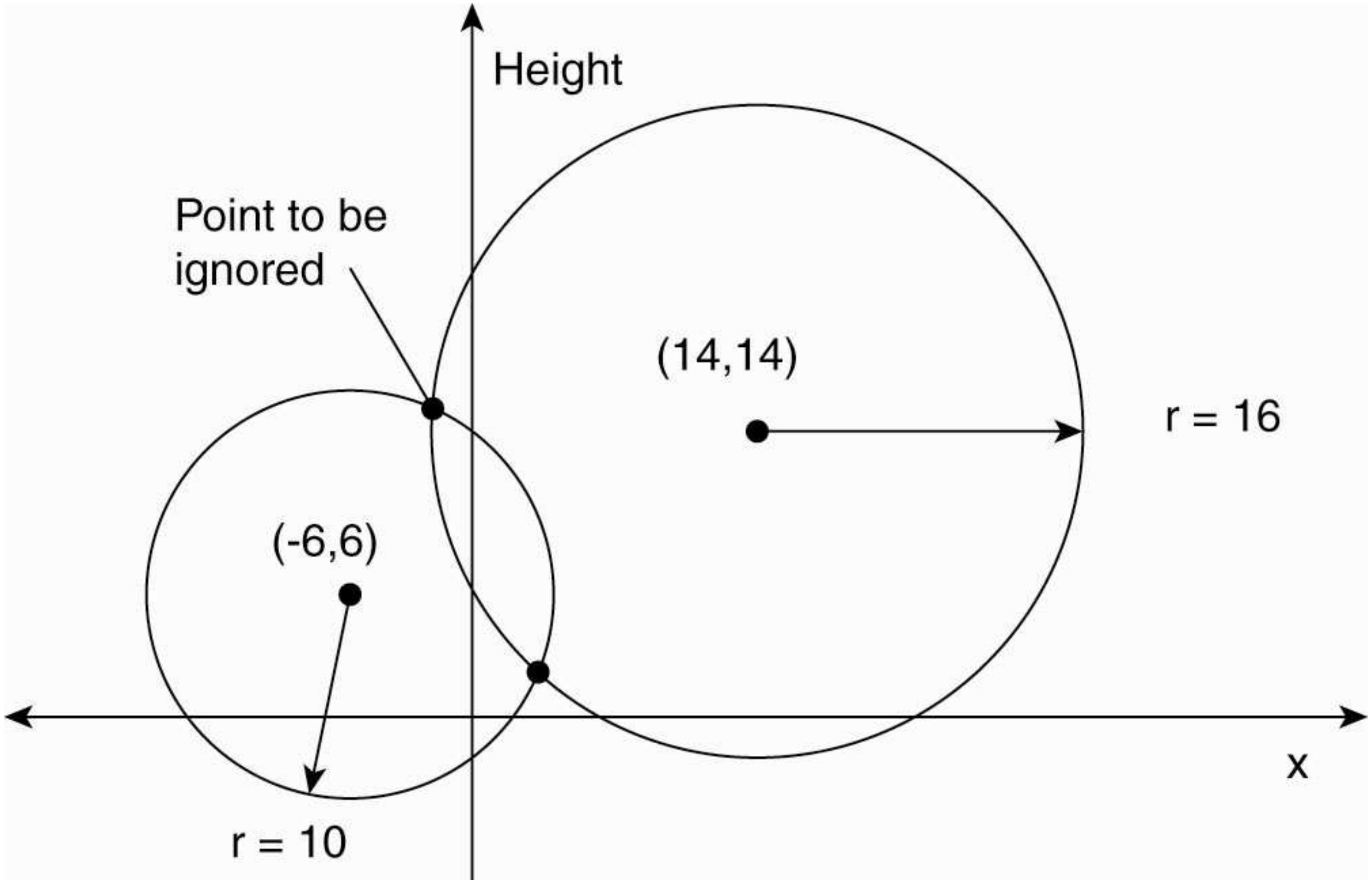


Figure 6-4. Computing a position in a two-dimensional space.

Global Positioning System (cont.)

Real world facts that complicate GPS

1. It takes a while before data on a satellite's position reaches the receiver.
2. The receiver's clock is generally not in synch with that of a satellite.

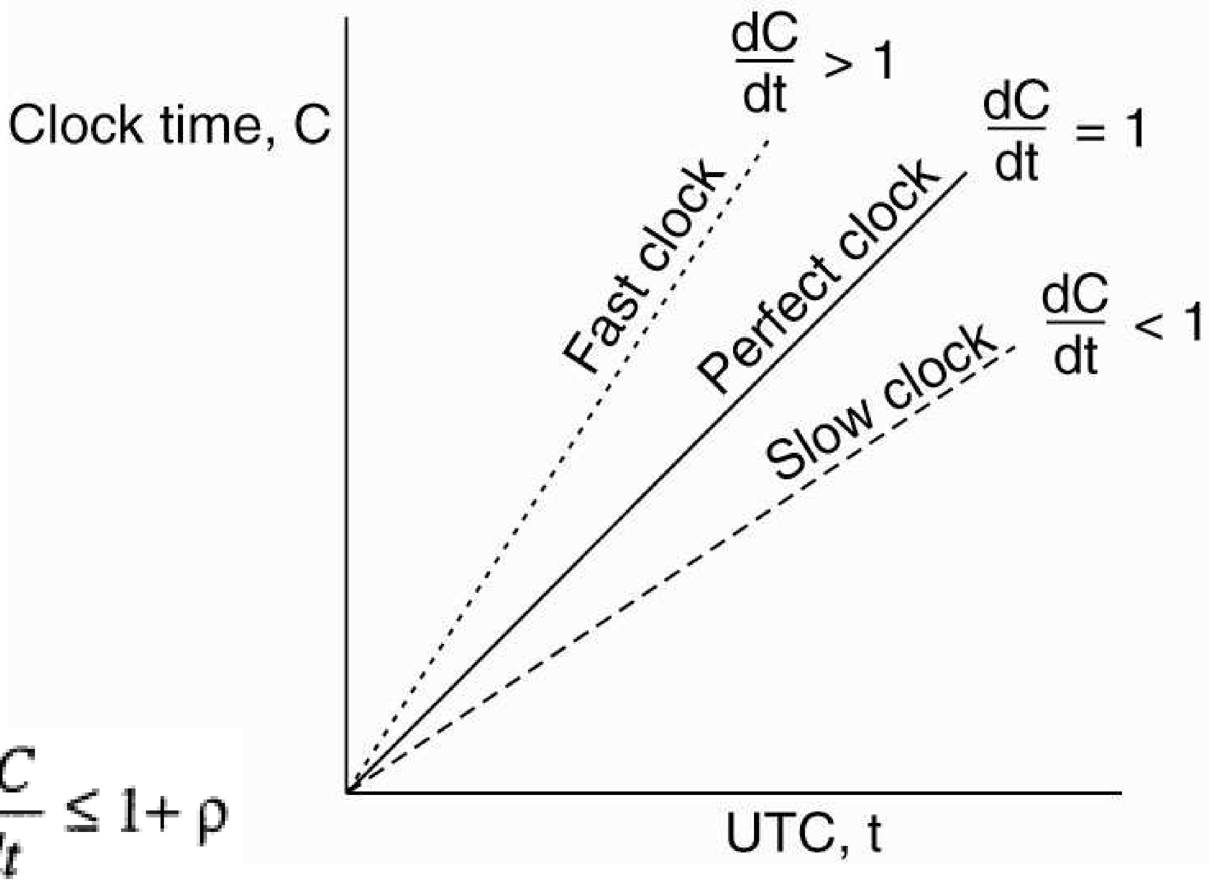
Global Positioning System (cont.)

$$d_i = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

$$\Delta_i = (T_{now} - T_i) + \Delta_r$$

$$d_i = c\,(T_{now} - T_i)$$

Clock Synchronization Algorithms



$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

Figure 6-5. The relation between clock time and UTC when clocks tick at different rates.

Network Time Protocol

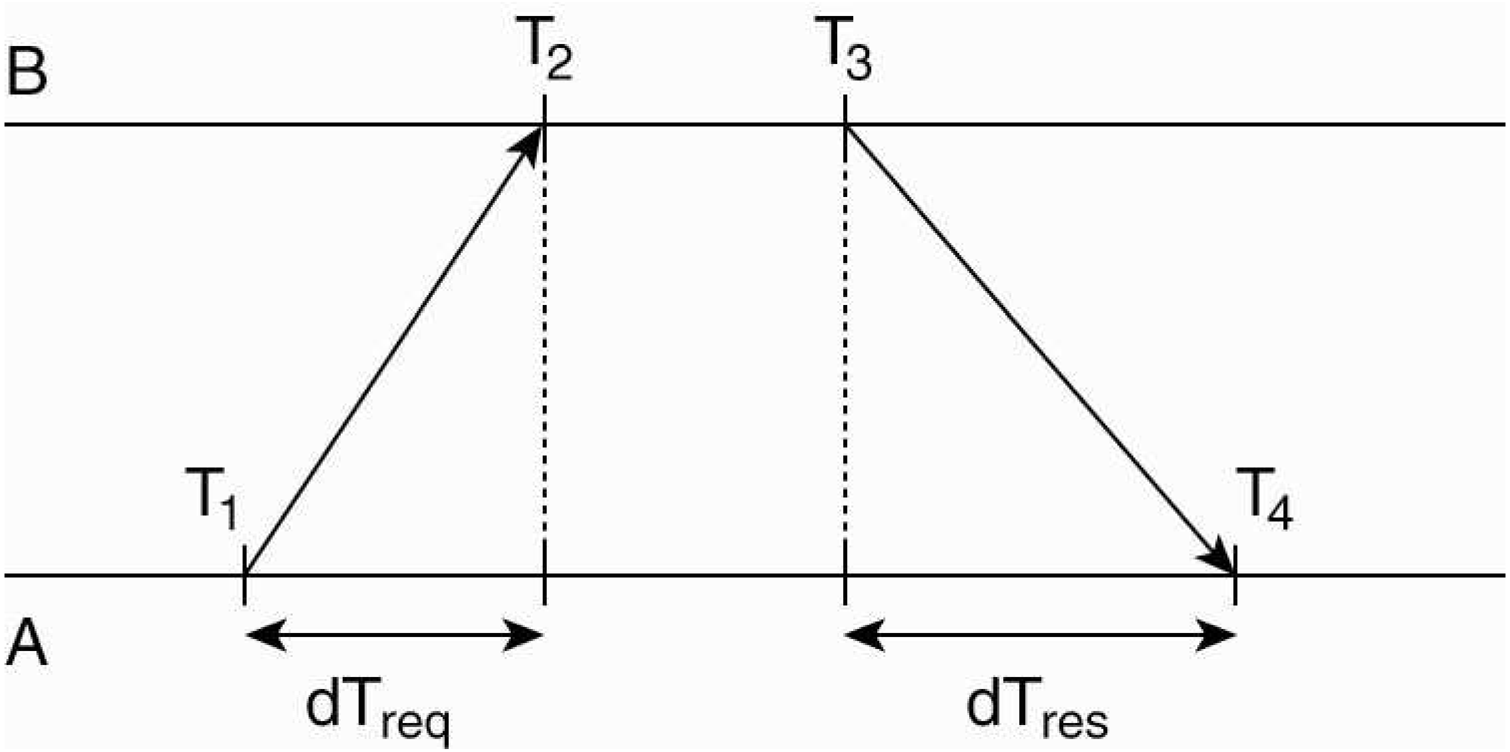
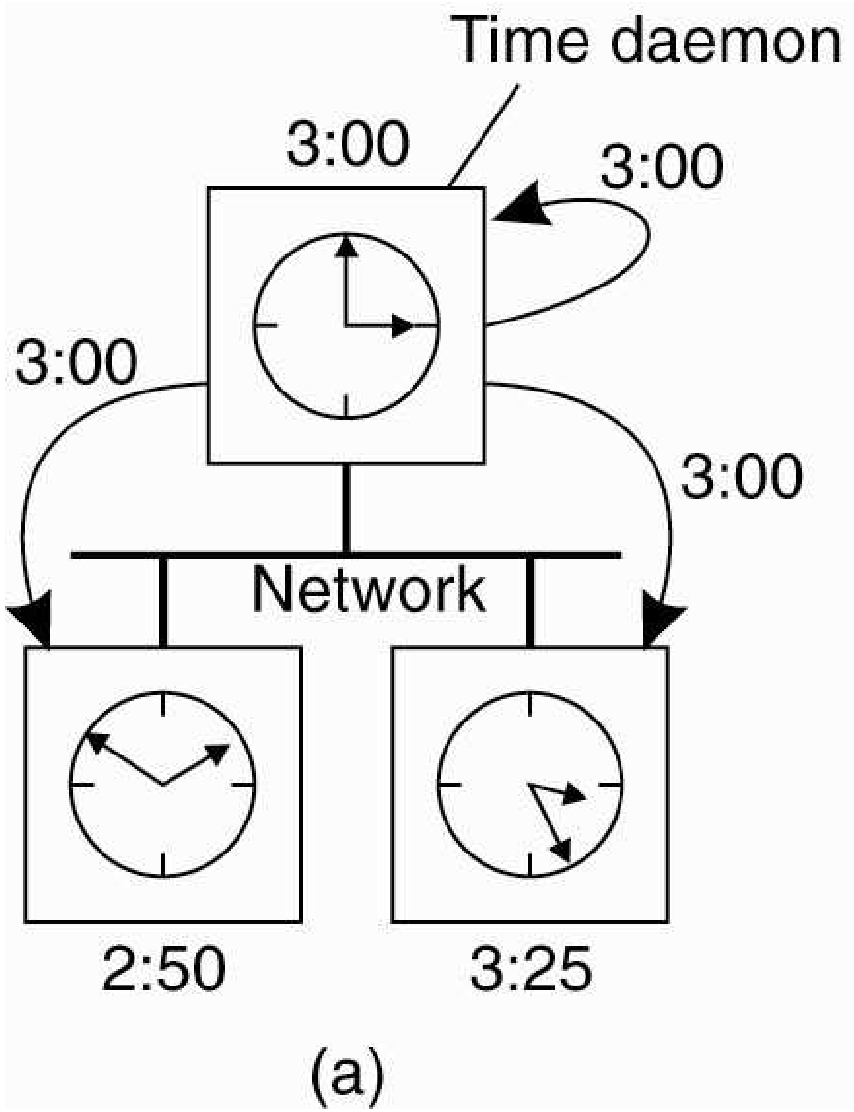


Figure 6-6. Getting the current time from a time server.

The Berkeley Algorithm

Figure 6-7. (a) The time daemon asks all the other machines for their clock values.



The Berkeley Algorithm (cont.)

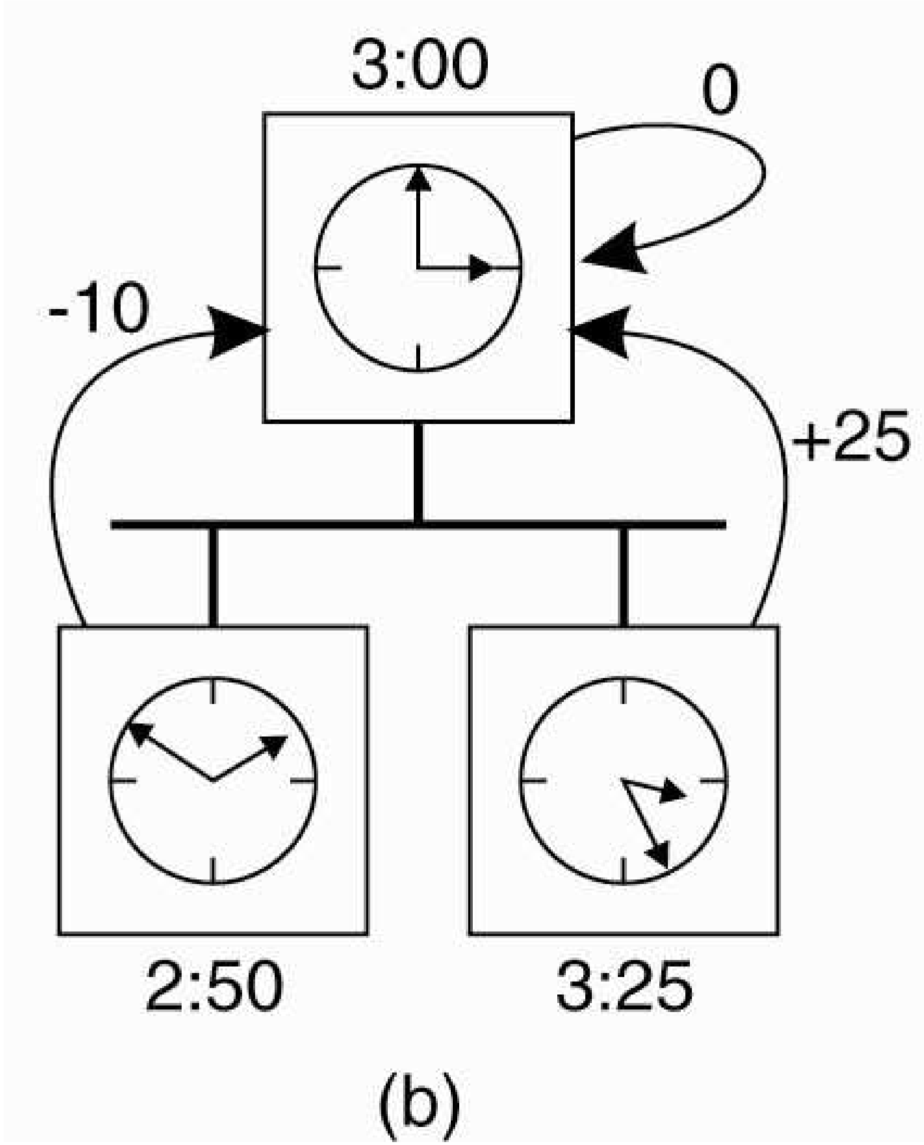
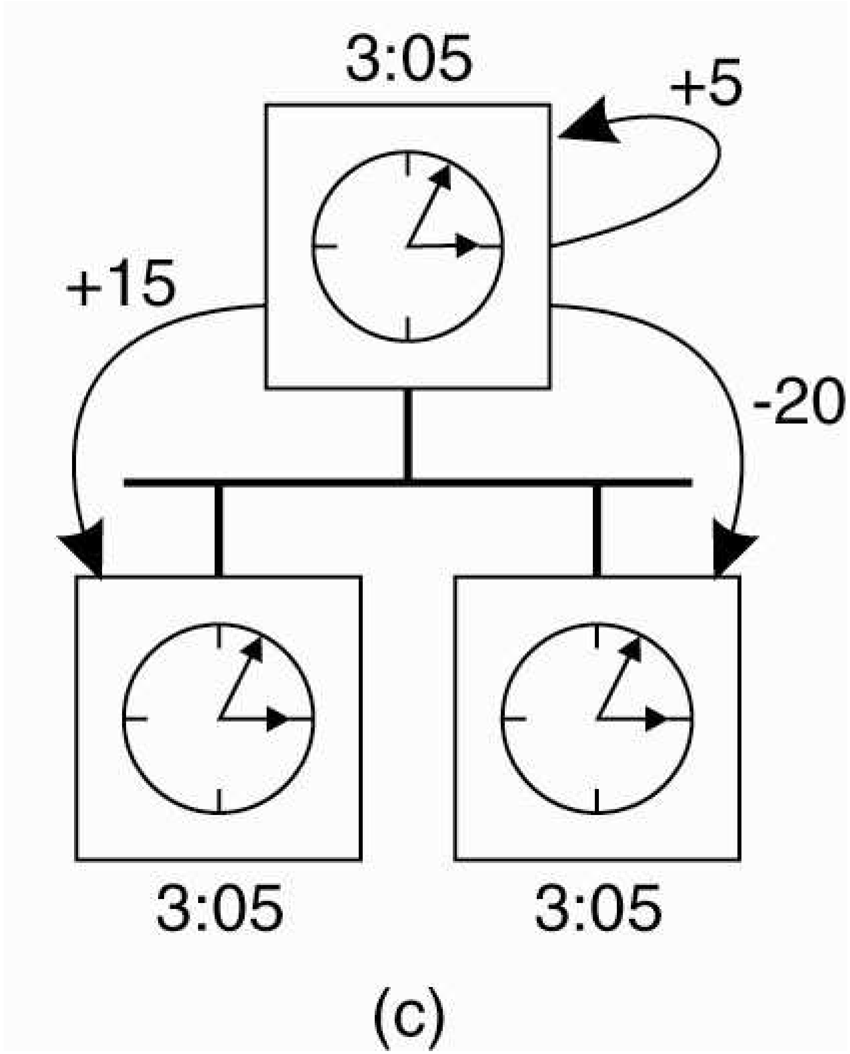


Figure 6-7.

(b) The machines answer.

The Berkeley Algorithm (cont.)

Figure 6-7. (c) The time daemon tells everyone how to adjust their clock.



Clock Synchronization in Wireless Networks

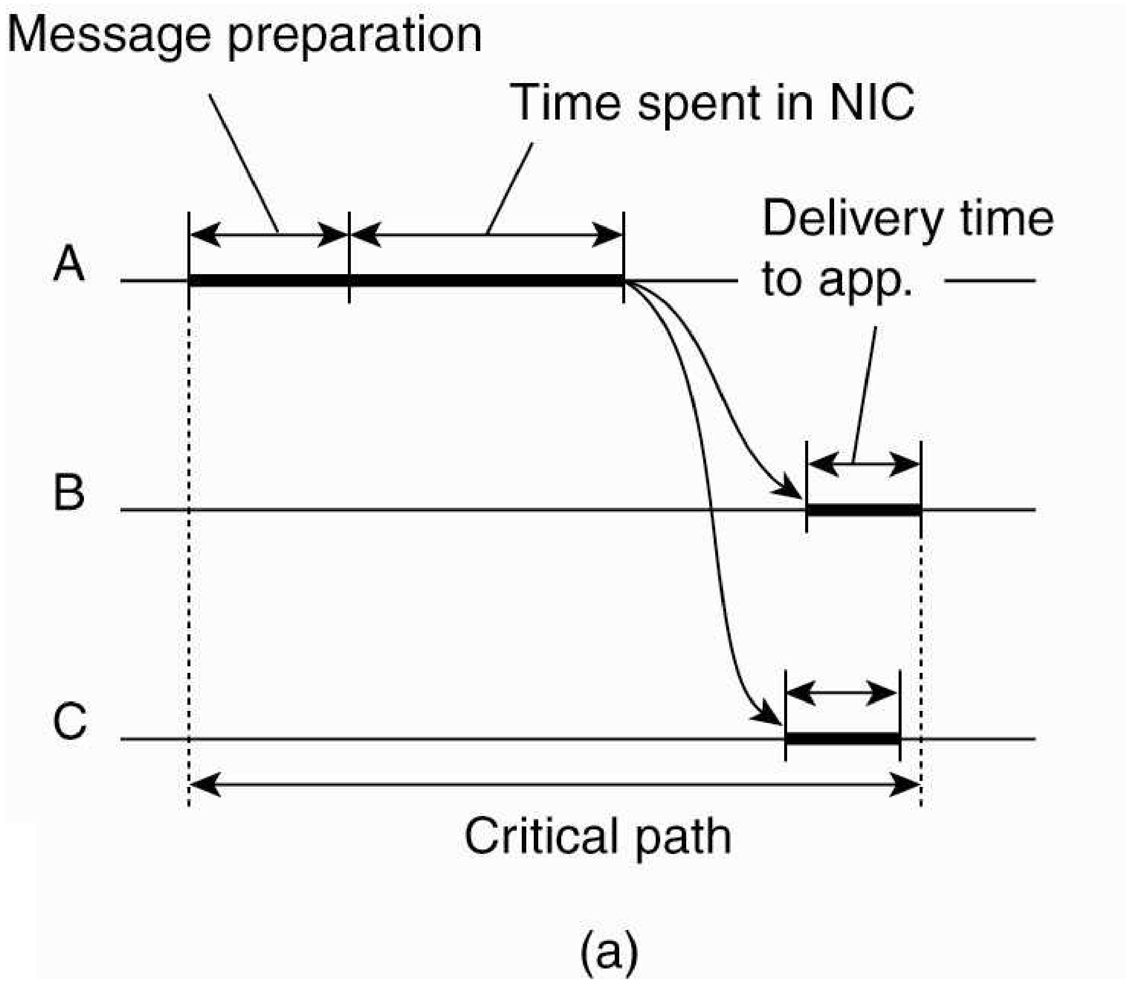


Figure 6-8. (a) The usual critical path in determining network delays.

$$Offset[p,q] = \frac{\sum_{k=1}^{M} (T_{p,k} - T_{q,k})}{M}$$

Clock Synchronization in Wireless Networks (cont.)

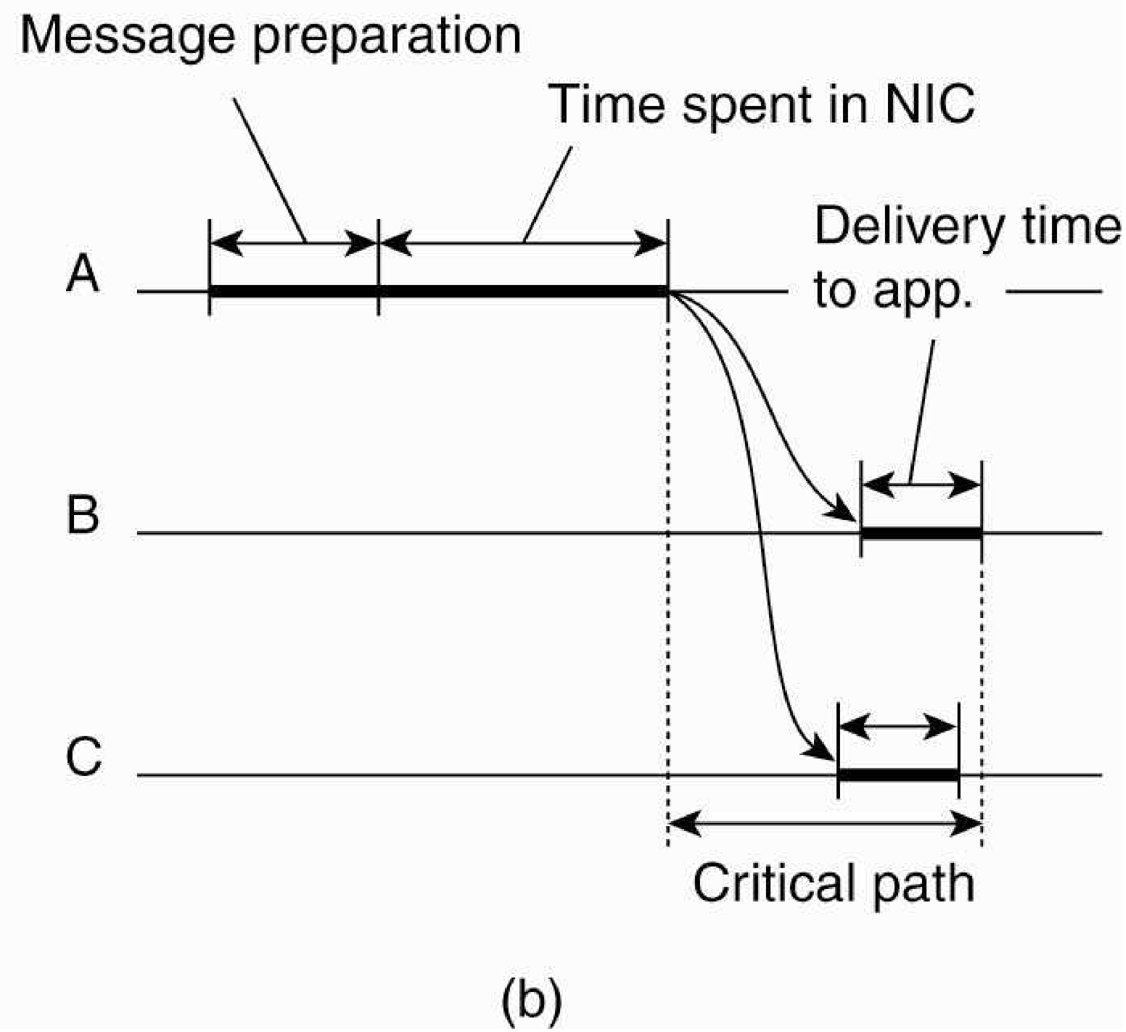


Figure 6-8. (b) The critical path in the case of RBS.

$$Offset[p,q](t) = \alpha t + \beta$$

Lamport's Logical Clocks

- To synchronize logical clocks, **Lamport** defined a relation called happens-before.
- The "happens-before" relation $\rightarrow$ can be observed directly in two situations:
  - If *a* and *b* are events in the same process, and *a* occurs before *b*, then $a \rightarrow b$ is true.
  - If a is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then $a \rightarrow b$

Lamport's Logical Clocks (cont.)

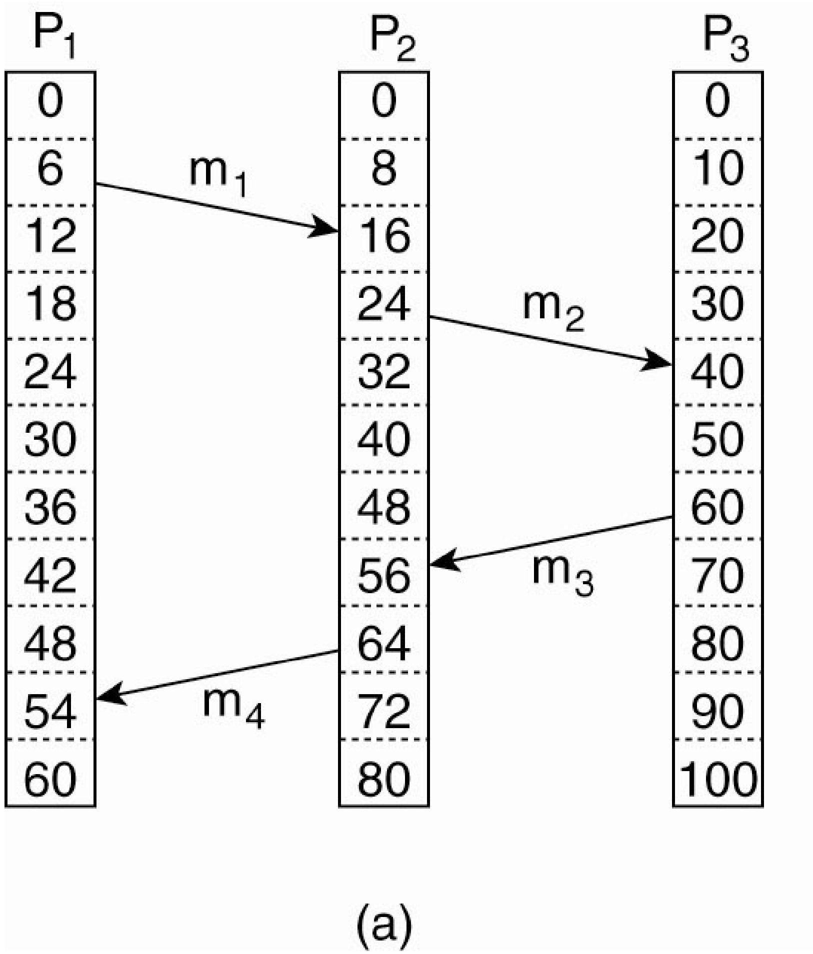


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates.

Lamport's Logical Clocks (cont.)

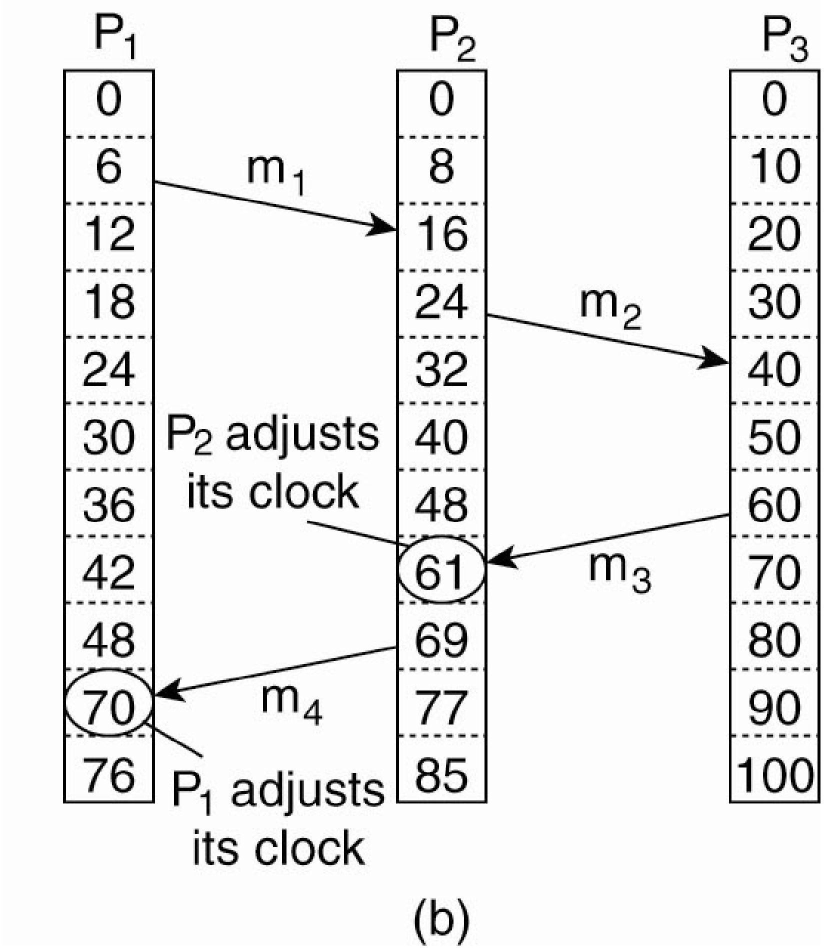


Figure 6-9. (b) Lamport's algorithm corrects the clocks.

Lamport's Logical Clocks (cont.)

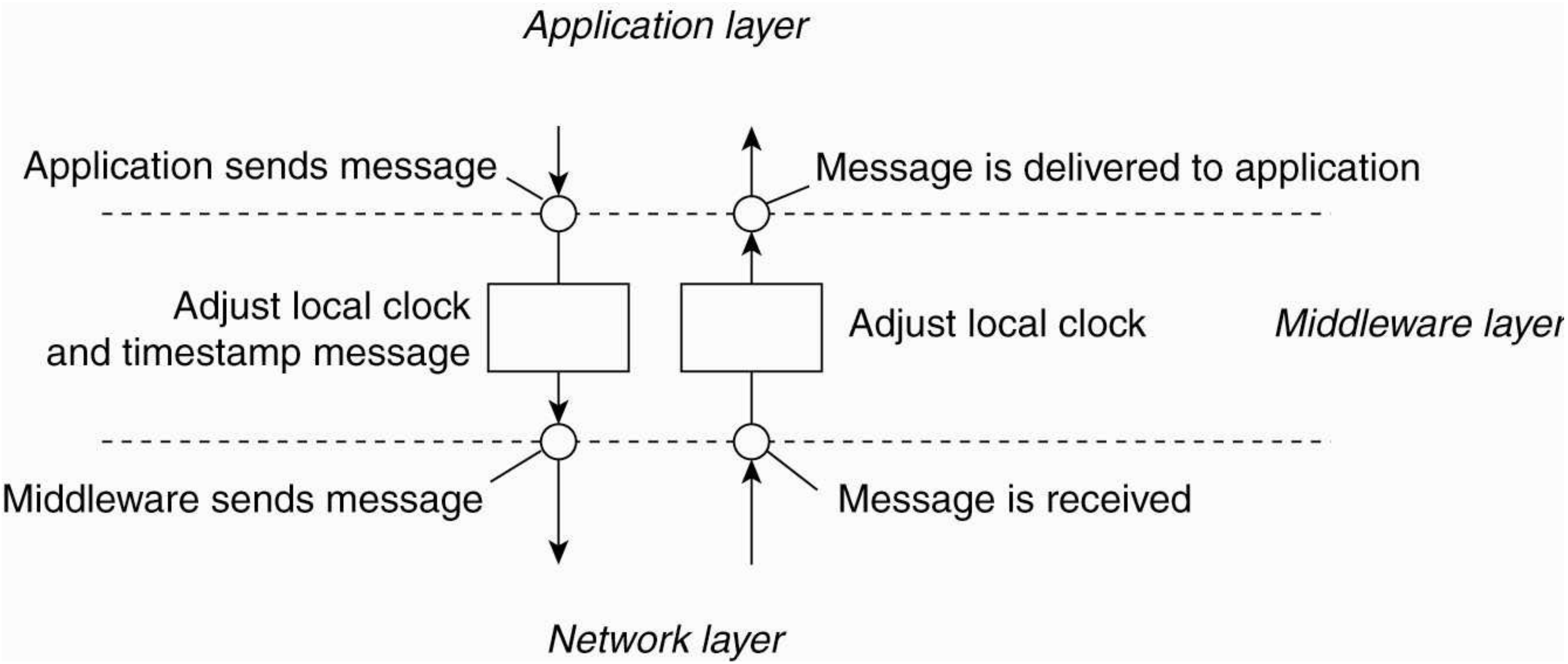


Figure 6-10. The positioning of Lamport's logical clocks in distributed systems.

Lamport's Logical Clocks (cont.)

Updating counter C_i for process P_i

1. Before executing an event P_i executes $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. Upon the receipt of a message *m*, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\}$, after which it then executes the first step and delivers the message to the application.

Vector Clocks

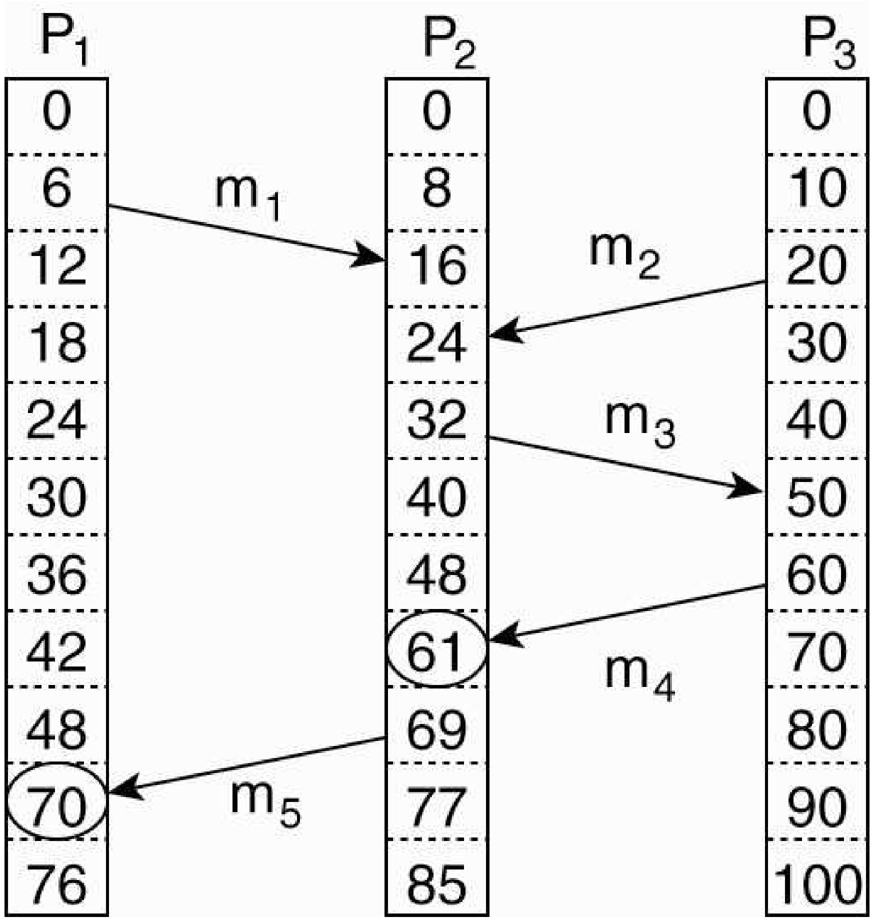


Figure 6-12. Concurrent message transmission using logical clocks.

Vector Clocks (cont.)

Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:

1. VC_i [i] is the number of events that have occurred so far at P_i. In other words, VC_i [i] is the local logical clock at process P_i .
2. If VC_i [j] = k then P_i knows that k events have occurred at P_j. It is thus P_i's knowledge of the local time at P_j .

Vector Clocks (cont.)

Steps carried out to accomplish property 2 of previous slide:

1. Before executing an event P_i executes $VC_i[\,i\,] \leftarrow VC_i[i] + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s (vector) timestamp *ts (m)* equal to VC_i after having executed the previous step.
3. Upon the receipt of a message m, process P_j adjusts its own vector by setting $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ for each *k*, after which it executes the first step and delivers the message to the application.

Enforcing Causal Communication

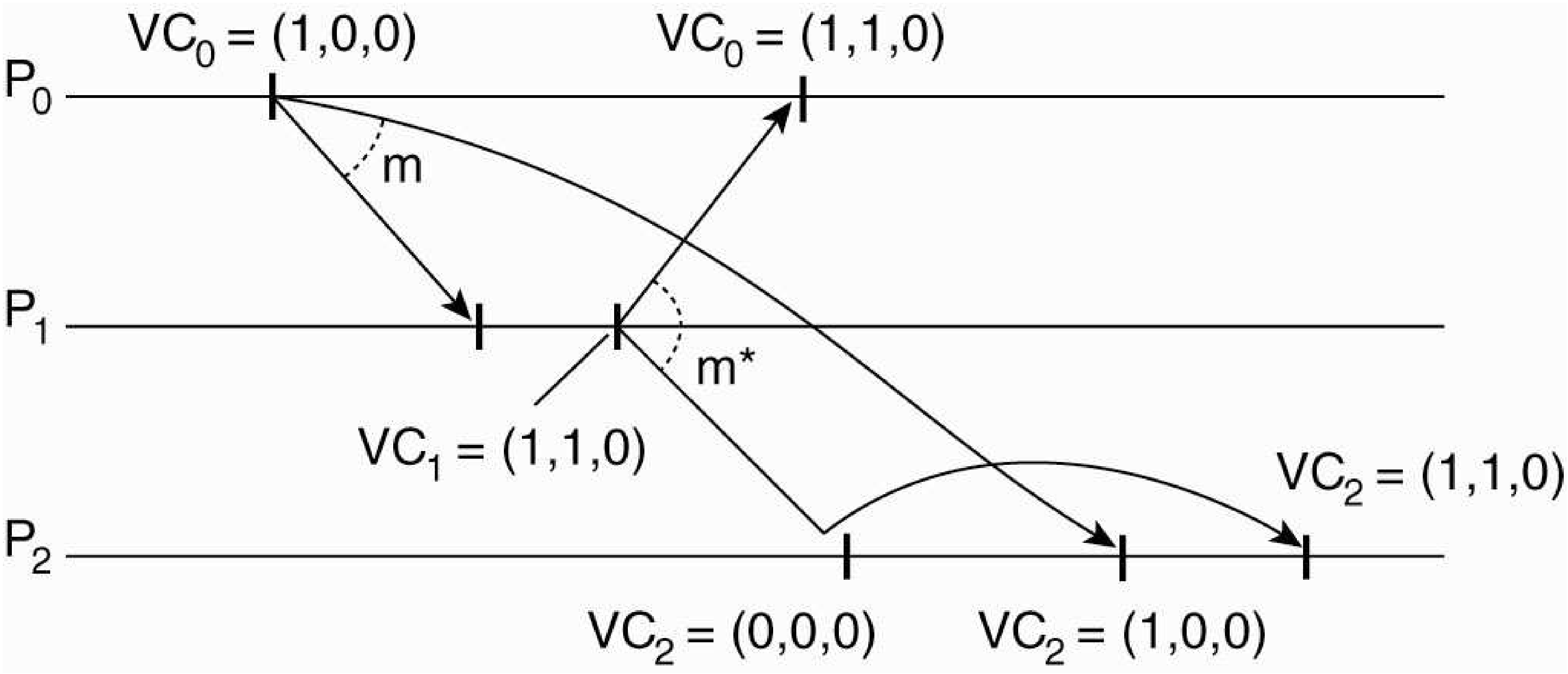


Figure 6-13. Enforcing causal communication.

Mutual Exclusion
A Centralized Algorithm

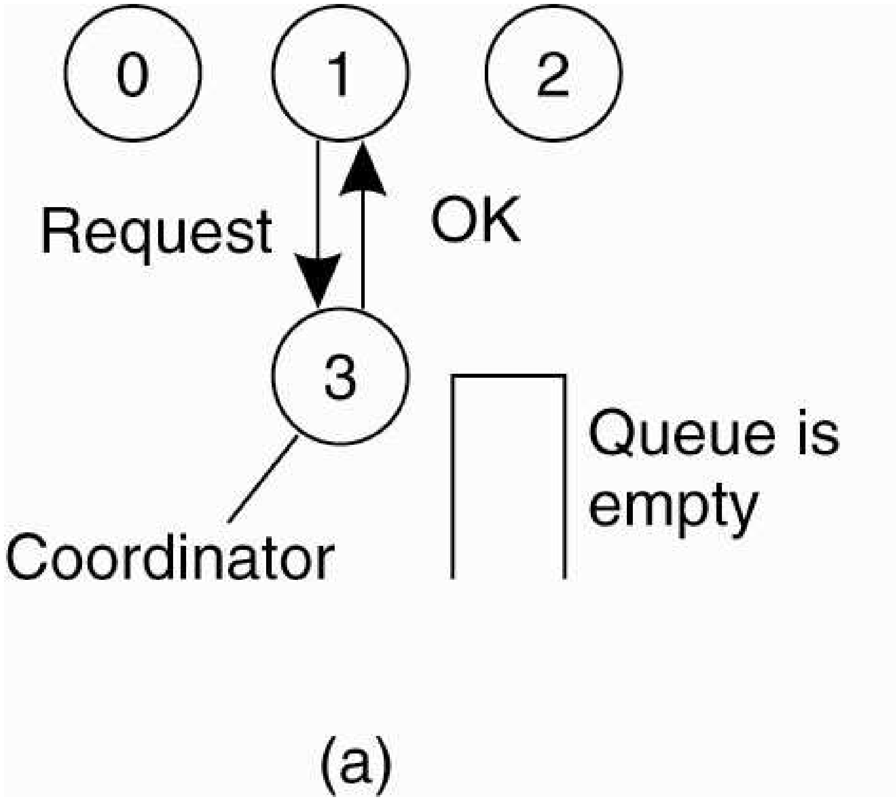


Figure 6-14. (a) Process 1 asks the coordinator for permission to access a hared resource. Permission is granted.

Mutual Exclusion
A Centralized Algorithm (cont.)

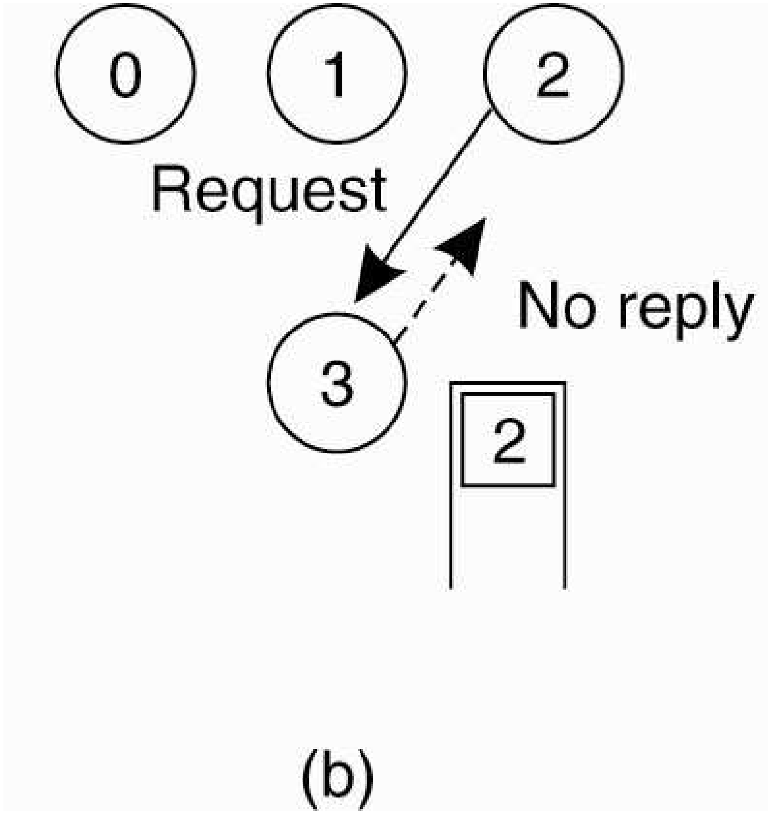


Figure 6-14. (b) Process 2 then asks permission to access the same resource. The coordinator does not reply.

Mutual Exclusion
A Centralized Algorithm (cont.)

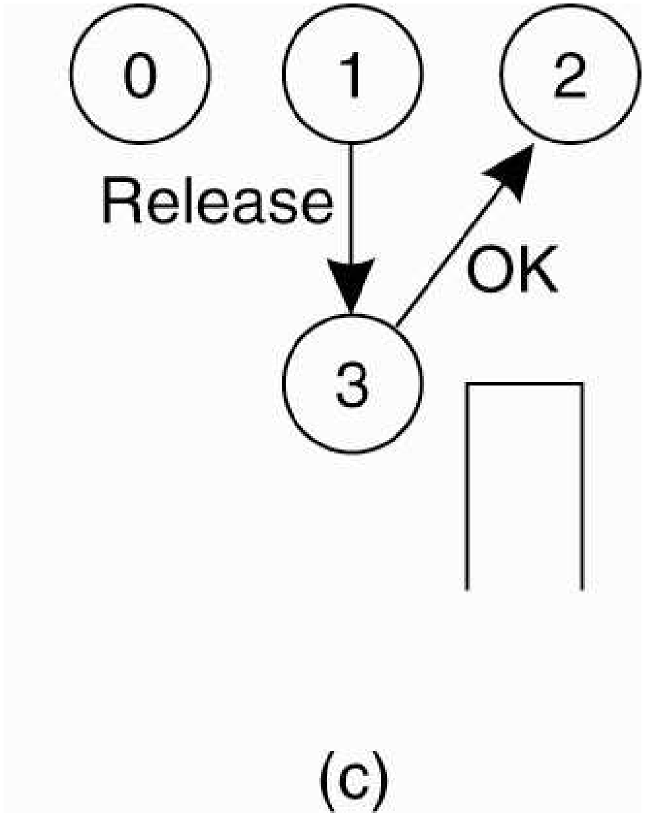


Figure 6-14. (c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.

Mutual Exclusion
A Distributed Algorithm

Three different cases:

1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

Mutual Exclusion
A Distributed Algorithm (cont.)

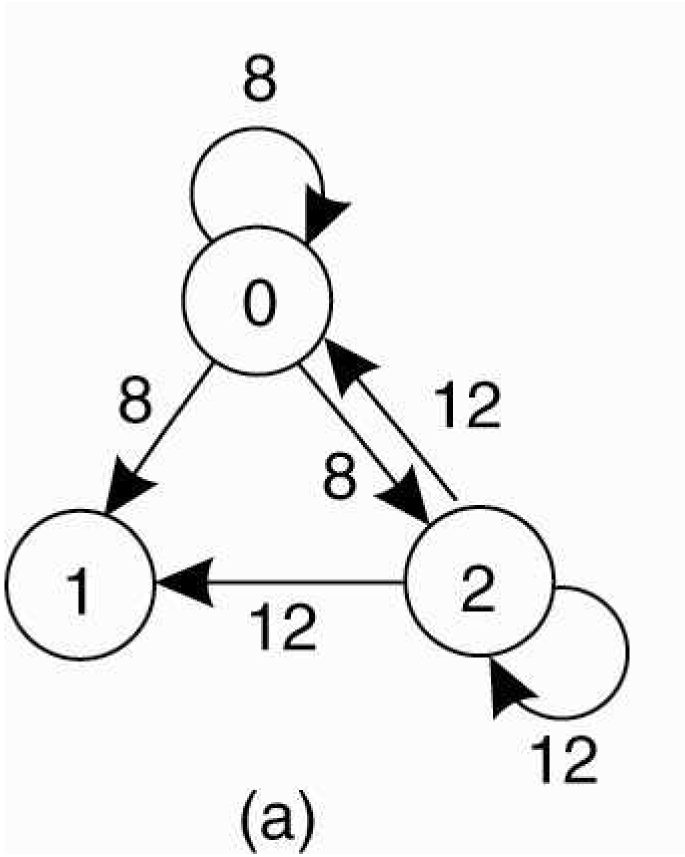


Figure 6-15. (a) Two processes want to access a shared resource at the same moment.

Mutual Exclusion
A Distributed Algorithm (cont.)

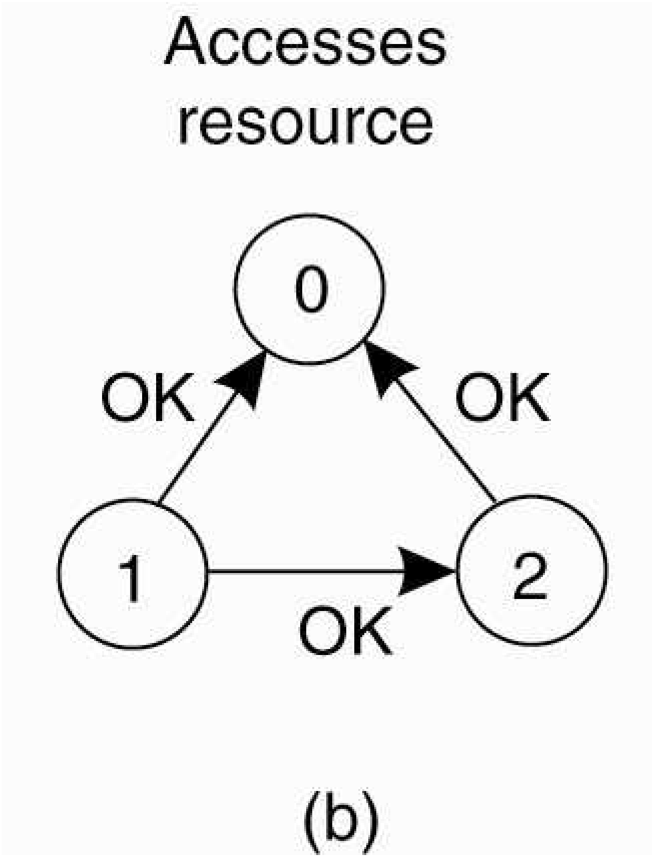


Figure 6-15. (b) Process 0 has the lowest timestamp, so it wins.

Mutual Exclusion
A Distributed Algorithm (cont.)

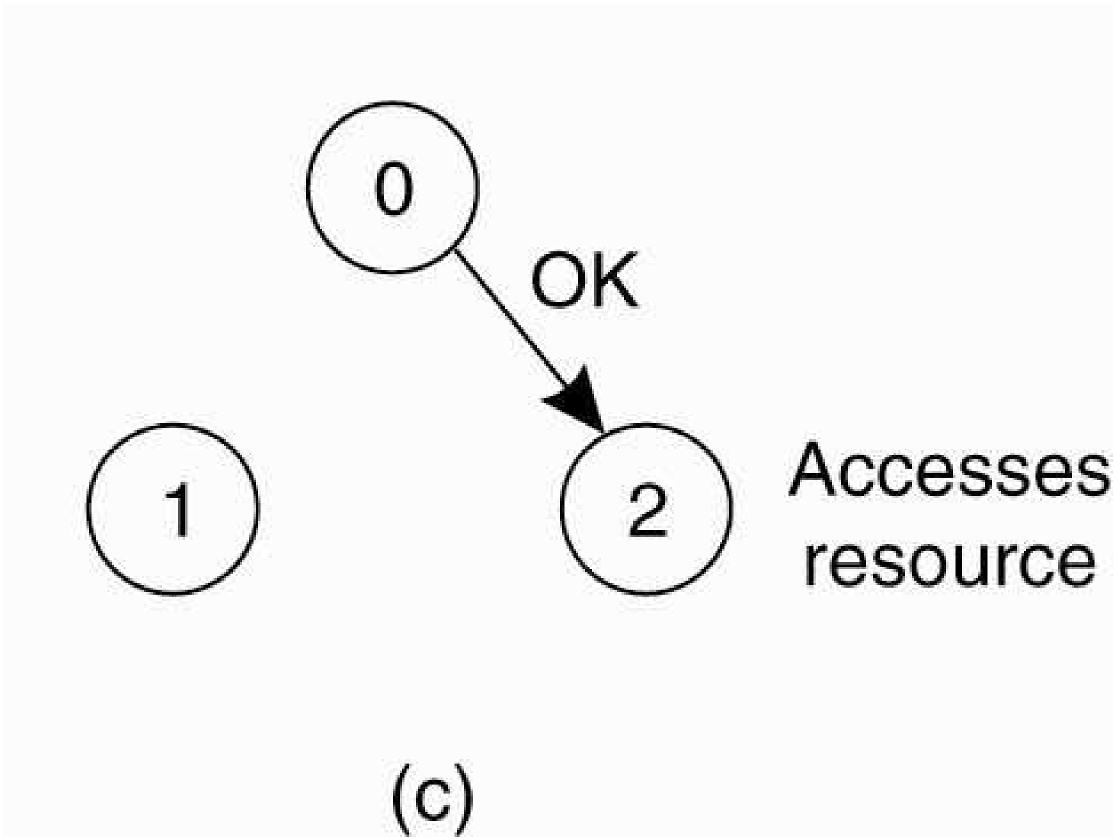


Figure 6-15. (c) When process 0 is done, it sends an OK also, so 2 can now go ahead.

Mutual Exclusion
A Token Ring Algorithm

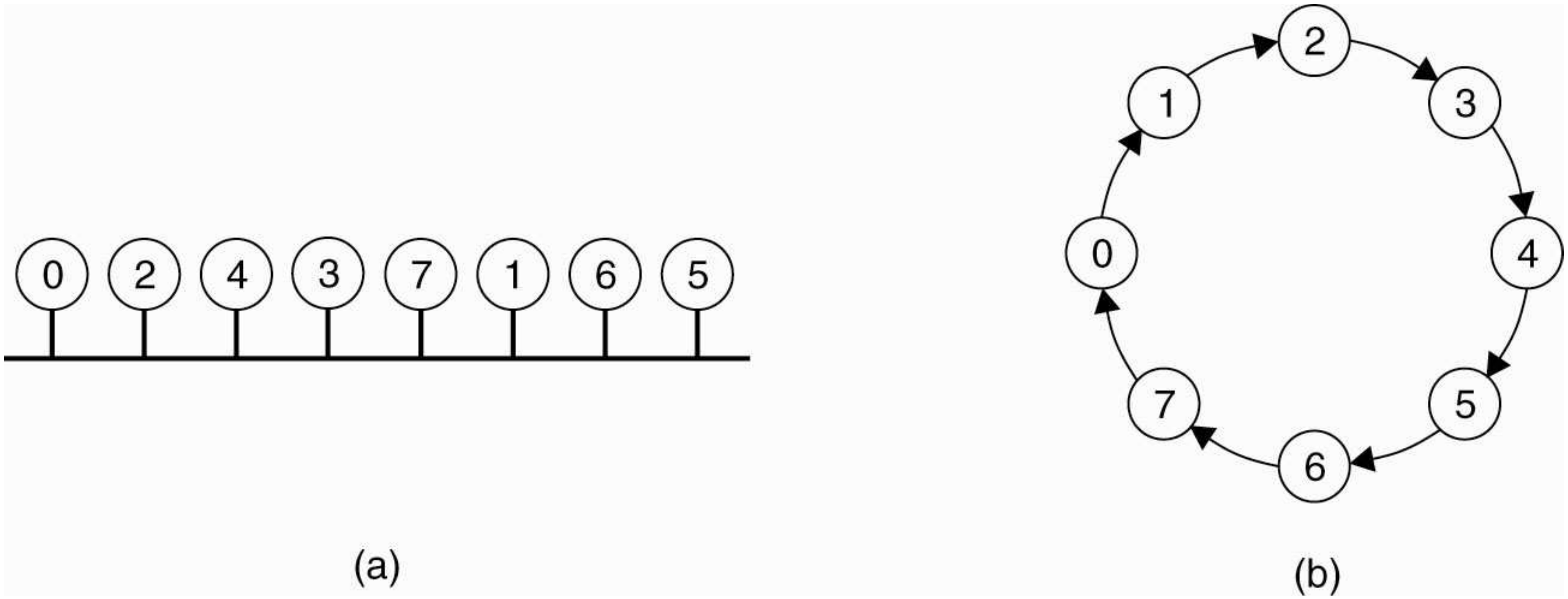


Figure 6-16. (a) An unordered group of processes on a network.
(b) A logical ring constructed in software.

A Comparison of the Four Algorithms

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	3mk, k = 1,2,...	2 m	Starvation, low efficiency
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Figure 6-17. A comparison of three mutual exclusion algorithms.

Global Positioning Of Nodes

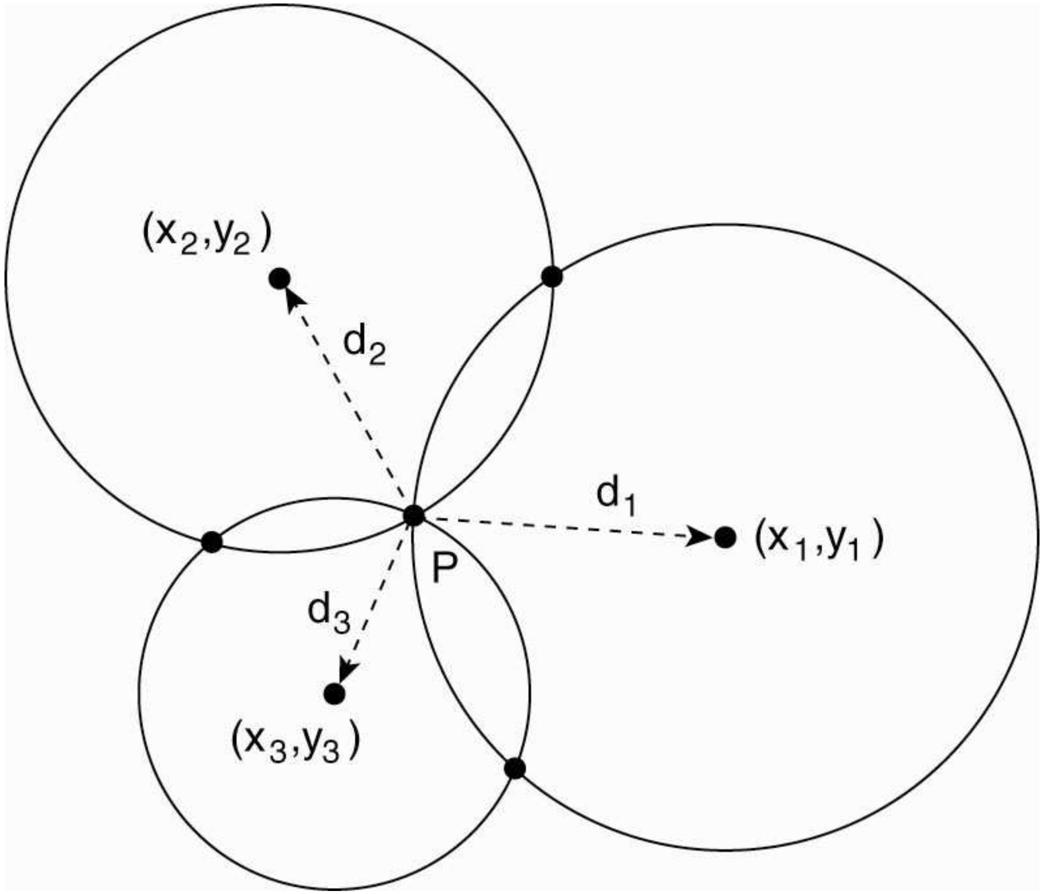


Figure 6-18. Computing a node's position in a two-dimensional space.

Global Positioning Of Nodes (Cont.)

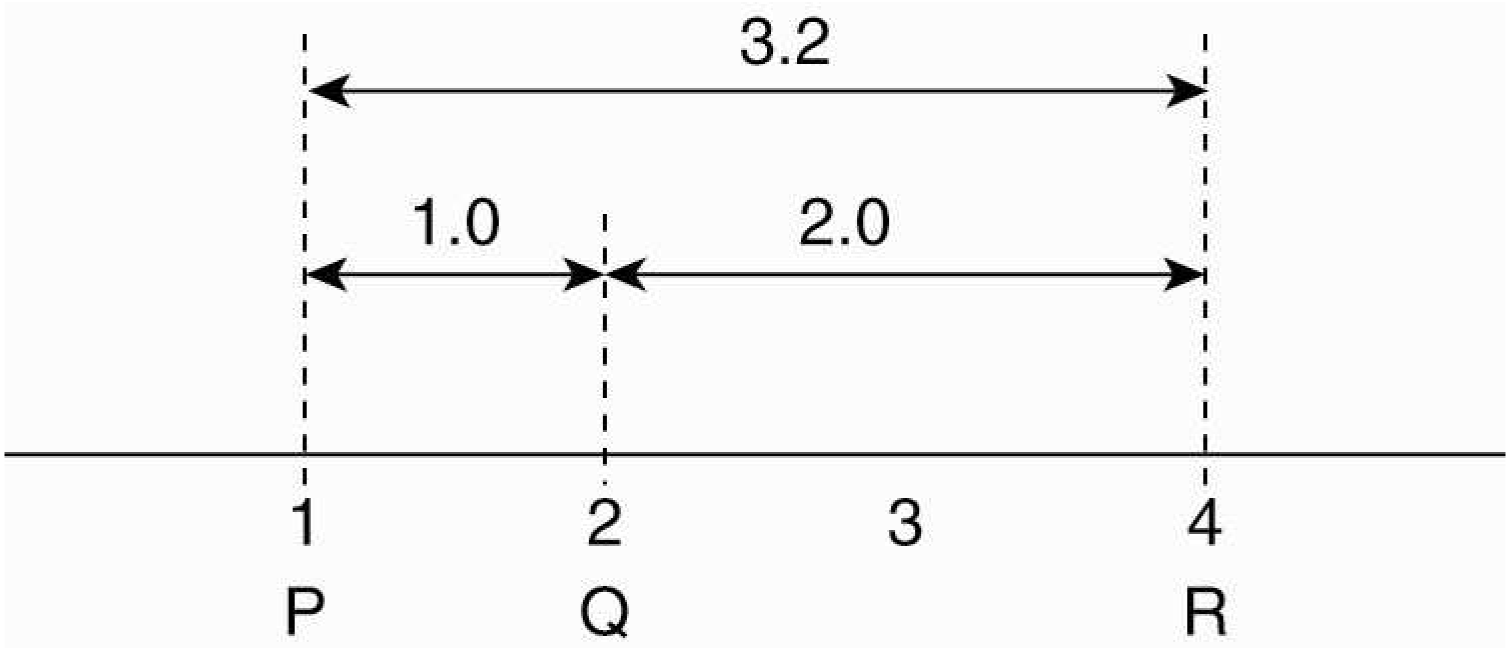


Figure 6-19. Inconsistent distance measurements in a one-dimensional space.

Election Algorithms

The Bully Algorithm

1. *P* sends an *ELECTION* message to all processes with higher numbers.
2. If no one responds, *P* wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. *P*'s job is done.

The Bully Algorithm

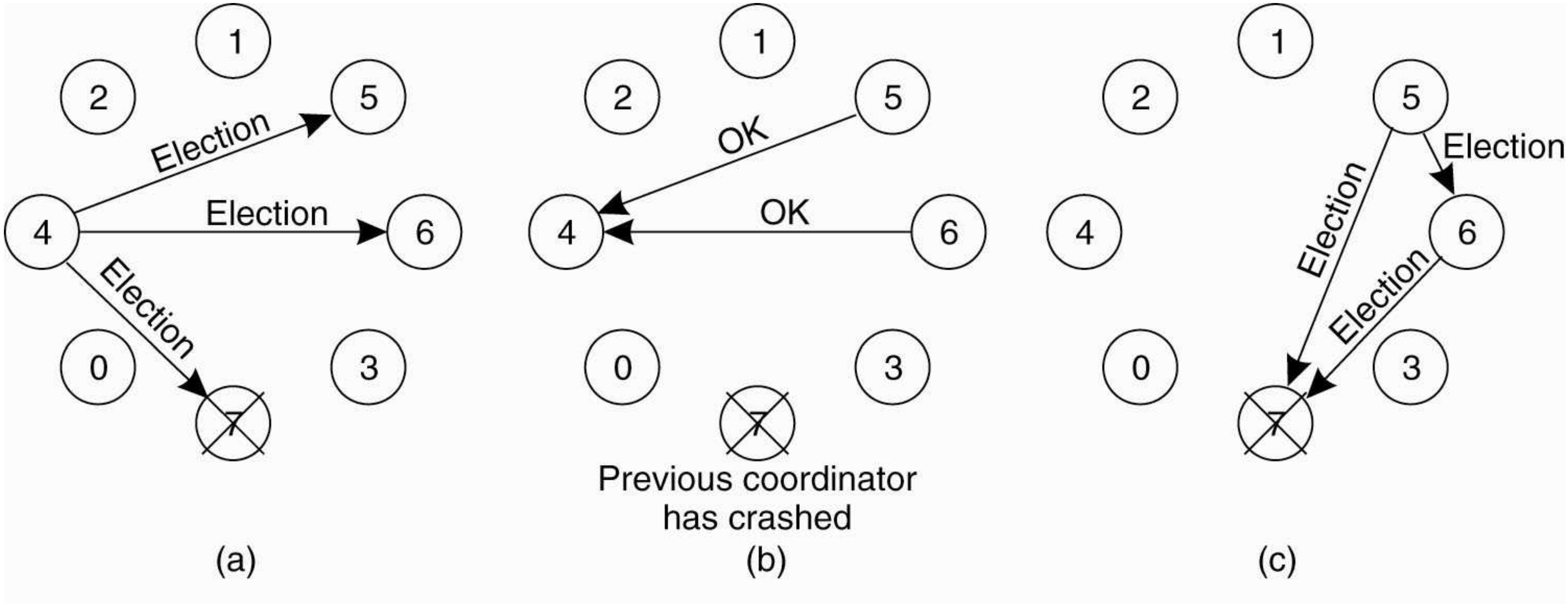


Figure 6-20. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election.

The Bully Algorithm (Cont.)

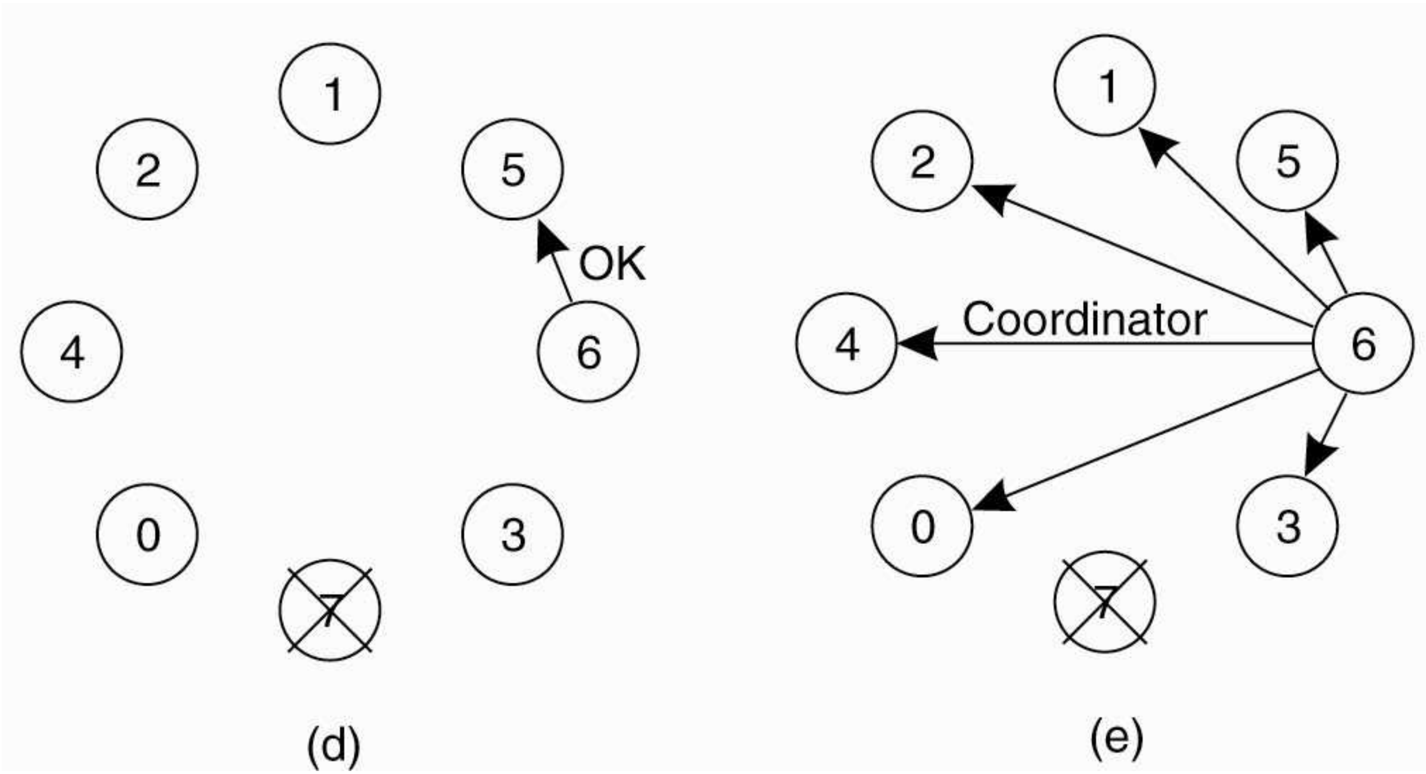


Figure 6-20. The bully election algorithm. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

A Ring Algorithm

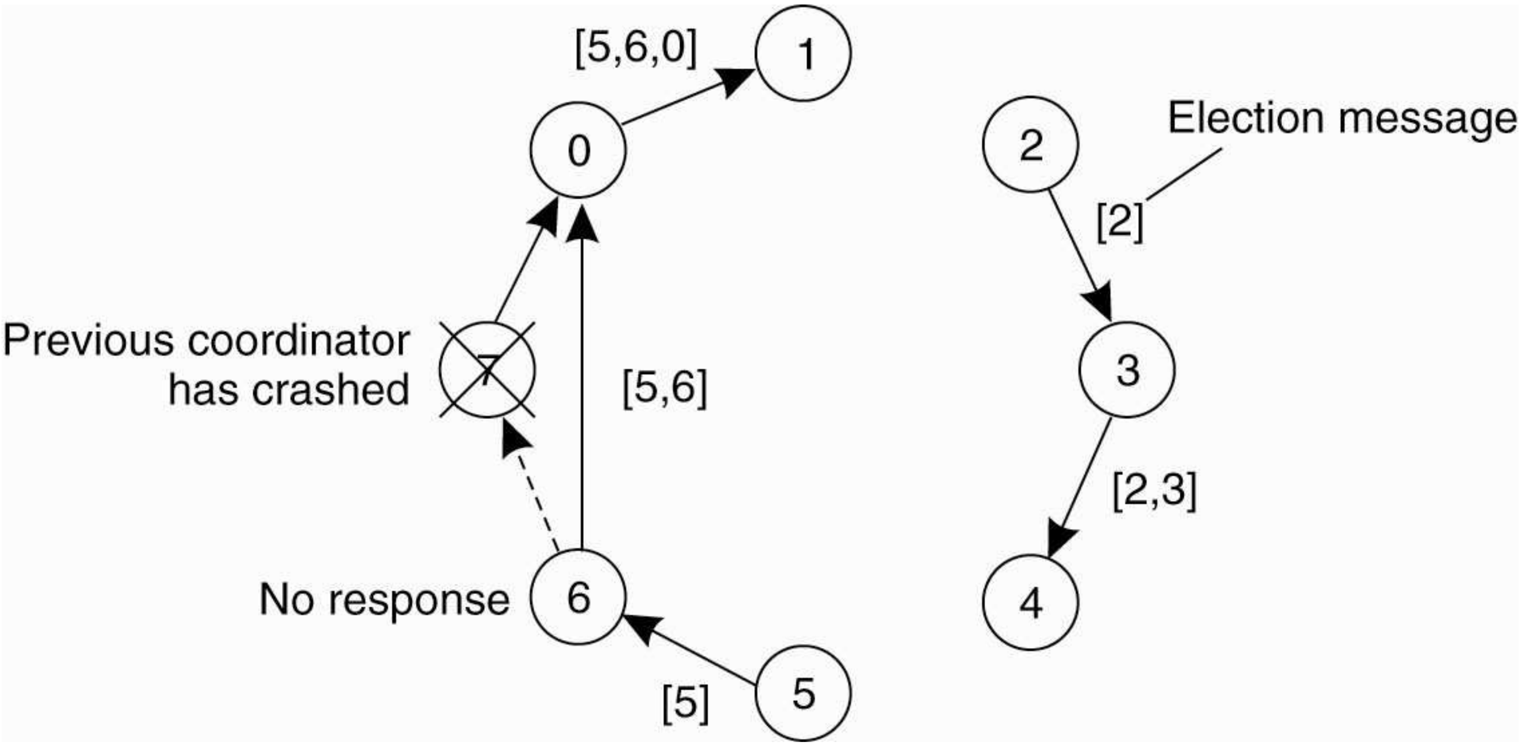


Figure 6-21. Election algorithm using a ring.

Elections in Wireless Environments

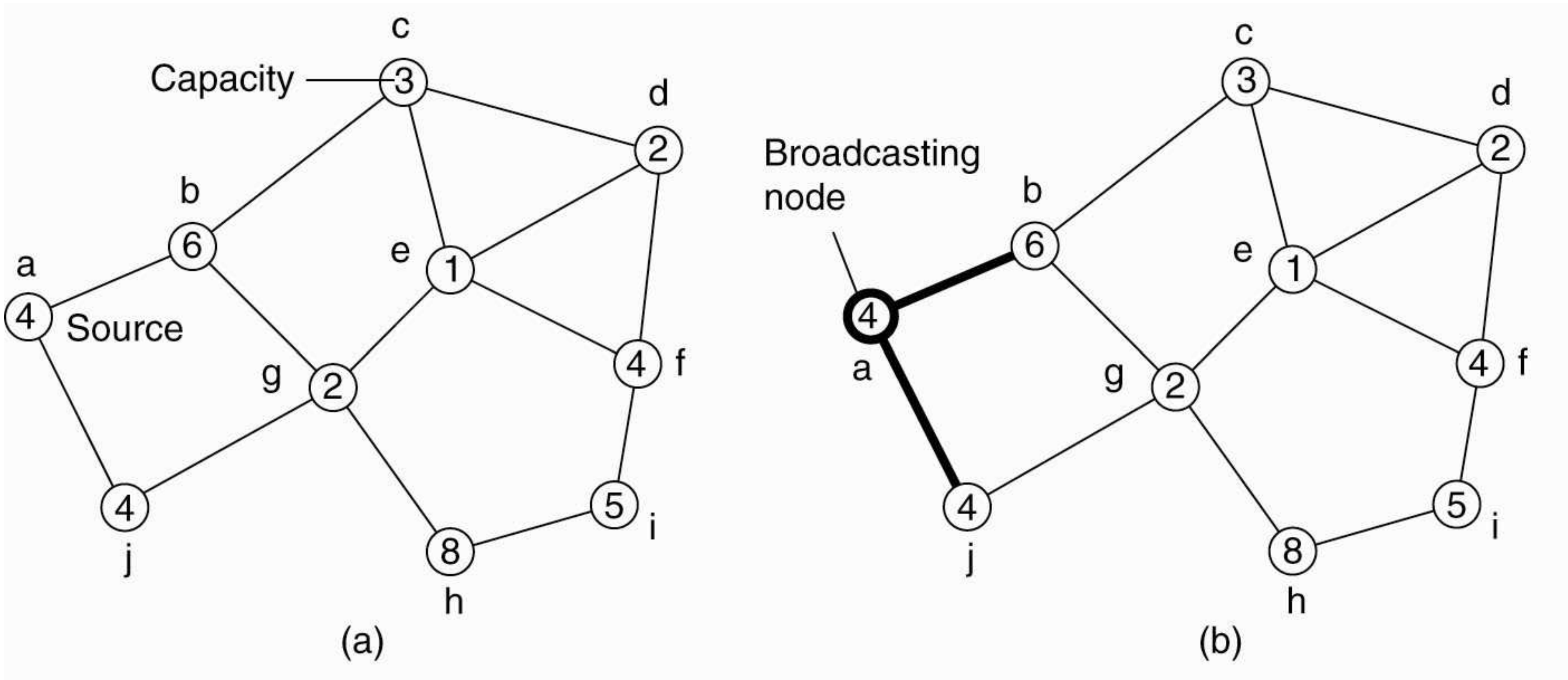


Figure 6-22. Election algorithm in a wireless network, with node a as the source. (a) Initial network. (b)–(e) The build-tree phase

Elections in Wireless Environments (Cont.)

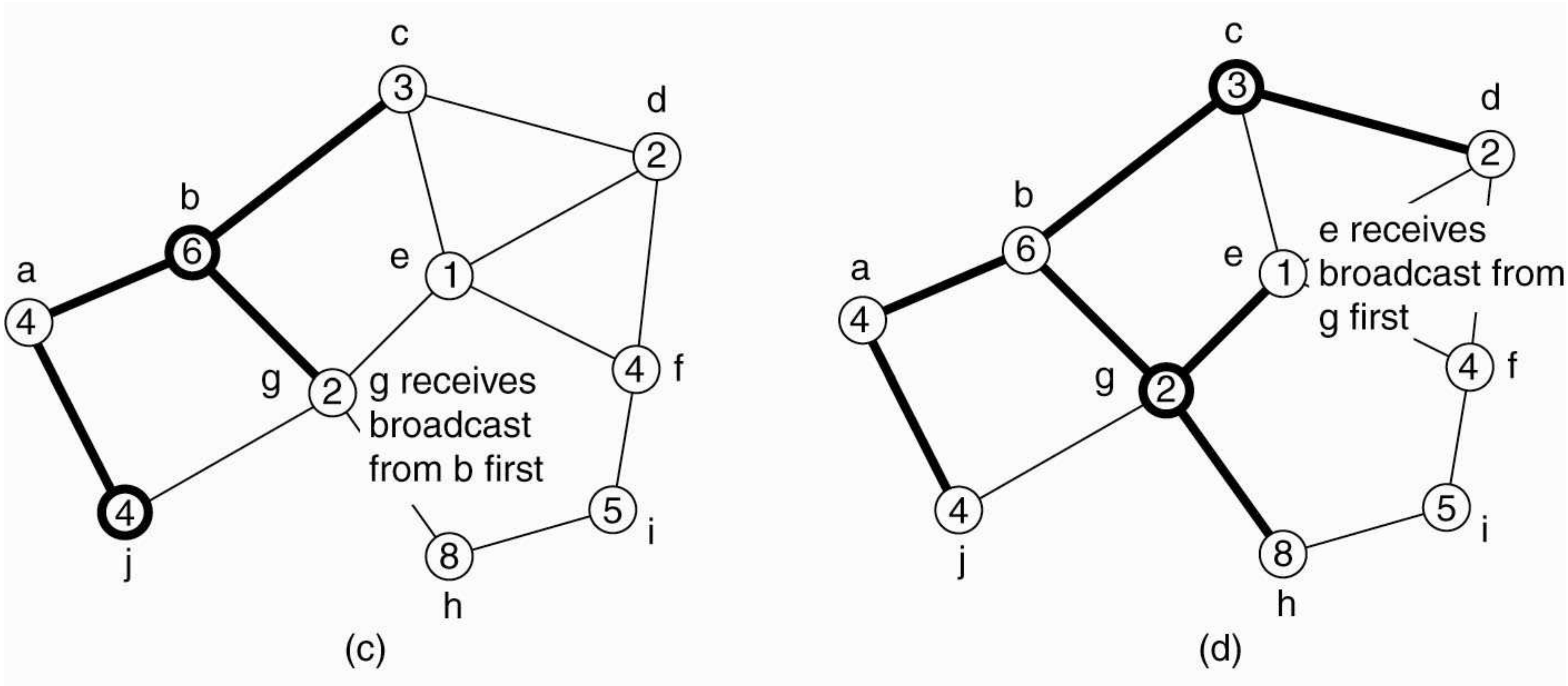


Figure 6-22. Election algorithm in a wireless network, with node a as the source. (a) Initial network. (b)–(e) The build-tree phase

Elections in Wireless Environments (Cont.)

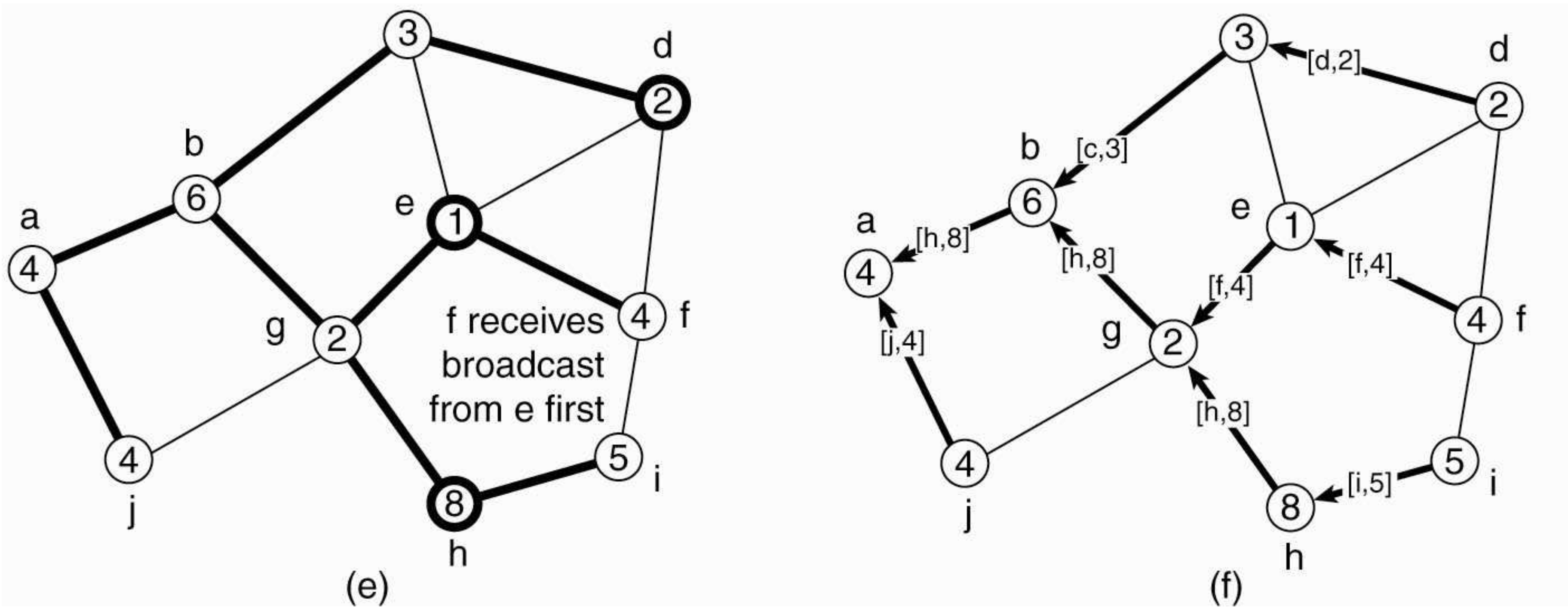


Figure 6-22. (e) The build-tree phase.
(f) Reporting of best node to source.

Elections in Large-Scale Systems

Requirements for superpeer selection:

1. Normal nodes should have low-latency access to superpeers.
2. Superpeers should be evenly distributed across the overlay network.
3. There should be a predefined portion of superpeers relative to the total number of nodes in the overlay network.
4. Each superpeer should not need to serve more than a fixed number of normal nodes.

Elections in Large-Scale Systems (Cont.)

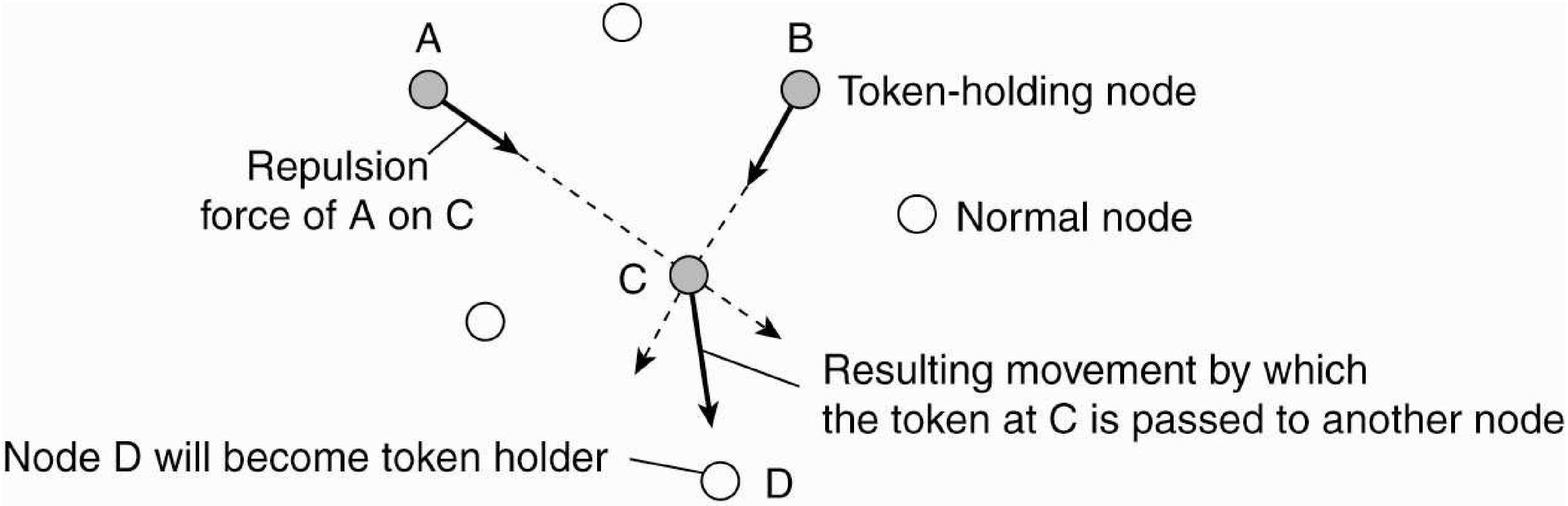


Figure 6-23. Moving tokens in a two-dimensional space using repulsion forces.