[8] Haritsa, J. R., M. Livny, and M. J. Carey: "Earliest Deadine Scheduling for Real-Time Database Systems," *Proc. IEEE Real-Time Systems Symp.*, pp. 232–242, IEEE, Los Alamitos, CA, 1991.

[9] Huang, J., J. A. Stankovic, D. Towsley, and K. Ramamritham: "Experimental Evaluation of Real-Time Transaction Processing," *Proc. IEEE Real-Time Systems Symp.*, pp. 144–153, IEEE, Los Alamitos, CA, 1989.

[10] Huang, J., J. A. Stankovic, D. Towsley, and K. Ramamritham: "On Using Priority Inheritance in Real-Time Databases," *Proc. IEEE Real-Time Systems Symp.*, pp. 210–221, IEEE, Los Alamitos, CA, 1991.

[11] Kim, W., and J. Srivastava: "Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Based Disk Scheduling," *Proc. IEEE Real-Time Systems Symp.*, pp. 222–231, IEEE, Los Alamitos, CA, 1991.

[12] Korth, H. F., N. Soparkar, and A. Silberschatz: "Triggered Real-Time Databases with Consistency Constraints," *Proc. 16th Conference on Very Large Data Bases*, pp. 95–116, Morgan-Kaufmann, Palo Alto, CA, 1990.

[13] Kung, H. T., and J. T. Robinson: "On Optimistic Methods for Concurrency Control," *ACM Trans. Database Systems* 6(2):213–226, 1981.

[14] Lehman, T. J., and M. J. Carey: "A Study of Index Structures for Main Memory Database Management Systems," *Proc. 12th Conference on Very Large Data Bases*, pp. 294–303, Morgan-Kaufmann, Palo Alto, CA, 1986.

[15] Lin, Y., and S. H. Son: "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *Proc. IEEE Real-Time Symp.*, pp. 104–112, IEEE, Los Alamitos, CA, 1990.

[16] Lortz, V. B.: An Object-Oriented Real-Time Database System for Multiprocessors, CSE-TR-210-94, Dept. of Electrical Engineering and Computer Science, University of Michigan, 1994.

[17] O'Neil, P. E., K. Ramamritham, and C. Pu: A Two-Phase Approach to Predictable Scheduling Real-Time Transactions, technical report 92-35, Computer Science Department, University of Massachusetts, 1992.

[18] Pang, H., M. Livny, and M. J. Carey: "Transaction Scheduling in Multiclass Real-Time Database Systems," *Proc. IEEE Real-Time Systems Symp.*, pp. 23–34, IEEE, Los Alamitos, CA, 1992.

[19] Papadimitriou, C.: *The Theory of Database Concurrency Control,* Computer Science Press, Rockville, MD, 1986.

[20] Ramamritham, K.: "Real-Time Databases," *Distributed and Parallel Databases* 1:199–226, 1993.

[21] Yu, P. S., H.-U. Heiss, and D. M. Dias: "Modeling and Analysis of a Time-Stamp History Based Certification Protocol for Concurrency Control," *IEEE Trans. Knowledge and Data Engineering* 3(4):525–537, 1991.

[22] Yu, P. S., K.-L. Wu, K.-J. Lin, and S. H. Son: "On Real-Time Databases: Concurrency Control and Scheduling," *Proc. IEEE* 82(1):140–157, 1994.

CHAPTER 6

REAL-TIME COMMUNICATION

6.1 INTRODUCTION

Effective communication between the various devices of a real-time system is vital to its correct functioning. In embedded systems, data flows from the sensors and control panels to the central cluster of processors, between processors in the central cluster, and from processors to the actuators and output displays. The communication overhead adds to the computer response time. Hard real-time systems must therefore use communication protocols that allow the communication overhead to be bounded.

In soft real-time systems, such as multimedia or videoconferencing, where voice and image data are being transmitted, the need to deliver messages in a timely fashion is equally obvious; excessive delays in message delivery can significantly degrade the quality of service provided. However, in such applications, the occasional failure to meet message-delivery deadlines is not fatal.

The goals of communication protocols in real-time systems are somewhat different from those in traditional nonreal-time data-communication systems. In traditional systems, the key performance measure is system throughput, that is, how much data can be transferred over the network in one unit time from source to destination. In real-time systems, the key measure is the probability of delivering a message by a certain deadline. Note that a lost message has an infinite delivery time, so that this measure captures both the speed with which messages are delivered and the probability of losing messages. Message delay is caused by the following overheads.

- Formatting and/or packetizing the message.
- Queueing the message, as it waits for access to the communication medium.
- Sending the message from the source to the destination.
- Deformatting the message.

Real-time traffic can typically be categorized into multiple message classes, with each class being characterized by its deadline, arrival pattern, and priority. In hard real-time systems, such as embedded applications, the deadline of the traffic is related to the deadline of the task to which that communication belongs. In multimedia-type applications, the deadline is related directly to the application.

Priority is based on the importance of that message class to the application. If there is an overload of traffic, message priority can be used to determine which messages are dropped to ensure that the more important traffic is delivered in a timely fashion.

Most real-time sources generate traffic that fall into one of the following two categories.

Constant rate: Fixed-size packets are generated at periodic intervals. Many sensors produce such traffic. Constant rate traffic is the easiest to handle since it is smooth and not bursty. The smoother the traffic, the smaller the number of buffers that must be provided at each node.

Variable rate: This may take the form of fixed-size packets being generated at irregular intervals or of variable-size packets being generated at regular intervals. Bursty traffic makes greater demands on buffer space. Voice and video traffic typically exhibit variable rates. For example, voice sources can consist of *talkspurts* (a burst of packets, followed by a period of silence). See Figure 6.1. Video packets are an example of variable-sized packets being generated at regular intervals.

As mentioned earlier, traffic characteristics may change as packets flow through multiple hops in a network. At these intermediate nodes, the various traffic classes compete for bandwidth on the node's output link, and thus interfere with one another. Consider, for example, a high-priority bursty traffic class (Class 1) competing with a lower-priority constant-rate traffic class (Class 2) at some node n. See Figure 6.2. Since Class 1 has priority at node n, it will cause Class 2 to pile up at that node over the intervals $[0, a]$ and $[b, c]$. As a result of this, the output of Class 2 from this node will also become bursty.

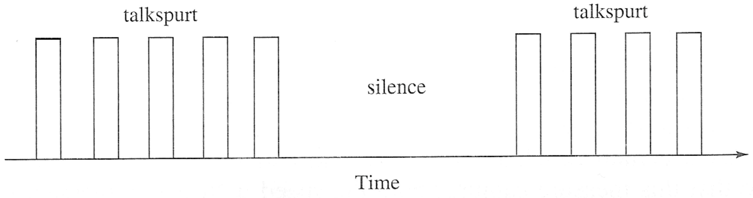
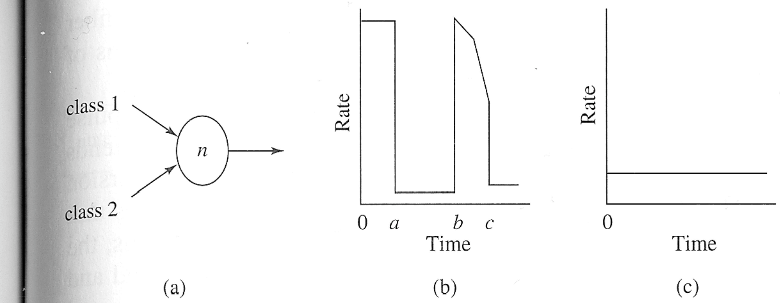


FIGURE 6.1
Voice traffic.



(a) (b) (c)

FIGURE 6.2
(a) Two competing classes at a node; (b) Class 1 (high-priority, bursty); (c) Class 2 (lower-priority, constant).

6.1.1 Communications Media

While most of this chapter will be devoted to studying communications protocols, it is useful to be aware of the underlying physical communications medium. Each medium has a distinct set of properties. Let us consider the three most important media.

ELECTRICAL MEDIUM. This is the medium with which readers will be most familiar. Electrical media are manifested as a twisted pair of wires or as coaxial cable. Coaxial cable has a copper conductor at its core, surrounded by some insulation, surrounded by an outer conductor, and finally surrounded by a plastic coating. Twisted wires have bandwidths of several kHz, while broadband coaxial cable bandwidth can be as high as 450 MHz.

Devices can be connected to coaxial cables either by breaking the cable and inserting a T-junction (illustrated in Figure 6.3) or by using a vampire tap. A vampire tap is constructed by drilling through the outer layers of the cable to its core and connecting the core to the device via a conductor.

OPTICAL FIBERS. In a system with an optical medium, the electrical signals from the nodes are converted to light pulses by means of laser diodes. These diodes can be operated at up to 10 Gbps (technological improvements keep increasing

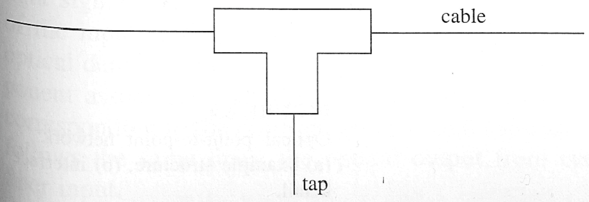


FIGURE 6.3
A T-junction.

this bound). The optical pulses thus generated are then launched on the fiber medium. The receiver then converts them back to electrical signals by means of photodiodes at the receiver.

As a light pulse travels down a fiber, two things happen to it. First, the pulse amplitude decreases, that is, the signal is *attenuated*. Second, the pulse width tends to increase with the distance travelled, a phenomenon called *dispersion*. Dispersion is linked to the nature of the optical fiber as well as the range of frequencies that are transmitted (physical transmitters cannot transmit pure sine waves). Thus, the size of a network is linked to the power that is needed at the transmitter end and to the maximum frequency that can be supported.

Optical fiber has two main advantages over electrical media. First, the raw bandwidth of a typical fiber can be as high as several hundred GHz. Second, optical signals are immune to the effects of electromagnetic interference.

One disadvantage with optical fiber is that it is difficult to passively tap them without a significant signal loss. Optical amplifiers to restore signal levels are expensive, and so taps are impractical unless the system is very small. There are two network structures that work well with optical fiber, point-to-point networks and the passive star.

In *point-to-point networks*, there are no taps, but there are optical-to-electric and electric-to-optical converters at each interface. See Figure 6.4. The first stage of the interface converts optical signals into electrical ones. The node then checks to see if the message was addressed to it or to someone downstream on the network. If the former, it accepts the message. If the latter, it retransmits the message by using the second stage of the interface, which converts the electrical signal from the node to light.

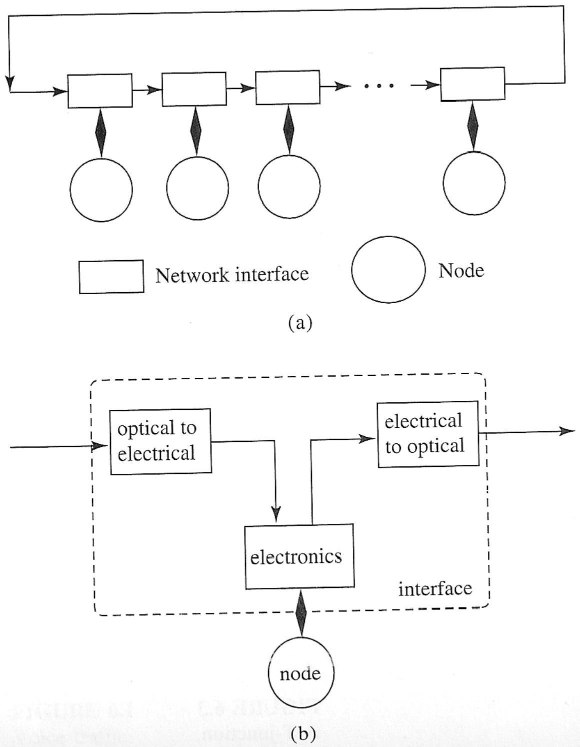


FIGURE 6.4
Optical point-to-point network: (a) example structure; (b) interface detail.

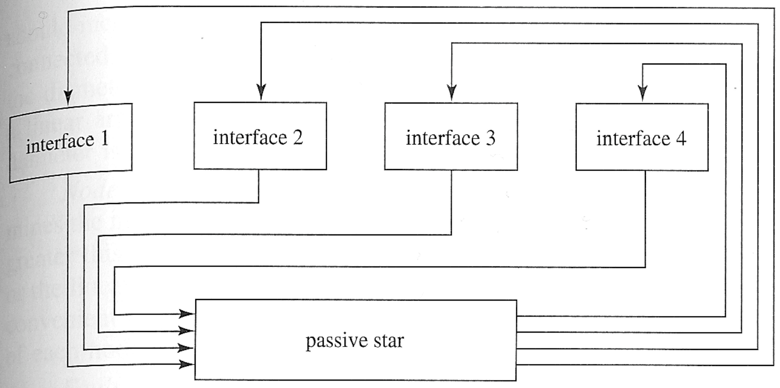


FIGURE 6.5
Star-configured architecture.

The passive star arrangement is as shown in Figure 6.5. Each interface delivers its optical output to a glass cylinder, the *passive star*, which delivers at its output the sum of all the signals at the input. This output energy is divided among the fibers that go out of the cylinder. Thus, every interface receives the input for all the interfaces in the system, and simply picks the ones addressed to it. For this arrangement to work, the output from the passive star must have sufficient energy to be detected by each of the interfaces even after being divided among them. This requires sensitive receivers or powerful transmitters, or both.

The fiber bandwidth is, as we have explained above, of the order of hundreds of GHz. This is well above the speed of the electronics at the interface. This mismatch requires us to either run the fiber at the speed of the interface electronics, which would waste most of the fiber bandwidth, or use *wavelength-division multiplexing* (WDM) to divide the fiber channel into several virtual channels, each of sufficiently low bandwidth to match the interface-electronics bandwidth. The details of how this is done is out of the scope of this book. (Section 6.4 contains sources about WDM.) All that we need to know here is that the single physical channel is divided into multiple virtual channels, each with its own interface. Figure 6.6 shows how such a fiber is connected to the node interface receiver. Its light signal is split into a number of channels, each of which carries a fraction of the input signal. Each such channel has an interface attached to it, with an optical detector tuned to the wavelength of a virtual channel. The interface component associated with each channel provides its node with an electrical signal corresponding to the corresponding virtual channel. The transmission of messages follows the same lines; the optical output from each channel is summed at the fiber input.

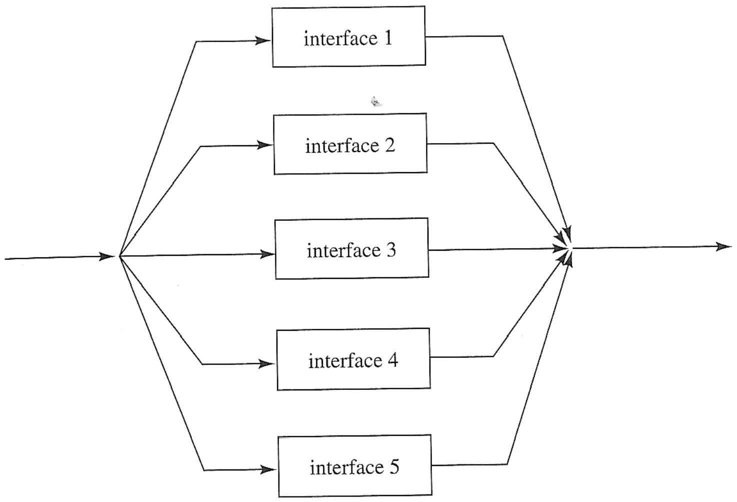


FIGURE 6.6
Interfacing to a WDM network.

The design of tunable lasers and receivers is currently the focus of much research. The challenge is to produce devices that can be tuned at high speed to enable a transmitter or receiver to hop through a wide range of the frequency spectrum; however, this area is outside the scope of this book.

WIRELESS. There has recently been mounting interest in wireless communication (using microwave radio) between computers. The advantage of such a medium is that it does not require wired contact between communicating nodes. As a result, ad hoc networks can be rapidly set up and reconfigured. However, the potential for interference is much greater than for either electrical or optical media. The distance over which a radio link can be maintained depends on the transmitter power, receiver sensitivity, noise levels, type of error-control coding used, and any attenuating barriers (e.g., walls, partitions, equipment, filing cabinets) between the transmitter and the receiver. At the time of writing, wireless bandwidths are around a few megabits/second (Mb/s). This is expected to increase with time. Very little work on real-time protocols specifically geared toward wireless links has been done; the protocols described in this chapter are geared towards electrical and optical media.

6.2 NETWORK TOPOLOGIES

The network topology for a computer or a distributed system must be carefully chosen since it affects the system response time and reliability. The following features are important.

Diameter: This is the maximum distance (number of hops) between any two nodes in the system, as a function of the number of nodes. Ideally, the diameter should increase only slowly as a function of the number of nodes. In a completely connected network, where each node has a dedicated link to every other node, the diameter is one, regardless of the number of nodes; at the other extreme, in a linear array (where the nodes are connected as a one-dimensional string), the diameter is the number of nodes minus one.

Node degree: This is the number of edges adjacent to each node, and determines the number of I/O ports per node and the number of links in the system. The greater this number is, the greater the cost. In some networks (such as the mesh or the linear array) the node degree is independent of the number of nodes: this is convenient since adding nodes to a network does not then require the architecture of each node to change.

Fault-tolerance: This measures the extent to which the network can withstand the failure of individual links and nodes while still remaining functional. The *node (link) connectivity* is the minimum number of nodes (links) that must fail before the network is disconnected. The *node (link) diameter stability* is the minimum number of nodes (links) that must fail before the network diameter is increased. There is no single measure that adequately captures the fault-tolerant capability of a network.

Network topologies can be broadly classified into point-to-point and shared (or broadcast) categories. In a *point-to-point* topology, nodes are connected by dedicated links. If a node wishes to send a message to a destination that is not its neighbor, that message must be forwarded by intermediate nodes. In a *shared* (or *broadcast*) topology, the nodes all have access to the communications channel and only one node can transmit at any time over a channel.

Example 6.1. Figure 6.7 shows examples of point-to-point and shared networks. In the point-to-point network, there is no edge connecting nodes 1 and 3. A message going from node 1 to node 3 must thus pass through either node 2 or node 4. That is, it must take two hops from input to output. In contrast, if the shared network is used, a node can communicate with any other node in one hop.

Buses and rings are the most popular topologies. Figure 6.8 shows examples of them. In a bus network, the ends are terminated by matching impedances to sharply attenuate reflections. The interfaces can either consist of taps or of forwarding points.

A ring network is a set of network interfaces connected in a ring by point-to-point links. Bits arriving at the input end of an interface are copied into a buffer. They can then be processed (if necessary) and transmitted at the output end of the interface.

Example 6.2. Node 1 wishes to send a message to node 5 through nodes 2, 3, and 4. It transmits its message, with a header to specify the destination, to node 2. The node-2 interface receives the message and checks the message destination. Realizing that it is for node 5, it transmits the message, unaltered, to the node-3 interface. This in turn forwards it to node 4, which forwards it to node 5. Node 5 reads in the

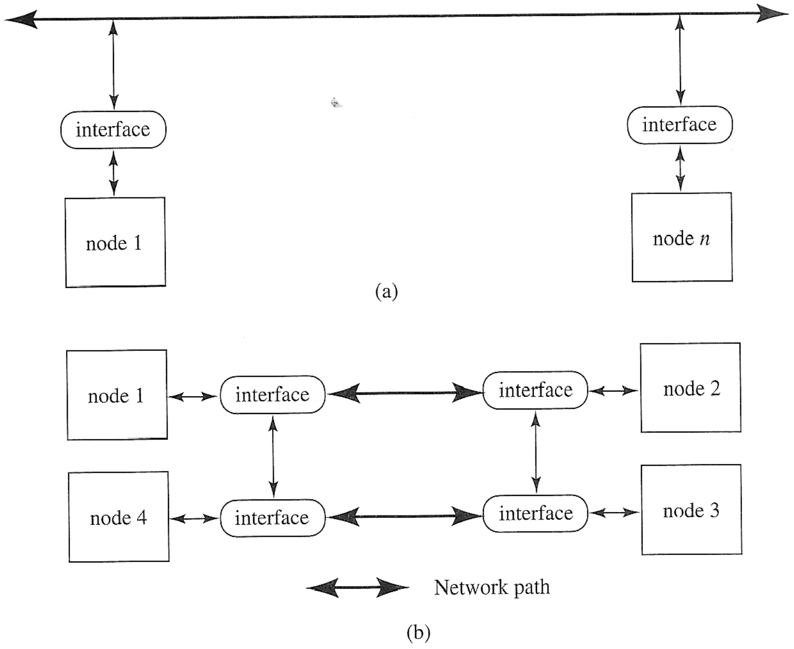


FIGURE 6.7
(a) Shared network and (b) point-to-point network.

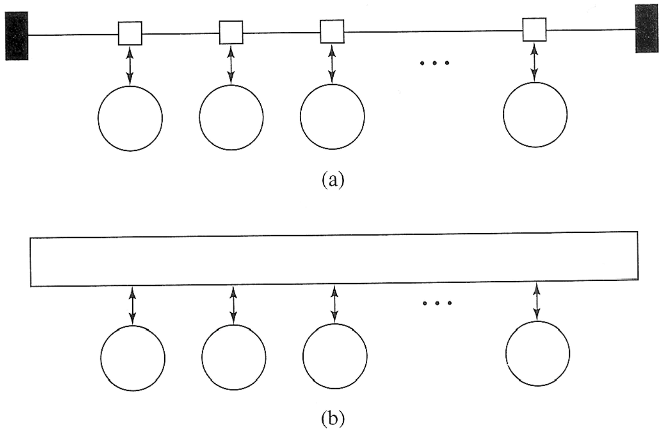


FIGURE 6.8
(a) A single-bus system and (b) a ring system.

message and passes it along to the next node. Ultimately, the message comes back to node 1, which removes it from the ring.

Other popular topologies are illustrated in Figures 6.9 and 6.10. The hypercube (see Figure 6.10) is defined as follows. There are 2^n nodes in an n-dimensional hypercube. Label the nodes in binary from 0 to $2^n - 1$, and connect by a line those nodes whose labels differ in exactly one bit position. An n-dimensional hypercube is built by taking two $(n - 1)$-dimensional hypercubes and connecting like nodes.

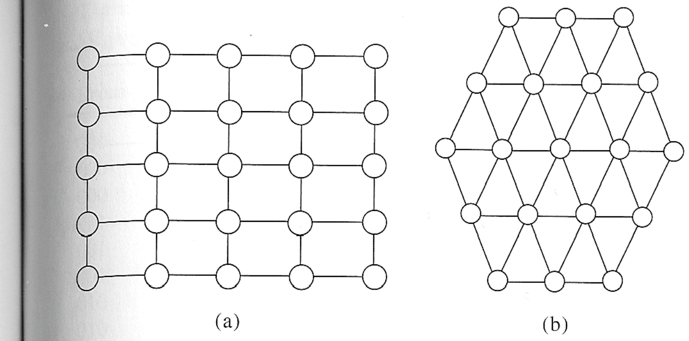


FIGURE 6.9
Mesh networks: (a) rectangular mesh; (b) hexagonal mesh.

Another popular topology is the multistage network. This is built out of switchboxes, typically 2 × 2 switchboxes. Four possible configurations of an individual switch are shown in Figure 6.11. An 8-input, 8-output network built out of such switches is shown in Figure 6.12.

Many structures can support either a point-to-point or a shared topology. Consider the single-bus system in Figure 6.8, which uses forwarding interfaces. If each interface, upon receiving a message at one end, copies it out on the other regardless of its destination, the network behaves as a broadcast topology. On the

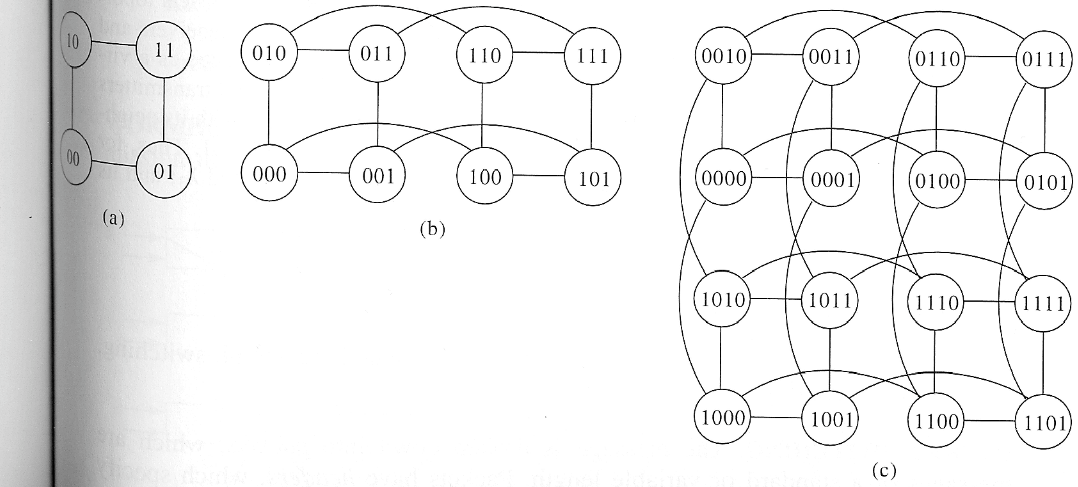


FIGURE 6.10
Hypercube networks: (a) two-dimensional; (b) three-dimensional; (c) four-dimensional.

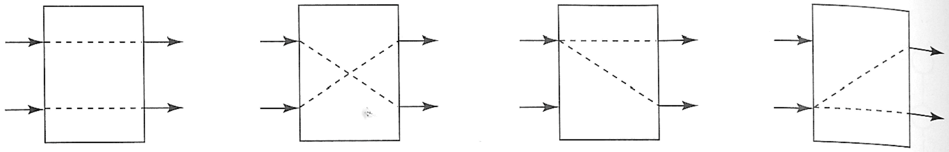


FIGURE 6.11
Four configurations of a 2 × 2 switch.

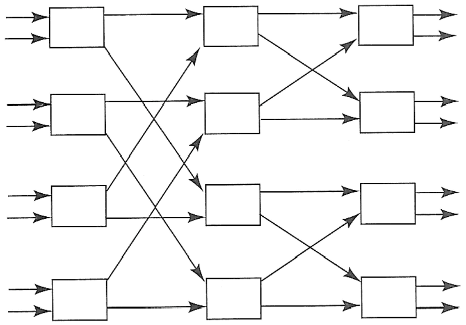


FIGURE 6.12
Multistage network.

other hand, if the interface checks the destination and only forwards the message if the path to the destination lies through it, the network behaves as a point-to-point topology.

It is sometimes important to distinguish between the physical and virtual topologies. The *physical* topology is the structure determined by the physical connections. On this base, we can sometimes build many different virtual topologies. This is illustrated in Example 6.3.

Example 6.3. We have an optical network using a passive star as its physical topology and using wavelength-division multiplexing. Each node has three receivers and three transmitters. We want this to support a three-dimensional hypercube as a virtual network. This is done by assigning wavelengths to the receivers and transmitters so that each node has a unique wavelength on which it communicates with its neighbor in the hypercube. The wavelength allocations are shown in Table 6.1. Thus, for example, node 000 has its transmitters tuned to wavelengths λ_1, λ_2, and λ_3; and its receivers tuned to λ_4, λ_7, and λ_{13}.

6.2.1 Sending Messages

Three common ways of sending messages are: packet switching, circuit switching, and wormhole routing.

PACKET SWITCHING. The message is broken down into *packets*, which are messages of a standard or variable length. Packets have *headers*, which specify their source, destination, and any other information that may be required. They are then sent to their destination by the routing and flow-control algorithm.

TABLE 6.1
Wavelength allocations for embedding a hypercube in a passive star

From	To	Wavelength	From	To	Wavelength
000	001	λ_1	100	000	λ_{13}
000	010	λ_2	100	101	λ_{14}
000	100	λ_3	100	110	λ_{15}
001	000	λ_4	101	001	λ_{16}
001	011	λ_5	101	100	λ_{17}
001	101	λ_6	101	111	λ_{18}
010	000	λ_7	110	010	λ_{19}
010	011	λ_8	110	100	λ_{20}
010	110	λ_9	110	111	λ_{21}
011	001	λ_{10}	111	011	λ_{22}
011	010	λ_{11}	111	101	λ_{23}
011	111	λ_{12}	111	110	λ_{24}

Example 6.4. The network is a two-dimensional hypercube shown in Figure 6.10. Node 00 is sending a packet to node 11. There are two possible paths: 00 → 01 → 11, and 00 → 10 → 11. Suppose the first of these is chosen. Node 00 sends the packet to node 01, which notes its destination from the packet header, and forwards the packet to that node.

CIRCUIT SWITCHING. A circuit is set up between the source and the destination for such time as is required to send the message. The entire circuit is then meant exclusively for this message; any other messages that require all, or part, of this path must wait for the transmission to be completed. In other words, circuit-switching involves setting up a dedicated path from source to destination for the duration of the message transfer.

Example 6.5. In the multistage network shown in Figure 6.13, if we wish to send a message from S to D, the switches are set as shown by the heavy line and the circuit is held until the message has been delivered.

WORMHOLE ROUTING. Wormhole routing is a way of pipelining packet transmission in a multihop network. Each packet is broken down into a train of flits,

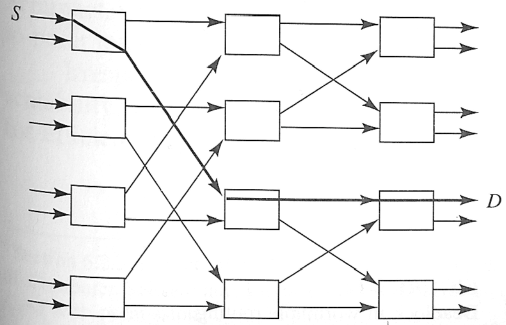


FIGURE 6.13
Setting up a circuit.

each about one or two bytes long. The sender transmits one flit per unit time, and the flits are forwarded from node to node until they reach their destination. Over time, a train of flits, in contiguous nodes, forms and makes its way to its destination.

Example 6.6. The network is a three-dimensional hypercube, as shown in Figure 6.10. A message is to be sent from node 000 to node 111. Node 000 breaks up its packet into flits, and sends them to node 001 at the rate of one flit per cycle. Node 001 forwards the flits it receives to node 011, which forwards them to 111. If the packet consists of six flits, the activity is as follows.

Time	000 →001	001 →011	011 →111
0	Flit 0	–	–
1	Flit 1	Flit 0	–
2	Flit 2	Flit 1	Flit 0
3	Flit 3	Flit 2	Flit 1
4	Flit 4	Flit 3	Flit 2
5	Flit 5	Flit 4	Flit 3
6	–	Flit 5	Flit 4
7	–	–	Flit 5

The final flit is received at the destination at time 7.

Only the header flit in a train has the destination information; each node simply forwards the next flit to the same node that it sent the previous flit to in the train. It is therefore impossible to interleave one train of flits with another; successive flits in the same train must be in either the same or adjacent nodes. Wormhole routing requires less buffer space in the forwarding nodes, since nodes deal with flits rather than packets.

If we are not careful, multiple trains of flits can cause deadlock. Figure 6.14 shows an example. Train α is prevented from turning at node m since that would cause it to be interspersed with train β. Similarly, β is prevented from turning at node p due to train γ; γ is prevented from turning at node q due to train δ; δ is prevented from turning at n due to α. The situation is illustrated graphically in Figure 6.15, where an arrow from node α to node δ indicates that train α is stopping δ. The graph is cyclic, indicating a deadlock.

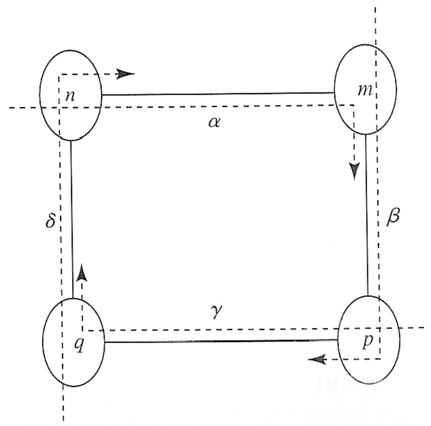


FIGURE 6.14
Deadlock in wormhole routing.

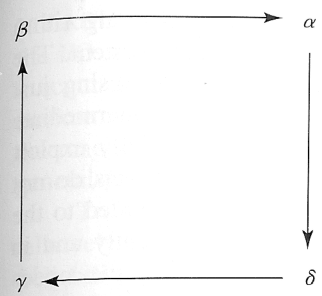


FIGURE 6.15
Blocking graph of deadlock in wormhole routing.

6.2.2 Network Architecture Issues[1]

HIGH-LEVEL ARCHITECTURAL ISSUES. At the highest level, a distributed system comprises a set of nodes communicating through an interconnection network. Each node may itself be a multiprocessor comprising application, system, and network processors; a shared memory segment; and I/O interfaces. Although the application processor may be an off-the-shelf product, the system and network processors usually have to be custom-designed because they provide the specialized support necessary for real-time applications. The memory subsystem may also be specially designed to provide fast and reliable communication between the processors at a node. For example, the memory subsystem may support a mailbox facility to support efficient interprocessor communication within a node of a distributed system.

The nodes of the system must be interconnected by a suitable network. In the past, for small systems, the network was a custom-designed broadcast bus with redundancy to meet the fault-tolerance requirements. More recently, however, architects have turned to either a high-speed token ring or a point-to-point network with a carefully-chosen topology. For example, the Spring system at the University of Massachusetts uses a high-speed optical interconnect called Scramnet, whereas the HARTS project at The University of Michigan uses a point-to-point interconnection network called the C-wrapped hexagonal mesh topology.

Irrespective of the exact topology, the network architecture should support scalability, ease of implementation, and reliability. It should also have support for efficient one-to-one, as well as one-to-many, communications. For instance,

[1] Section 6.2.2 is based on K. G. Shin and P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," *Proceedings of the IEEE*, Vol. 82, No. 1, 1994. © IEEE. Used with permission.

the C-wrapped hexagonal mesh topology used in HARTS has a $\theta(1)$ algorithm for computing all the shortest paths between any two nodes in the system. The information about all the shortest paths can also be easily encoded using just three integers and included as part of each message so that the intermediate nodes need not do much computation. The routing algorithm can fully exploit advanced switching techniques like wormhole routing, in which packets do not always have to be buffered at intermediate nodes before being forwarded to the next node in the route. Broadcasting can also be done fairly efficiently and in a fault-tolerant manner using the multiple disjoint paths between any two nodes in the system. Such capabilities are very important because reliable and timely exchange of information is crucial to the distributed execution of any real-time application.

LOW-LEVEL ARCHITECTURAL ISSUES. Low-level architectural issues involve packet processing, routing, and error/flow control. In a distributed real-time system, there are additional issues related to the support for meeting deadlines, time management, and housekeeping. Since the support of these low-level issues impedes the execution of application tasks, nodes in a distributed real-time system usually have a custom-designed processor for handling these chores. In the description below, this special processor is referred to as the network processor (NP).

The main function of the NP is to execute operations necessary to deliver a message from a source task to its intended recipient(s). In particular, when an application task wants to transmit a message, it provides the NP with information about the intended recipient(s) and the location of the message data, and then relies on the NP to ensure that the information reaches the recipients in a reliable and timely fashion.

The NP must establish connections between the source and destination nodes. It must also handle end-to-end error detection and message retransmission. As far as routing is concerned, the NP may select primary and alternative routes, allocate bandwidths necessary to guarantee timely delivery, packetize the information into data blocks and segments, and reassemble packets at the destination node. In point-to-point interconnections, the NP must support and choose an appropriate switching method such as wormhole routing, store-and-forward, or circuit switching. In token rings, the NP must select suitable protocol parameters to guarantee the deadlines of all messages. The NP must also perform framing, synchronization, and packet sequencing.

The NP must implement buffer management policies that maximize the utilization of buffer space, but guarantee the availability of buffers to the highest priority messages. Similarly, if noncritical messages hold other resources that are needed by more critical ones, NP must provide a means for preemption of such resources for use by the critical messages.

The NP may also have to monitor the state of the network in terms of the traffic load and link failures. The traffic load affects the ability of the NP to send real-time messages to other processors, while link failures affect system reliability. It may also keep track of the processing load of its host (or hosts), and use the information for load balancing/sharing and task migration operations.

I/O ARCHITECTURE. Little work has been done on network architectures for the I/O subsystem. Clearly, a real-time computer can process data no faster than it acquires it from sensors and operators. The nature of I/O devices in a real-time environment, consisting as it does of sensors and actuators, is quite different from the magnetic disks and tapes that are the I/O devices commonly encountered in general-purpose systems.

To improve I/O, multiple I/O devices need to be distributed and managed by relatively simple and reliable controllers. Moreover, to improve both accessibility (reliability) and performance, there must be multiple access paths (called *multi-accessibility* or *multi-ownership*) to these I/O devices.

One possible way to provide multi-accessibility, used in HARTS, is to cluster the I/O devices together and assign a controller to manage access to the devices in each cluster. The controller has a set of full-duplex links to certain nodes of the distributed system. In order to limit the number of links in each controller while providing multi-accessibility, a controller is connected to three nodes in the system as shown in the Fig. 6.16. Since each controller can be accessed by three nodes, different management protocols are proposed for handling the I/O requests. In a static scheme, one node is assigned the primary responsibility of managing the controller with the proviso that the other nodes can take over control if the primary node becomes faulty. In a dynamic scheme, all three nodes connected to a controller manage the controller using a more complicated protocol.

An alternative approach for connecting the I/O controllers to the nodes of the system is to connect an I/O controller to just one node. However, the placement of the I/O controllers must be done in such a way that a node is at most one hop away from a node with an I/O controller connected to it. To achieve fault-tolerance,

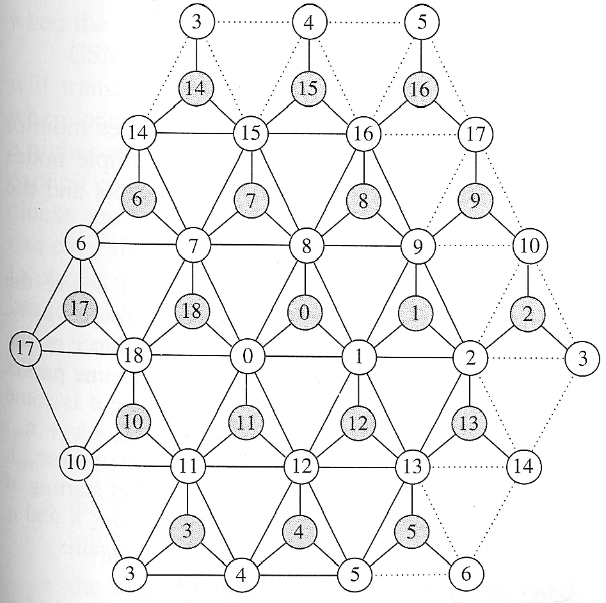


FIGURE 6.16
I/O controller placement.

TABLE 6.2
List of protocols

Protocol	Deadline guarantees?	Network
VTCSMA	No	Broadcast
Window	No	Broadcast
Timed token	Yes	Ring
IEEE 802.5	Yes	Ring
Stop-and-go	Yes	Point-to-point
Polled bus	No	Bus
Hierarchical round-robin	Yes	Point-to-point
Deadline-based	No	Point-to-point

schemes for the placement of I/O controllers have also been proposed in which a node is at most one hop away from j nodes with I/O controllers, where j is a design parameter.

Although the above solutions have made some headway in dealing with the I/O architectural issue, there is a lot of work that needs to be done in designing communication networks that can perform the levels of I/O needed for real-time systems.

6.3 PROTOCOLS

In the following sections, we describe protocols suitable for real-time systems. In Table 6.2, we provide a directory of these protocols. Some of these offer deadline guarantees; that is, they guarantee that messages will be delivered before their assigned deadlines. Others do not, but are "best-effort" algorithms, suitable only for soft real-time systems.

6.3.1 Contention-Based Protocols

These are distributed protocols that assume a broadcast medium. Nodes monitor the channel and transmit only when they detect that it is idle. If multiple nodes start transmitting at about the same time, there is a collision of packets and the transmissions have to be aborted and then retried.

Example 6.7. Consider the bus network shown in Figure 6.17 Let τ_{ij} denote the propagation delay between nodes i and j. Node m is transmitting from t_1 to t_2. Nodes n and q see the end of this transmission, which is followed by silence on the bus, at times $t_2 + \tau_{mn}$ and $t_2 + \tau_{mq}$, respectively. Suppose node n has some packet awaiting transmission. It starts transmitting at time $t_2 + \tau_{mn} + \epsilon$, where ϵ is some positive number. This transmission does not reach q until $t_q = t_2 + t_{mn} + \epsilon + \tau_{nq}$. Suppose a packet arrives at node q at time t_y, where $t_y \in (t_2 + \tau_{mn}, t_2 + \tau_{mn} + \epsilon + \tau_{nq})$. Since q has not heard the node-n tranmission yet, it transmits the packet starting at $t_y + \epsilon$. The two messages collide; and upon detecting the collision, nodes n and q cease transmission, and each back off for a random time before trying again.

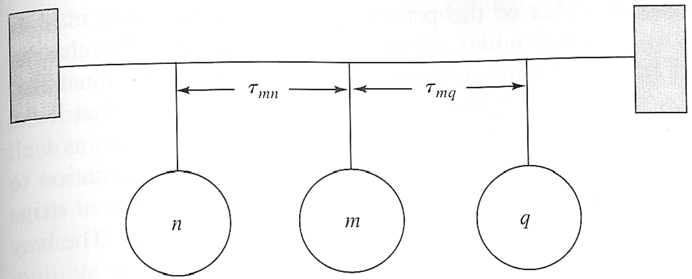


FIGURE 6.17
Bus network.

VIRTUAL-TIME CARRIER-SENSED MULTIPLE ACCESS (VTCSMA). The VTCSMA protocol has been designed for single-channel broadcast networks and for the bus and ring topologies.

In carrier-sensed multiple access (CSMA) protocols, all the nodes can monitor the communication channel. Suppose a node has something to transmit. If it observes that the channel is busy, it will refrain from interfering with the ongoing transmission. When it senses that the channel is free, the node will transmit its message. Since there is no coordination between the nodes, this can result in a collision resulting from multiple nodes attempting to transmit simultaneously. When a collision occurs, the transmitting nodes, which monitor the channel continually, abort their transmission and retransmit after waiting for a time. There are various types of CSMA, each with its own formula for computing the retransmission epoch. CSMA is an efficient communication scheme when the end-to-end transmission delay is much less than the average time to transmit a packet and when the load is not very high.

CSMA is a truly *distributed* algorithm, with each node deciding when it will transmit. It may seem a hopeless task to implement a priority algorithm, where one node with a lower-priority packet is supposed to defer to another with a higher-priority packet, using CSMA. This is not so, however. While there is no explicit coordination between the nodes, the nodes do see a consistent time if their clocks are synchronized, and they observe the same channel (slight differences can exist due to transmission delays). This common information can be exploited to obtain quite effective priority communications algorithms, including a priority based on deadlines.

Suppose that a node has a set of packets to transmit. How is it to determine when to transmit them on the channel? The information it has is

- the state of the channel,
- the priorities of the packets waiting in its transmission buffer to be transmitted over the network, and
- the time according to the synchronized clock.

The node does not have any idea of the priorities of any packets that may be awaiting transmission at the other nodes. Simply using the state of the channel and the priorities of its packets is not sufficient; the time information must also be used. The key to the VTCSMA algorithm that we present below is that if the priority of the packet can be computed as a function of the current time, as well as of some other parameters, it may be possible to use the time information to implicitly arrive at some global ordering of priorities. For example, we can strive to serve packets according to their deadlines, arrival times, and laxities. The way this is done will become apparent when we describe the algorithm.

The VTCSMA algorithm uses two clocks at each node. One is the real clock (RC), which tells the "real time," and is synchronized with the clocks at the other nodes. The second is the virtual clock (VC), which behaves as follows. When the channel is busy, the VC freezes; when the channel becomes free, the VC is reset (according to a formula that we will present) and then runs at the rate $\eta > 1$. That is, the VC runs faster than the RC when the channel is free, and not at all when it is busy.

Example 6.8. Figure 6.18 illustrates the operation of the VC for $\eta = 2$. The abscissa denotes the real time and the ordinate the virtual time. From real time 0 to $t1$, the channel is busy, and so the virtual time remains frozen at $t1'$. At real time $t1$, the channel becomes idle, and the VC is set equal to the RC and starts running at a rate of 2. At real time $t2$, the channel becomes busy again, and so the virtual time freezes at $t2'$. At real time $t3$, when the channel is idle, it is initialized and starts running again at rate 2. And so on.

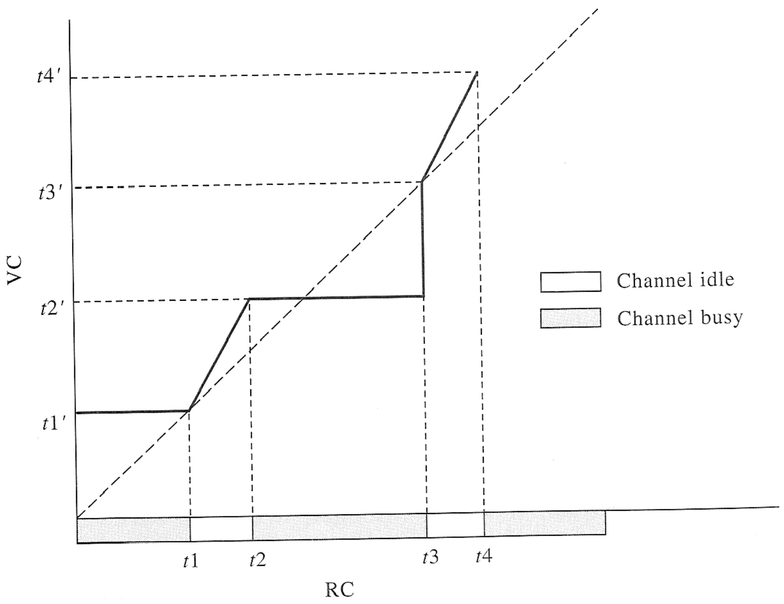


FIGURE 6.18
Operation of the virtual clock.

Since the real clocks are assumed to be synchronized and the virtual times are regularly reset with respect to the real times, the virtual times told at the various nodes are the same, plus or minus some small skew. This is the common information that is used to impose a global priority on the packets to be transmitted. Each node computes a virtual time to start transmission VSX(M), for every packet M awaiting transmission at that node. When the virtual time is greater than or equal to VSX(M), packet M becomes eligible for transmission. We leave to the reader the easy task of figuring out which packet is transmitted by a node that has multiple packets eligible for transmission at any one time. If a collision occurs, the VSX of that packet is modified suitably. The formula used for computing VSX depends on the type of VTCSMA that is used, and is described below.

Figure 6.19 is a flowchart that specifies the algorithm. How is the VC initialized when the channel becomes free, and the VSX(M) modified when there is a collision involving packet M? There are many ways of doing this, and each approach results in a different variation of VTCSMA. First, we need some notation.

τ	Propagation time from one end of the network to the other.
A_M	Arrival time of message (or packet) M.
T_M	Time required to transmit message M.
D_M	Deadline by which message M must be delivered to its destination.
L_M	Latest time by which the message must be sent to be able to meet the deadline. That is, $L_M = D_M - T_M - \tau$.
$\Lambda_M(t)$	Maximum amount of time that message M can be delayed at time t before missing its deadline. That is, $\Lambda_M(t) = D_M - T_M - \tau - t$.

We will list four variations of VTCSMA, with suffixes A, T, L, and D, respectively. When packet M arrives, we have:

$$\text{VSX}(M) = \begin{cases} A_M & \text{for VTCSMA-A} \\ T_M & \text{for VTCSMA-T} \\ L_M & \text{for VTCSMA-L} \\ D_M & \text{for VTCSMA-D} \end{cases} \tag{6.1}$$

When a collision occurs, each node involved in the collision either retransmits M immediately, with probability p, or (with probability $1 - p$), the node modifies VSX(M) to be a random number drawn from the following interval I:

$$I = \begin{cases} (\text{current VC}, L_M) & \text{for VTCSMA-A} \\ (0, T_M) & \text{for VTCSMA-T} \\ (\text{current RC}, L_M\ ) & \text{for VTCSMA-L} \\ (\text{current RC}, D_M) & \text{for VTCSMA-D} \end{cases} \tag{6.2}$$

When the channel switches state from busy to idle, the VC is initialized as follows:

$$VC = \begin{cases} \text{no change} & \text{for VTCSMA-A} \\ 0 & \text{for VTCSMA-T} \\ \text{RC} & \text{for VTCSMA-L and VTCSMA-D} \end{cases} \tag{6.3}$$