7

Resource Allocation for Local Computer Systems

Jie-Yong Juang
Benjamin W. Wah

7.1. Introduction

Because of the rapid advances in microelectronics and Very-Large-Scale-Integrated (VLSI) circuit design technologies, the cost of computer hardware has dropped drastically and the processing and communication speeds have approached some physical limitations. These technological advances, coupled with the explosion in size and complexity of new applications, have led to the development of *resource sharing computer systems*. Such systems usually consist of a large number of general- and special-purpose processors interconnected together by a communication network called the *resource-sharing interconnection network* (64).

A resource in a computer network is a processor that performs computation functions or manipulates data objects. It may generate requests to

Research supported partially by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW, and by National Science Foundation Grant DMC-85 19649.

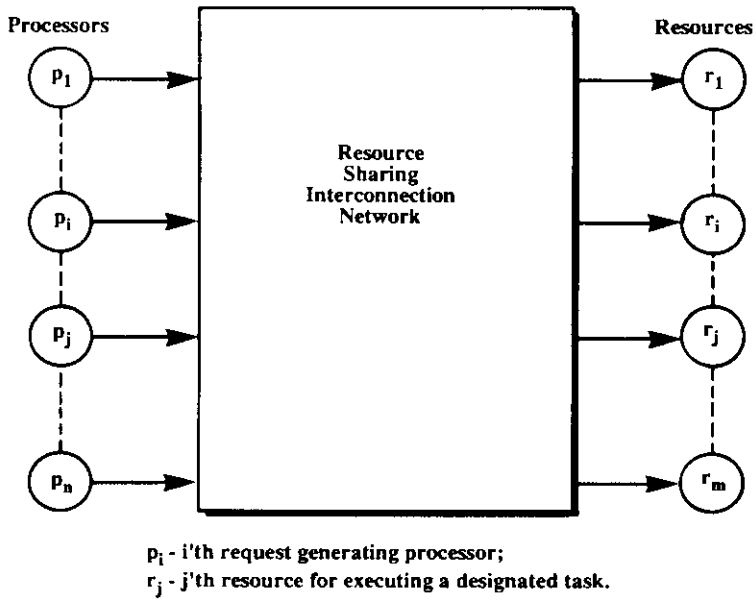


Figure 7.1 A generic model of resource sharing computer systems (Arrow K, Pesotchinsky L, Sobel M: On partitioning a sample with binary-type quetions in lieu of collecting observations. *Journal of the American Statistical Association*, Vol. 76, No. 324, June 1981, pp. 402–409.

lowing properties:

1. The global status information of the system is not available to the individual processors.
2. The interconnection network is the only intercommunication facility among processors.
3. A request may be dispatched to any one in the set of available resources that is capable of carrying out the designated task.
4. A resource is accessible by any request-generating processors.

A resource-sharing system with these characteristics has the following advantages:

1. Tasks may be executed in parallel, and workload of processors may be distributed evenly.
2. Efficient architectures for performing special tasks may be included in the system.
3. Modifications to include new functions or increased performance can be done easily because of the system's modularity.
4. Malfunctional devices may be removed from the system without stopping the entire system.

Consequently, the performance of such a system may be improved by increasing the number of resources or replacing an existing resource by a more efficient one. The system is also highly reliable and maintainable.

A shared resource may manipulate data objects or provide computational service on request. Issues on sharing data have been studied intensively in recent years. Many schemes have been proposed to deal with the synchronization and data-coherence problems. Examples include monitors and synchronization schemes in operating systems (13, 23), cache coherence schemes in multiprocessors (14), and the methods of maintaining data integrity in distributed database managements systems (51). However, as discussed in Section 7.2, schemes for sharing computational devices are less developed. Most existing schemes are based on centralized control or simple distributed extensions of centralized control. The characteristics of the network are usually not incorporated in the design of resource-sharing schemes.

This generic model encompasses many existing or proposed systems. The dynamic task migration is the basic feature of many proposed distributed programming languages such as Hoare's Cooperating Sequential Processes (24, 53), CLU developed at MIT (39, 40), and Brinch Hansen's distributed processes (21). New operating system designs also provide mechanisms to support dynamic task migrations. Examples include pipes in MEDUSA (47) and UNIX (50). Many architectures also exhibit the characteristics of this generic model. Examples include local computer networks with load balancing such as the ECN (26) and LOCUS (69), VLSI-systolic array multiprocessors (5, 35), and dataflow supercomputers (11). Because resources of these architectures represent different levels of abstraction, it is instructive to describe them and to indicate their mappings to the generic model.

Example 1: Local Computer Network with Load Balancing. Load balancing is a scheme that engages communication facilities in supporting remote job execution in a user-transparent manner, so the turnaround time is reduced through the enhancement of resource sharing. Depending on the workload of processors, the network operating system may distribute jobs to a remote processor or may schedule them for local execution. A local computer network with load balancing is illustrated in Figure 7.2 (26, 63). Corresponding to the model, those processors of heavy workloads are request-generators, and those processors of light workloads are resources. The resources in this system are job-level processors.

Example 2: VLSI-Systolic Array Multiprocessors. A VLSI systolic array is a parallel pipeline architecture for evaluating a recursive function, such as FIR filtering, matrix multiplication, and FFT. Such VLSI chips are usually organized as attached processors to host computers as shown in Figure 7.3 (35). In this organization, requests are generated from processors and routed to systolic arrays through the system bus. The resources in such systems are process-level special-purpose processors.

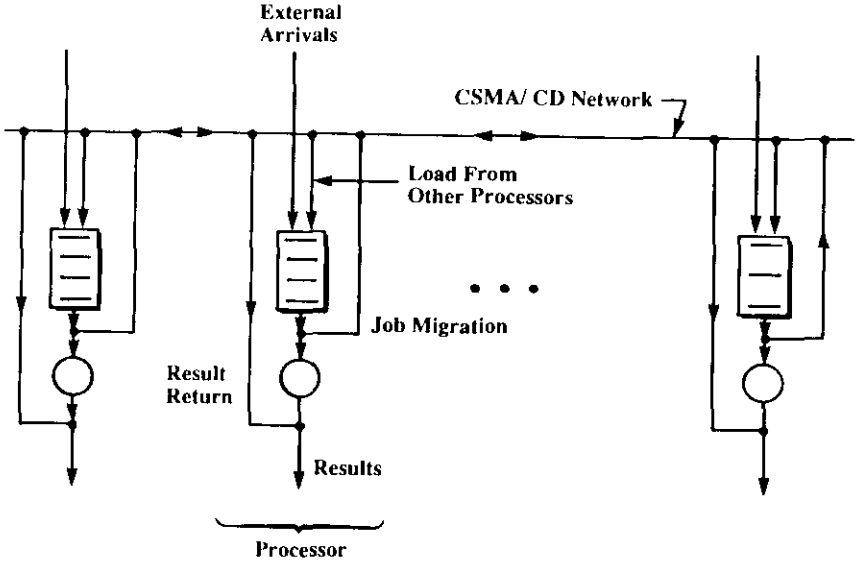


Figure 7.2 A queuing representation of local computer network with load balancing.

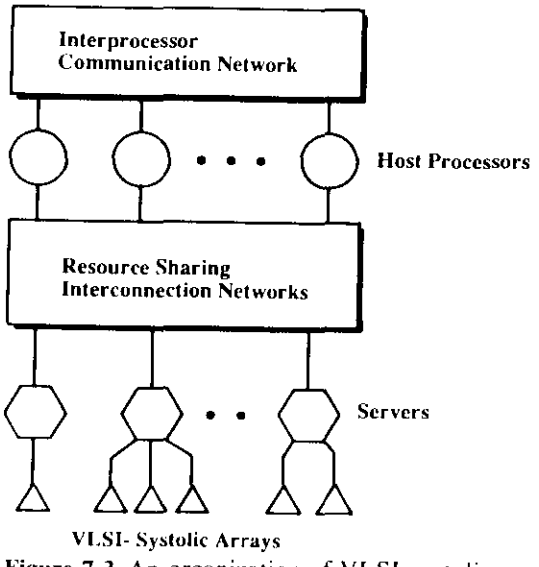


Figure 7.3 An organization of VLSI systolic-array multiprocessors.

Example 3: Dataflow Supercomputers. In contrast to the conventional von Neumann machine, there are no sequence-control mechanisms in a dataflow machine. The execution of an instruction is driven by the availability of its input data. An instruction is active when all its input arguments are ready. An active instruction is executed at a processing unit. The outputs of this instruction will activate other instructions for the subsequent executions. A typical dataflow multiprocessor is shown in Figure 7.4 (11). For a detailed discussion, see Chapter 9.

In this architecture, instructions are allocated in the activity store and waiting for their inputs. Once an instruction becomes active, it is routed through an arbitration network to a processing element and executed there. The output is then routed back to the activity store through a distribution network. The activity store is divided into cell blocks, and active instructions in a cell block are requests. The processing units are arithmetic and logical devices, hence they are instruction-level resources.

7.1.2. Resource Scheduling

Resource scheduling entails the allocation of resources (including communication facilities), so task migrations can be carried out efficiently. In general, the migration of a task in a distributed resource-sharing system is divided into three phases: resource-bidding, task-migration, and result-return. In the first phase, the local processor has to make a request for utilizing a

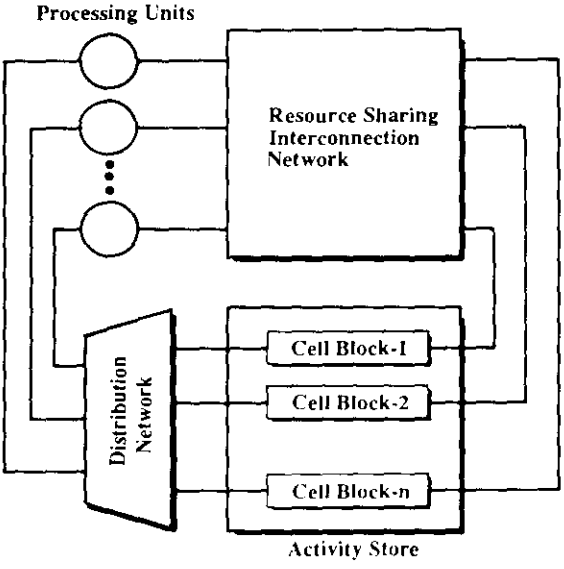


Figure 7.4 A dataflow multiprocessor.

resource. In the second phase, the body of the task, including the task control, program code, and data, are transferred to the resource allocated and executed remotely. In the last phase, the results generated from the execution of the task are routed back to the original processor. Basically, only data transmission is involved during the migration and result-return phases. Resource scheduling is carried out in the resource-bidding phase. In this phase, system status information has to be collected, and the decision of resource allocation has to be made.

Issues of Resource Scheduling

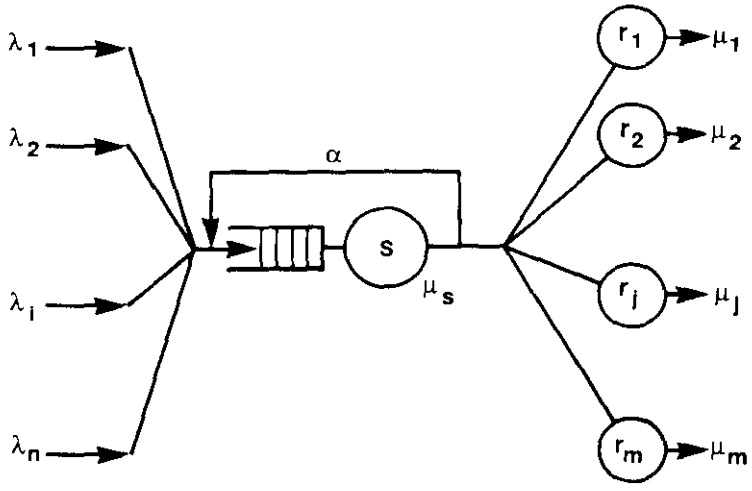
The central issue of resource scheduling is to determine a resource mapping that maps requests to resources. Resources will be allocated to associated requests determined by the mapping. Because a task can be allocated to one of a set of free resources, and multiple requests may contend for the same resource, multiple requests may be allocated the same resource while other resources are idle. This problem is called *resource conflict*. If the resource has local buffers, the tasks may still be migrated and queued at the resource regardless of conflicts. This causes no error in operation but may deteriorate resource utilization because of the imbalance of workload. If the resource has no buffering capability, every request except one has to be rescheduled again.

In addition to resource conflict, a bad resource allocation may degrade the performance of the network. Depending on the characteristics of the system, a physical communication link may be assigned to every processor-resource allocation and operate in a circuit-switching mode, or links may be shared in a packet-switching mode. A resource-allocation scheme that minimizes resource conflicts is not necessarily optimal because there may be many paths being blocked in the circuit-switching mode, and packets may be congested in the packet-switching mode.

In summary, three major issues will affect resource utilization in resource-sharing computer systems. These include network blocking (or packet congestions), request conflicts, and imbalanced workload. Unless the scheduling algorithm is carefully designed and implemented, there may be many adverse effects on the benefits of resource sharing.

Efficiency of Resource Scheduling

To illustrate the effects of these issues, a resource-sharing system may be transformed into a queuing network as shown in Figure 7.5. In this queuing model, a processor is represented by an arrival process with arrival rate λ_i, while a resource is represented by a server R_j with service rate μ_j. An additional server S is introduced to model the resource-allocation mechanism and the communication network. A branch from the output of server S feeds back to the input of this server and represents the unsuccessful resource allocations due to network blockages or resource conflicts. Although the



S – A server represents the resource allocation mechanism;
α – the probability of network blocking and resource conflicts;
λ_i – request generation rate of processor p_i ;
μ_j – Service rate of resource r_j;
μ_s – Scheduling rate of S.

Figure 7.5 A queuing model for resource sharing computer systems.

scheduling mechanism is represented by a single server, it does not imply centralized control. Instead, it may be realized in many alternative ways as will be described in Section 7.2. Task-transmission delays in the network are considered part of scheduling overhead.

In a word, the service rate of *S* depends on two factors: the speed of the scheduler and the delay of task transmissions. According to the results of queuing theory (32), the service rate of server *S* is crucial to the overall system performance. Thus, improving the efficiency of resource allocation is translated to increasing the service rate of *S* and reducing its feedback probability. These may be achieved (1) by a good design of a high-speed resource allocation mechanism, (2) by utilizing a good scheduling algorithm that generates a good resource mapping, and (3) by using a high-speed communication network.

Scheduling Disciplines

Depending on the scheduling disciplines, requests and resources may be characterized by multiple attributes. A request may be represented by the types of task it requests, expected execution time, and priority level. On the other hand, resources may be modeled by its speed, load, and reliability.

A scheduling algorithm has to evaluate the allocation costs based on these attributes.

Consider a scheduling example that consists of two types of requests: matrix computations and scalar computations. Suppose that requests for matrix computations have higher priority than requests for scalar computations, that all vector processors are busy, and that only pipelined processors are available. Then requests for matrix computations would be allocated if the requests are scheduled according to their priorities. On the other hand, matrix computations may be executed more efficiently on a vector processor than that of a pipelined processor. Hence, requests for matrix computations may not be allocated if the requests are scheduled according to the preferences of resources.

In this chapter, we consider only the class of scheduling disciplines in which multiple attributes are combined into a single parameter. The parameter that characterizes a request is called the *priority* of the request, and the parameter that characterizes a resource is called the *preference* of the resource. Our objective is to investigate the design of efficient resource-scheduling mechanisms for resource-sharing computer systems, and explore the integration of the scheduling algorithms and computer networks. Several goals are to be pursued:

1. A feasible scheduling strategy for improving resource utilization in resource-sharing computer systems will be studied.
2. The distribution of scheduling intelligence will be investigated.
3. Fast implementation for the scheduling mechanisms will be developed.

A unified design methodology is employed to incorporate these three design goals. However, we consider only those scheduling schemes that allocate one resource to a request at a time. When multiple resources are requested by a single request, they have to be allocated sequentially.

7.2. A Taxonomy of Resource-Allocation Schemes

In the design of resource allocation schemes, achieving high-speed scheduling and obtaining an optimal mapping are usually two mutually conflicting goals. Compromises between the optimality of the scheduling decision and the overhead of collecting system status information are reached in many ways. Resource allocation schemes can be characterized by the trade-off between these two goals. In this section, a taxonomy of these resource-allocation schemes is presented. The advantages and disadvantages of each class of resource-allocation schemes in the taxonomy are explored. This leads to the conclusion that a distributed state-dependent allocation scheme is preferable. To tackle the complex design problems of distributed state-

dependent resource allocation schemes, a systematic design methodology is proposed in this section.

7.2.1. A Taxonomy of Resource-Allocation Schemes

A taxonomy that categorizes most resource-allocation schemes is given in Figure 7.6. Resource-allocation schemes may be classified into two classes depending on whether global status information is used or not, and whether they are *state-dependent* or *state-independent*.

In the class of state-independent scheduling schemes, resource allocation is carried out by the individual request-generator. Each request-generator determines the resource to bid for based on the local information available. This information may include the statistics of the system's operating history, piggy-backed information carried by the return message of previous requests, and the specifications of individual resources. If the processor chooses a resource randomly, the scheduling scheme is a *random scheduling* scheme. On the other hand, if the statistics on previous requests and resource specifications are inputs to the scheduling decision, the scheduling scheme is a *probabilistic scheduling* scheme. Because requesting processors do not communicate, resource conflicts are unavoidable. Conflicting requests have to contend for the resource they bid for. Queuing-network analysis has been applied intensively to analyze the efficiency of this class of scheduling schemes (45, 59, 70).

On the other hand, in the class of state-dependent scheduling schemes, the global status information is crucial. Among them, a *localized state-de-*

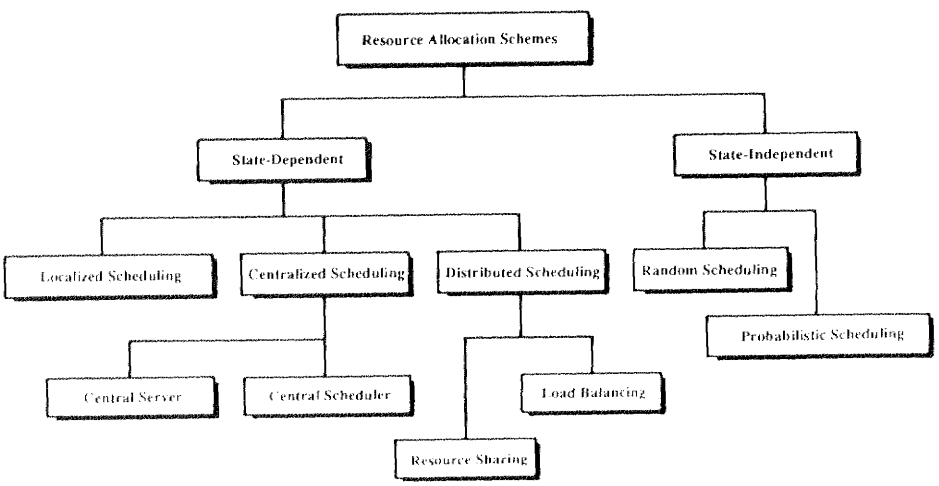


Figure 7.6 A taxonomy of resource allocation schemes.

pendent scheduling scheme requires every processor to maintain a copy of the global state information and to determine the resource allocation independently. In contrast, in *centralized state-dependent scheduling* schemes, the status information is collected by a central control node that determines the resource mapping to be distributed to all requesting processors. A scheduling scheme with this kind of organization is called a *central scheduler*. Essentially, only the tasks of resource bidding are carried out in the central control node. Task migrations and result returns are carried out independently. However, the central control node may be also responsible for buffering tasks and dispatching them to resources. In this sense, the scheduler becomes a *central server*. A central server is usually found in systems in which resources do not have local memories. Some master–slave multiprocessors belong to this class (15).

A *distributed scheduling* scheme differs from localized and centralized scheduling schemes in the way that global status information is collected and utilized. In a distributed scheduling scheme, only partial status information is maintained by each processor, and the scheduling decision is made cooperatively through exchanging information. The amount of information flow is usually lower than that of the previous two approaches.

Most existing resource scheduling schemes belong to the class of state-independent schemes (10, 27, 37, 38): The resource sharing protocol of ARPANET is a typical example in which task migrations are determined by end users (57). This class of resource scheduling schemes is simple and incurs relatively little overhead. Nevertheless, the problem of low resource utilization remains unsolved. Centralized state-dependent scheduling schemes can be found in many multiprocessor systems with master–slave structure (2, 15, 22). However, adopting this approach to distributed systems tends to eliminate their advantages. Localized state-dependent scheduling schemes are the direct distributed extensions of centralized control. The load balancing schemes of ECN and LOCUS belong to this class. Although this approach can be implemented in an existing network, it incurs a large amount of redundant information flow and is hard to maintain a consistent state information because of the network delay. Consequently, a resource mapping generated by an optimal scheduling algorithm is not necessarily the optimal one because inaccurate information may be used.

Distributed state-dependent scheduling schemes are generally preferable for the following reasons: (1) the information flow in maintaining the global information is reduced because status information is utilized efficiently; (2) they can achieve optimal resource allocation; and (3) their speed may be increased because of the concurrent execution of scheduling tasks. Only a few simple distributed state-dependent schemes have been proposed (29, 30, 41, 62–64). It is still an open area of study. We focus on the design of distributed state-dependent resource scheduling schemes in this chapter.

7.2.2. Implementation Considerations and a Design Methodology

In general, a resource-scheduling algorithm generates a resource mapping according to the system status information. A good resource mapping is one that minimizes a cost function under the network constraints. The cost function is usually determined by the scheduling disciplines. It is usually easier to optimize the cost function regardless of the constraints imposed by the network. As a result, many processors may not be allocated a scheduled resource because of conflicts in the network. To reduce this probability, a high-bandwidth network is usually used (16). A crossbar network has been used in systems such as the C.mmp (15, 18). It does not have the network blocking problem. However, the cost of a crossbar network is $O(n^2)$, where n is the number of devices connected, and is not practical when the system is large. A multistage interconnection network is a cost-effective choice (64), but blocking probability may be as high as 60% if resources are not allocated properly (17, 48). These observations indicate that a good resource-scheduling algorithm should incorporate network constraints in the optimization of a given scheduling discipline. The following design methodology is proposed:

1. Formulate the resource-scheduling problem into a constrainted optimization problem.
2. Design a distributed algorithm to solve the problem.
3. Identify primitive operations of each process in the distributed algorithm.
4. Integrate the primitive operations into the network.

The cost of collecting status information may also be included in the objective function, so trade-offs can be made between the amount of status information used and the efficiency of the scheduling algorithm. A well-designed distributed algorithm should reduce unnecessary message passing. The crucial speedup of the scheduling scheme lies in implementing the primitive operation into the network. This approach essentially shifts the responsibility of scheduling requests by the request generators to the network.

We will show the application of the methodology to the design of resource scheduling schemes for a single contention-bus network. Resource allocation is studied with respect to requests that need one resource only; multiple resources needed by a request are allocated sequentially. The network is assumed to be a *reliable multiaccess bus* with the *broadcast capability*. *Carrier-sense-multiaccess networks* with *collision detection* (CSMA/CD networks) belong to this class, and are exemplified by the Ethernet (56) (Figure 7.7a).

CSMA/CD networks evolved from CSMA networks, which have listen-before-talk protocols to avoid overlapping transmissions. The collision-detection capability of CSMA/CD networks allows processors to additionally listen-while-talk, so collisions resulting from simultaneous transmissions can

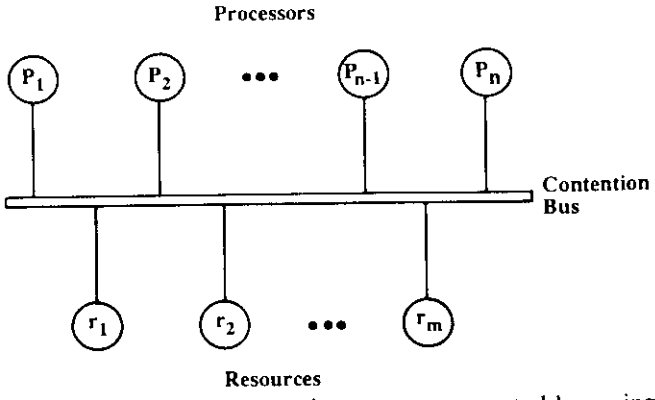


Figure 7.7a A resource-sharing system connected by a single multiaccess bus.

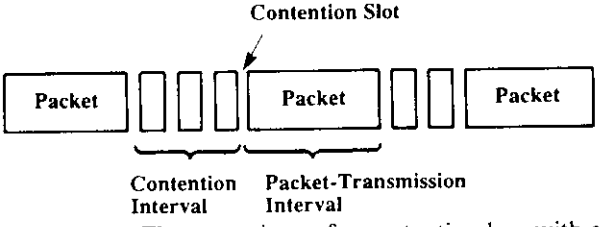


Figure 7.7b The operations of a contention bus with altering phases.

be detected and stopped immediately. The time for a processor to assert that there are no overlapping transmissions is the end-to-end propagation delay on the bus and is called a *contention slot*. To avoid repeated collisions, a contention-resolution protocol is used to control transmissions and to eventually isolate one station for transmitting the message. The operation of the bus is thus divided into two alternating phases, the contention-resolution phase consisting of a sequence of contention slots, and the data-transmission phase consisting of the message transmission (Figure 7.17b). Many contention-resolution protocols have been proposed and implemented (4, 7, 8, 19, 25, 28, 33, 34, 36, 42, 44, 60, 63). They are distinguished by the different transmission control.

7.3. Optimal Resource-Allocation Algorithms

The optimal resource-allocation problem can be considered as an optimization problem that optimizes the system performance or cost subject to constraints of the network. Let P be the set of request generators and R be the set of resources. Each request generator $p \in P$ is characterized by a

priority x_p, which measures the urgency that the request generated has to be serviced. Similarly, each resource $r \in R$ is characterized by a preference y_r, which measures its capability to service a generated request. Because there is only one communication channel in a single-bus system, only one resource can be allocated at a time, and the scheduling problem is reduced to finding a pair of request generator and resource that optimize the system performance or cost. The optimization can be represented as

$$\min_{(p,r)\in P\times R} H(x_p, y_r) \tag{7.1}$$

where H is a cost function defined with respect to a given scheduling discipline.

In general, the cost function H depends on the characteristics of tasks and resources, as well as the interconnection network. It may be very complex and difficult to optimize. We will only study a special class of the cost functions that are monotonic with respect to x_p and y_r. That is,

$$\frac{\partial}{\partial x_p} H(x_p, y_r) \text{ is either positive or negative for all } x_p \text{ and } y_r \tag{7.2a}$$

$$\frac{\partial}{\partial y_r} H(x_p, y_r) \text{ is either positive or negative for all } x_p \text{ and } y_r \tag{7.2b}$$

These conditions imply that, for a given resource, the cost is minimized by servicing a task of the highest priority (if Eq. (7.2a) is negative), or one with the lowest priority (if Eq. (7.2a) is positive). Similarly, for a given request, the cost is minimized by choosing a resource of the highest preference (if Eq. (7.2b) is negative), or one with the lowest preference (if Eq. (2b) is positive). For instance, if

$$\frac{\partial}{\partial x_p} H(x_p, y_r) \leqslant 0 \text{ and } \frac{\partial}{\partial y_r} H(x_p, y_r) \geqslant 0$$

it follows directly from Eqs. (7.1) and (7.2) that

$$\min_{(p,r)\in P\times R} H(x_p, y_r) = H(\max_{p\in P}(x_p), \min_{r\in R}(y_r)) \tag{7.3}$$

Optimal resource scheduling can thus be considered as choosing a request generator p with the maximum x_p and a resource r with the minimum y_r independently.

Many existing resource-scheduling problems can be solved by independently selecting the task to be serviced and the resource to service the task. Some notable examples are given here.

1. *Random-Access Protocols in CSMA Networks.* In CSMA networks, all processors share a single communication channel to communicate with each other. Processors with message to transmit are request generators, and the communication channel is the only shared resource. Contention-reso-

lution protocols in CSMA networks are designed to resolve contentions in using the channel. Because each request generator has equal right to access the channel, its priority can be considered as a random number in (0, 1], and the cost function $H(x_p, y_r) = x_p$. The request generator with the minimum number generated is given the access right to the channel.

2. *First-Come-First-Served Discipline in CSMA Networks.* The channel is the only resource to be scheduled. The priority level x_p is an increasing function of the task arrival time. The cost function $H(x_p, y_r) = x_p$.

3. *Shortest-Job-First Discipline in CSMA Networks.* The channel is the only resource to be scheduled. The priority level x_p is an increasing function of size of the job. The cost function $H(x_p, y_r) = x_p$, and the scheduler selects the smallest job.

4. *Priority Scheduling.* Messages in the network are divided into priority classes (levels), and the channel is allocated to service messages in decreasing order of priority levels. Several CSMA protocols for handling priority messages have been suggested recently (20, 46, 55, 58). They may be classified as linear protocols and logarithmic protocols. Each station is assigned the highest priority of the local messages. In a linear protocol, a slot is reserved for each priority level during the resolution of priorities. An active station contends during the slot reserved for the local priority level. When the station(s) with the highest priority level is determined, the process is switched to identifying a unique station within this priority level. This scheme is good when high-priority messages are predominantly sent. A logarithmic protocol determines the highest priority level in $O(\log_2 P)$ steps by a binary-divide scheme, where P is the maximum number of priority levels (46). This assumes that the highest priority level is equally likely to be any one of the P priority levels. Neither of these schemes is able to adapt to the various traffic patterns.

Resource scheduling in this case can be carried out in two phases. The first phase determines the highest priority level present in the network. A cost function $H(x_p, y_r) = -x_p$ is assumed. There may be multiple stations in this priority level, and scheduling for these stations is done in the second phase using one of the preceding criteria.

5. *Resource Sharing of a Pool of Identical Resources.* The priority of a request generator is an integer between 1 and P. The preference of a resource can be a random number in [0, 1] indicating its status (0 indicates that it is busy; any number between 0 and 1 indicates that it is free). Resource scheduling is carried out in two phases. The first phase identifies a request generator with the highest priority. The second phase identifies a free resource to service the task. Examples of cost function $H(x_p, y_r)$ that can be used are $(x_p - y_r)$ or $(-x_p y_r)$.

6. *Load Balancing.* This uses the communication facility to support remote job execution in a user-transparent fashion to improve resource utilization and to minimize response time. A decision to load balance a job is

made if the job is likely to be finished sooner when executed remotely than when executed locally. Resource scheduling is performed in two phases. In the first phase, processors are treated as request generators and are assigned priority equal to the average response time of executing a job locally. The processor with the highest response time is chosen as the request generator to send the job. In the second phase, processors are treated as resources and are assigned preferences equal to the sum of the average transmission time of sending a job across the network and the average response time of executing a job locally. The processor with the lowest preference is chosen. The cost function $H(x_p, y_r) = -x_p + y_r$ is the reduction in response time of executing a job remotely at processor r.

In these examples, only linear functions on x_p and y_r are defined. In general, they can be any function satisfying Eqs. (7.2a) and (7.2b).

A general organization of a resource scheduler is shown in Figure 7.8. There may be multiple classes of problems in resource sharing and they will be assigned different priorities in scheduling. For example, the network may be designed primarily for message transfers, and load balancing may be its secondary function. The resource scheduler will schedule all message transmissions before initiating load balancing for the system. For this example,

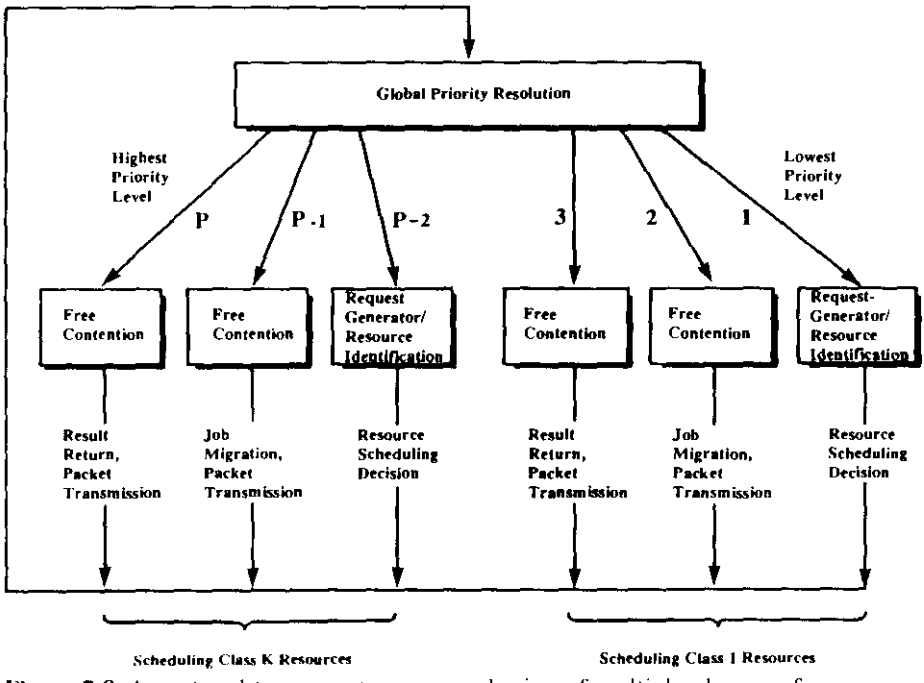


Figure 7.8 A protocol to support resource sharing of multiple classes of resources connected by a multiaccess bus.

$P = 6$ and $K = 2$ in Figure 7.8. Class 1 tasks refer to load-balancing operations, and Class 2 tasks refer to message transmissions. Generally, within a class of resource-scheduling problem, the return of results from a previously migrated job, if any, is given a higher priority than the migration of a new request because any delay in returning results contributes to an increase in response time, while an earlier transmission of a request to a remote resource may not reduce the response time unless the remote resource is idle. The migration of a request is given a higher priority than the identification of a new request-generator/resource pair because a request must be completely sent before it can be processed, and any delay in completing a job transfer may tie up valuable buffer space unnecessarily and reduce the resource utilization.

7.4. A Distributed Minimum-Search Algorithm

The operations of the resource scheduler in Figure 7.8 can be reduced to the primitive operation of identifying the extremum from a set of physically dispersed random numbers called *contention parameters*. The generation of these parameters may be dependent on each other, and may also be site-dependent. For tractability reasons, the parameters are assumed to be independently generated and possibly site-dependent in this section. Specifically, the identification of the task with the highest priority in Figure 7.8 can be considered as the search of the maximum priority level from a set of priority levels, one from each processor. Similarly, the transmission of a message (result or job) can be regarded as the selection of a ready station from a set of ready stations, each of which generates a contention parameter from a uniform distribution between 0 and 1. Likewise, the identification of a request generator or resource is again the selection of the station with the maximum or minimum parameter.

Conventionally, the implementation of an extremum-search algorithm relies on the message-passing mechanism to collect all information to a central site. This requires $O(n)$ messages, where n is the number of stations. In this section, an efficient distributed protocol for identifying the minimum is presented. The algorithm for searching the maximum is similar. The proposed algorithm has a load-independent behavior, which is important for resource-sharing applications because the number of processors to participate in identifying the extremum is usually large. Conventional contention-resolution algorithms, such as Ethernet's Binary Exponential Backoff algorithm, is load-dependent, but performs satisfactorily because the channel load is normally low for point-to-point message transmissions. Moreover, these algorithms cannot be directly applied to identify the extremum.

It is assumed that each processor in the network is capable of maintaining a global reference interval or *window*, and counting whether there

```
procedure window_protocol_station_i:
/* procedure to find window boundaries for isolating one of the contending stations */
[ /* window - function to calculate window size w,
   random - function to generate local contention parameter,
   estimate - function to estimate channel load,
   transmit_signal - function to send signal to bus with
      other stations synchronously,
   detect - function to detect whether there is collision on the bus (three-state),
   r_i - local contention parameter,
   n̂ - estimated channel load,
   lb_minimum - lower bound of interval containing minimum (minimum is L),
   ub_minimum - upper bound of interval containing minimum (maximum is U),
   contending - boolean to continue the contention process,
   state - state of collision detect, can be collision, idle, or success
       (for three-state collision detection). */
   lb_minimum := L;
   ub_minimum := U;
   r_i := random (L,U);
   n̂ := estimate ();
   w := window (lb_minimum, ub_minimum, n̂);
   contending := true;
   while (contending) do [
       if (r_i > lb_minimum and r_i < w) then [
             /* parameter is inside window, contend for bus */
             transmit_signal();
             /* test for unique station in the window */
             state := detect ();
             if state = collision then
                 /* update upper bound of interval containing minimum */
                 ub_minimum := w;
              else /* successful isolation of minimum */
                  return (lb_minimum, ub_minimum);
             w := window (lb_minimum, ub_minimum, n̂) ]
       else [
             state := detect();
       if state = idle then
                  /* all parameters are outside window */
                  /* update lower bound of interval containing minimum */
                  lb_minimum := w;
   ]                 w: = window (lb_minimum, ub_minimum, n̂)
             else
                  /* some other parameters are inside window, stop contending */
                  contending := false ]
   return (failure)
]
```

Figure 7.9 Procedure illustrating the basic steps executed in each station for contending the channel with a three-state collision-detection mechanism.

is none, one, or more than one contention parameter falling in the window. A global window can be maintained in all stations if they start in the same initial state, receive identical information from the bus, and execute the same control algorithm in updating the window with information received from the bus. Suppose that the set of contention parameters is $\{x_1, \ldots, x_n\}$ in the interval between L and U, and that y_i is the i-th smallest of the x_js. To search for the minimum, an initial window is chosen with the lower bound at L and the upper bound between L and U. There can be zero, one, or more than one contention parameter in this window. If there is exactly one contention parameter in the window, it can be verified as the minimum, y_1. Otherwise, the window has to be updated: it is moved if it is empty, or shrunk to a smaller size if it contains more than one number. This process is repeated until the minimum is uniquely isolated in the window. An implementation of the distributed window-search scheme at Station i, $1 \leq i \leq n$, on a multiaccess bus with a three-state collision-detection mechanism is shown in Figure 7.9.

Figure 7.10 illustrates the steps involved in the window-search scheme. Initially, five stations are ready, and they sense that the bus is free. Each

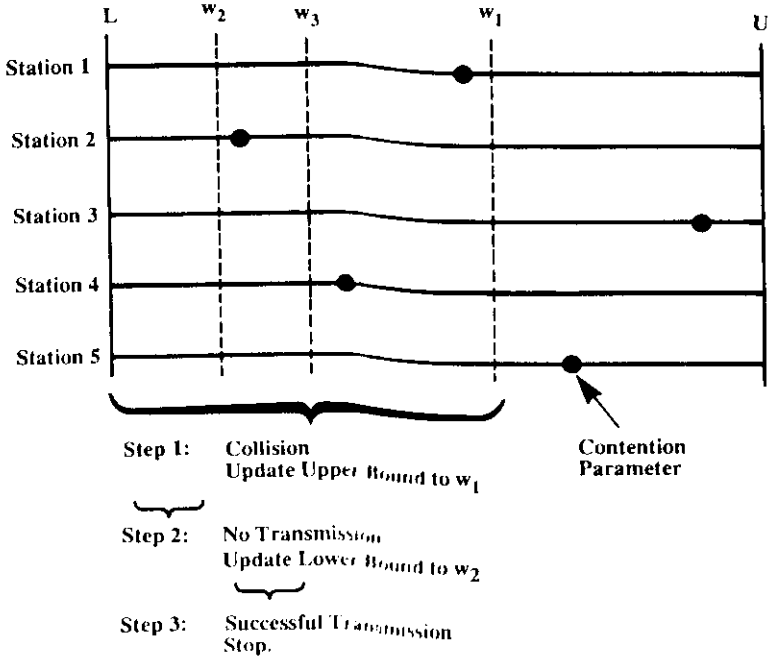


Figure 7.10 An example illustrating the updates of the global window to isolate the station with the minimum contention parameter. (Braces indicate windows used in different steps.)

of them generates a random contention parameter in $(L, U]$, sets the window as $(L, w_1]$, and transmits in the next contention slot if its contention parameter falls in the window. Station 3 is eliminated in the first iteration. As Stations 1, 2, 4, and 5 transmit, collision is detected. The stations reduce the upper bound of the interval to w_1 and set the windows to $(L, w_2]$ (identical for all stations as they use identical window-control algorithms and inputs). In the second iteration, no transmission is detected because all contention parameters are outside the window. The lower bound of the interval is set at w_2, and all stations set the windows as $(w_2, w_3]$. In the third iteration, successful transmission is detected, and the process terminates.

The concept of window protocols has been proposed with respect to contention resolution on multiaccess networks, but has not been developed for resource sharing and extremum search in general. Moreover, efficient and practical window-control algorithms have not been found. The optimization of the window size in the Urn Protocol (34) was studied by Hluchyj (25), who formulated it into a Markov decision process with an exponentially large number of states. Mosley and Humblet proposed to use the generation times of messages as a basis for the transmission order on the bus (44). This protocol is a generalization of Gallagher's procedure (19), which is itself based on an idea from Hayes (22) and Capetanakis (6). Towsley and Venkatesh (60) and Kurose and Schwartz (36) further extended Mosley and Humblet's algorithm by developing new heuristics. Mosley and Humblet also proposed that stations can generate random numbers as the contention parameters (44). The throughput was analyzed according to an infinite-population assumption.

The basic operations required for the proposed window-search scheme can be implemented easily either in hardware or in software on an existing multiaccess network such as the Ethernet. The global window can be maintained by updating an initially identical window with a common algorithm and using identical information broadcast on the bus. Assuming that information broadcast is received correctly by all stations, the global window will be synchronized at all sites.

To count the number of contention parameters falling in the window, the collision-detection capability of the network interface can be used effectively to detect whether the previous contention slot was empty, successful, or had collision. Stations with parameters inside the window contend for the bus in a contention slot. If there is more than one station with a parameter in the window, a collision will be detected. If there is exactly one station with a parameter in the window, a successful transmission will be detected. If there is no station with a parameter in the window, an empty slot will be detected. Each iteration of the protocol in Figure 7.9 will be completed in one contention slot. Hardware implementation will be discussed in Section 7.4.7 after the window-control algorithms are presented.

In systems where modification to existing hardware is impossible, the window protocol can be implemented in software. Software implementation

is only necessary for applications that select a station based on the meaning of the contention parameter (such as identifying the station with the maximum response time). For applications that need to randomly select a station, such as identifying a free resource, the existing interface suffices. Suppose that an existing Ethernet for point-to-point and broadcast transmissions is available. Stations with parameters inside the window contend for the bus. The station that is granted the bus will broadcast its parameter. However, in this case, it is not clear whether exactly one station or more than one station have parameters inside the window. (This is equivalent to a network with a two-state collision-detection capability.) Hence, a verification phase must follow to assert that the broadcast parameter is the minimum. This verification phase can be implemented as a timeout period, so other stations with smaller parameters can continue to contend and broadcast a smaller parameter inside this timeout period. In each iteration of the window protocol, the channel has to be contended twice, and two broadcasts of contention parameters have to be made. By suitably adjusting the timeout period according to the channel load, the station with the minimum parameter can be isolated with a high degree of certainty and without significant degradation in performance. The window will be adjusted according to the minimum of the two broadcast contention parameters.

7.5. Window-Control Algorithms

To minimize the number of iterations in the protocol, to identify the minimum, the window used in each step must be chosen appropriately. Given the lower and upper bounds of the interval containing the minimum contention parameter, the lower bound of the window is set at the lower bound of this interval, and the upper bound of the window is to be chosen. The contention parameters are assumed to be independently generated from a uniform distribution in (0, 1]. When the distribution functions are identical but nonuniform, the contention parameters can be transformed by the distribution function into uniformly distributed parameters. Four algorithms to determine the upper bound of the window are described in this section. These algorithms assume that the channel load and the distribution functions from which the contention parameters are generated are exactly known. Methods to estimate the channel load will be presented in Section 7.5.5. The performance is worse when the channel load is estimated. Lastly, issues on finding the distribution functions and implementation are discussed.

7.5.1. Binary-Divide Window Control

A straightforward way to choose the upper bound of the window in each iteration is to set it midway in the interval containing the minimum. Binary search is applied in each iteration to eliminate half of the remaining interval. This method provides a lower bound on the performance.

The overhead is analyzed in terms of the number of iterations of the protocol to determine the minimum. In any given step, if the window size is greater than the distance between the two smallest parameters, y_1 and y_2, the minimum may be isolated depending on the relative positions of y_1, y_2, and the window (Figures 7.11a and 7.11b). On the other hand, if the window is reduced to a size smaller than the distance between y_1 and y_2, and the bounds of the window are updated according to the procedure in Figure 7.9, the minimum will always be isolated in such a window. This is illustrated in Figure 7.11c. Hence, the maximum number of iterations to resolve the minimum never exceeds the number of steps to reduce the window to a size smaller than the distance between y_1 and y_2. Assuming that k steps are required, the following condition holds:

$$2^{-k} < y_2 - y_1 < 2^{-(k-1)} \tag{7.4}$$

Taking the logarithm of the inequality in Eq. (7.4) and rearranging it,

$$\left[-\frac{\log_e(y_2 - y_1)}{\log_e 2} \right] < k < \left[1 - \frac{\log_e(y_2 - y_1)}{\log_e 2} \right] \tag{7.5}$$

This inequality gives the upper bounds of the binary-divide window-control rule for given y_1 and y_2.

From the theory of ordered statistics (9), if the y_is are uniformly distributed in (0,1], then the joint probability density function of y_1 and y_2 is

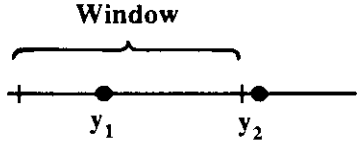
$$f_{Y_1Y_2}(y_1, y_2) = \begin{cases} \dfrac{n!}{(n-2)!}(1 - y_2)^{n-2} & \text{for } 1 \geq y_2 > y_1 \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.6}$$

From Eqs. (7.5) and (7.6), $E(k)$, the average number of iterations to resolve contentions in the binary-divide window-control rule, can be obtained by integrating the weighted upper bound over the domains of y_1 and y_2.

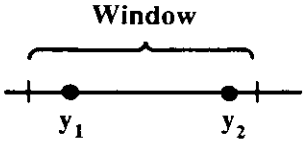
$$\begin{aligned} E(K) &< \int_0^1 \int_0^{y_2} \left[1 - \frac{\log_e(y_2 - y_1)}{\log_e 2} \right] \frac{n!}{(n-2)!}(1 - y_2)^{n-2}\, dy_1 dy_2 \\ &= 1 - \frac{n!}{(n-2)!\log_e 2} \int_0^1 \left[\int_0^{y_2} \log_e(y_2 - y_1)\, dy_1 \right] (1 - y_2)^{n-2}\, dy_2 \end{aligned} \tag{7.7}$$

Because $\int_0^{y_2} \log_e(y_2 - y_1)\, dy_1 = y_2 \log_e y_2 - y_2$, Eq. (7.7) can be simplified as

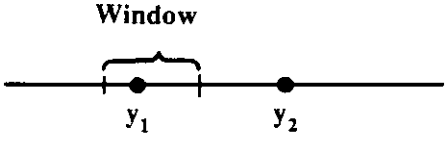
$$E(K) < 1 - \frac{n!}{(n-2)!\log_e 2} \int_0^1 (1 - y_2)^{n-2}(y_2 \log_e y_2 - y_2)\, dy_2 \tag{7.8}$$



(a) Contention is resolved by a window with size greater than $(y_2\text{-}y_1)$.



(b) Contention is not resolved by a window with size greater than $(y_2\text{-}y_1)$.



(c) Contention is always resolved if size of window is smaller than $(y_2\text{-}y_1)$ and lower bound of window is smaller than y_1.

Figure 7.11 Possible sizes and positions of a window during a contention step.

The integration in Eq. (7.8) can be evaluated to become

$$\int_0^1 (1 - y_2)^{n-2} y_2 \log_e y_2 \, dy_2 = H_n - H_{n-1} \tag{7.9}$$

where H_n is the harmonic mean of the series $\{1, 2, \ldots, n\}$, i.e.,

$$H_n = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{i} \tag{7.10}$$

The harmonic mean is approximately equal to $[\log_e n + \gamma + O(1/n)]/n$ (49), where γ is a constant. Hence, from Eqs. (7.7) through (7.10), we obtain

$$E(k) < 1 - \frac{n(n-1)}{\log_e 2} \left[\left(\frac{\log_e n}{n} - \frac{\log_e (n-1)}{n-1} \right) + \left(\frac{1}{n} - \frac{1}{n-1} \right) \right] \tag{7.11}$$

Because $\log_e n \approx \log_e (n-1)$ for large n, Eq. (7.11) may be reduced to

$$E(k) < 1 + \frac{n(n-1)}{\log_e 2} \frac{(1 + \log_e n)}{n(n-1)} < 3 + \log_2 n \tag{7.12}$$

Hence,

$$E(K) = O(\log_2 n) \tag{7.13}$$

In addition to the preceding analysis, simulations have been conducted to evaluate the performance of the binary-divide window-control rule. The simulation program was written in FORTRAN 77 and was executed on a DEC VAX 11/780 computer. In each simulation run, N random numbers were first generated in (0, 1], and successive windows were generated until the station with the minimum parameter was obtained. A 95% confidence interval of ± 0.1 was used in the simulations. The results are plotted in Figure 7.12. Note that the average number of iterations is smaller than $O(\log_2 n)$, which confirms that $O(\log_2 n)$ is the upper bound of the average performance.

7.5.2. Dynamic-Programming Window Control

The size of the window in each iteration of the window protocol can be controlled by a dynamic-programming algorithm that minimizes the expected total number of iterations before the minimum is isolated. The following notations are first defined:

$N(a, b)$ the minimum expected number of iterations to resolve con-

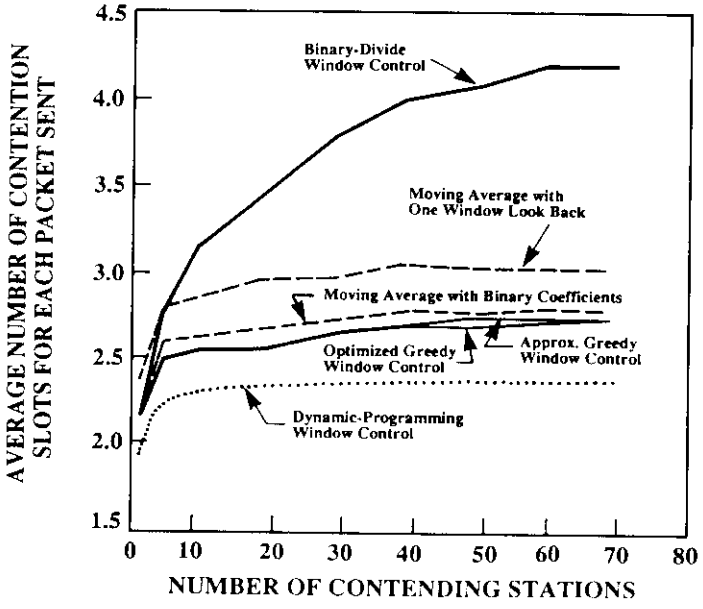


Figure 7.12 Performance of the window protocol with different window-control and load-estimation methods. (Solid lines assume that the channel load is exactly known; for dashed lines, the channel load is evaluated from previous experience.)

tention given that there are n contention parameters in $(a, U]$ and collision occurs in the current window $(a, b]$

$g(w, n, a, b)$ probability of *success* in the next iteration if a window of $(a, w]$, $a < w < b$, is used

$\ell(w, n, a, b)$ probability of *collision* in the next iteration if a window of $(a, w]$, $a < w < b$, is used

$r(w, n, a, b)$ probability of *no transmission* in the next iteration if a window of $(a, w]$, $a < w < b$, is used

It follows directly from the preceding definitions that

$$\ell(w, n, a, b) + g(w, n, a, b) + r(w, n, a, b) = 1 \tag{7.14}$$

As the Principle of Optimality is satisfied, the problem of minimizing the expected total number of iterations is reduced to that of finding w that minimizes the expected number of future iterations should collision or no transmission be detected in the current iteration. The problem can be formulated recursively as

$$N(a, b) = \min_{a<w<b} \{1 + 0 \cdot g(w, n, a, b) + N(a, w) \cdot \ell(w, n, a, b) + N(w, b) \cdot \gamma(w, n, a, b)\} \tag{7.15}$$

The probabilities $g(w, n, a, b)$, $\ell(w, n, a, b)$, and $r(w, n, a, b)$ can be derived from the distributions of the contention parameters and the state of contention. When transmission is unsuccessful, it is always possible to identify a window $(a, b]$ such that at least two of the x_is lie in $(a, b]$ and no x_i is smaller than a. This condition is designated as event A.

A = {at least two x_is are in $(a, b]$, given that all x_is are in $(a, U]$}

Suppose that the window is reduced to $(a, w]$, $a < w < b$, in the next iteration, three mutually exclusive events corresponding to three possible outcomes can be identified:

B = {exactly one of the x_is is in $(a, w]$, given that all x_is are in $(a, U]$}

C = {no x_i is in $(a, w]$, given that all x_is are in $(a, U]$}

D = {more than one x_i is in $(a, w]$, given that all x_is are in $(a, U]$}

From these events, the probabilities can be expressed as

$$g(w, n, a, B) = Pr\{B \mid A\} = \frac{Pr\{A \cap B\}}{Pr\{A\}} \tag{7.16}$$

$$r(w, n, a, b) = Pr\{C \mid A\} = \frac{Pr\{A \cap C\}}{Pr\{A\}} \tag{7.17}$$

The set $A \cap B$ represents the events that exactly one of the $x_i s$ is in $(a, w]$, that at least one x_i is in $(w, b]$, and that all others are in $(w, U]$. The set $A \cap C$ represents the event that at least two $x_i s$ are in $(w, b]$, given that all $x_i s$ are in $(w, U]$.

Let $F_i(x)$ (resp. $f_i(x)$) be the distribution (resp. density) function that governs the generation of x_i, $1 \leq i \leq n$, where n is the number of contending stations. Then event A occurs with probability:

$$Pr(A) =$$

$$\frac{\prod_{i=1}^{n} [1 - F_i(a)] - \sum_{i=1}^{n} \left\{ [F_i(b) - F_i(a)] \prod_{\substack{j=1 \\ j \neq i}}^{n} [1 - F_j(b)] \right\} - \prod_{i=1}^{n} [1 - F_i(b)]}{\prod_{i=1}^{n} (1 - F_i(a))} \tag{7.18}$$

The first and last terms of Eq. (7.18) indicate the probabilities that all $x_i s$ are greater than a and b, respectively. The second term is the probability that exactly one of the $x_i s$ is in the window $(a, b]$. Similarly,

$$g(w, n, a, b) =$$

$$\frac{\sum_{i=1}^{n} [F_i(w) - F_i(a)] \times \left\{ \prod_{\substack{j=1 \\ j \neq i}}^{n} [1 - F_j(w)] - \prod_{\substack{j=1 \\ j \neq i}}^{n} [1 - F_j(b)] \right\}}{Pr(A) \prod_{i=1}^{n} (1 - F_i(a))} \tag{7.19}$$

$$r(w, n, a, b) =$$

$$\frac{\prod_{i=1}^{n} [1 - F_i(w)] - \sum_{i=1}^{n} \left[[F_i(b) - F_i(w)] \prod_{\substack{j=1 \\ j \neq i}}^{n} [1 - F_j(b)] \right] - \prod_{i=1}^{n} [1 - F_i(b)]}{Pr(A) \prod_{i=1}^{n} (1 - F_i(a))} \tag{7.20}$$

It follows that an optimal window can be derived in each iteration once the channel load and the distributions of contention parameters are known. However, the dynamic-programming formulation is continuous and requires infinite levels of recursion. Boundary conditions must be set to terminate the evaluations after a reasonable number of levels. In practice, the $x_i s$ may represent indistinguishable physical measures when their difference is less than δ. It is assumed that when the window size is small than δ, the prob-

ability that two stations have generated parameters in this interval is so small that contention can always be resolved in one step. The following boundary condition is included:

$$N(a, b) = 1 \qquad \text{for all } (b - a) < \delta$$

The value of δ was set to $1/(10 \times n)$ in our evaluations for continuous distributions, and to 1 for discrete distributions. The results of the evaluation are plotted in Figure 7.12, which shows that the average number of iterations is bounded by 2.4, independent of the number of contending stations. This performance is much better than that of the Binary Exponential Backoff Protocol of Ethernet (52) as shown in the simulation results in Figure 7.13. It must be pointed out the simulation results for the proposed window protocol assume that the number of contending stations is known, while those of the Binary Exponential Backoff protocol start out with one contending station. However, the advantage of the window protocol is that the channel load can be estimated easily from previous windows (Section 7.5.5), provided that the arrival rate does not change abruptly and that the degradation in performance with estimated loads is negligible. In case the channel load cannot be estimated and the binary-divide window-control protocol has to be used, the performance is still much better than that of Ethernet.

Arrow et al. had studied a similar problem with the difference that the number of contending stations in a collided window is assumed to be known exactly (1, 54). The problem was formulated into a *finite* recursion, and an

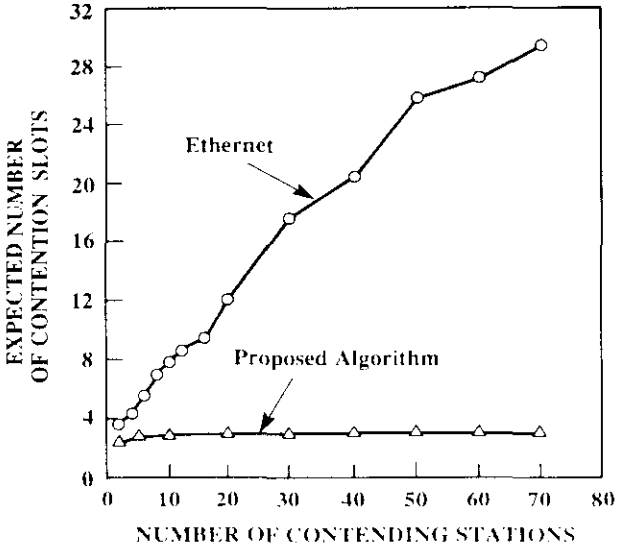


Figure 7.13 Comparison of Ethernet's Binary Exponential Backoff Protocol with the proposed window protocol.

asymptotic bound of 2.4 iterations was obtained by numerical evaluations. We have obtained comparable results when only ternary information on collision is available and the infinite dynamic programming tree is truncated. This shows that the information on the exact number of contending stations is insignificant.

Although optimal, the dynamic programming algorithm has a high computational complexity, which makes the algorithm impractical for real-time applications. As an example, the execution time to evaluate Eq. (7.15) on a DEC VAX 11/780 computer is 1.3 s for $n = 20$, and increases to 828 s for $n = 100$. Efficient hardware implementations will be discussed in Section 7.5.7.

7.5.3. Optimal Greedy Window Control

The optimization of window control using dynamic programming requires a high computational overhead because it examines the entire sequence of possible future windows to determine the window to be used in the next iteration. To reduce this overhead, only one future window may be examined. An optimal greedy window-control scheme is one that finds a window to maximize the probability of success, $g(w, n, a, b)$, in the next iteration. When the contention parameters have identical continuous distributions, $F(x)$, $g(w, n, a, b)$ can be expressed in a simple form as

$$g(w, n, a, b) = K[F(w) - F(a)]\{[1 - F(w)]^{n-1} - [1 - F(b)]^{n-1}\} \tag{7.21}$$

where $K = n/\{Pr(A)[1 - F(A)]^n\}$. It can be shown that Eq. (7.21) is unimodal between a and b, so a maximum exists in the window $(a, b]$. To find the optimal value of w, we set $(\partial/\partial w)g(w, n, a, b) = 0$ and solve for w. This derivation leads to the following equation if $f(w) \neq 0$:

$$[1 - F(w)]^{n-1} - [1 - F(b)]^{n-1} = (n - 1)[F(w) - F(a)][1 - F(w)]^{n-2} \tag{7.22}$$

If $z = 1 - F(w)$, Eq. (7.22) becomes

$$z^{n-1} - \frac{(n - 1)[1 - F(a)]z^{n-2}}{n} - \frac{[1 - F(b)]^{n-1}}{n} = 0 \tag{7.23}$$

It can be shown that a real root of Eq. (7.23) exists and satisfies the inequality $(1 - F(b)) < z_o < (1 - F(a))$. There is no closed-form solution to Eq. (7.23), and z_o has to be solved numerically. Once z_o is obtained, w_o, the upper boundary of the window, can be computed directly from z_o as

$$w_o = F^{-1}(1 - z_o) \tag{7.24}$$

The performance of the greedy scheme is measured by the average number of iterations expended before the minimum is identified. It has been

proved that the average number of iterations to resolve contention is bounded by 2.7 when the contention parameters are generated from a single distribution function [see Figure 7.12 (62)]. The computational overhead to solve Eq. (7.23) numerically is independent of n and is less than 1 s of CPU time on the DEC VAX 11/780 in most cases.

It is worth noting that a binary-divide window-control scheme is derived from the optimal greedy window-control scheme by setting n to 2. When n is 2, Eq. (7.23) is evaluated to become $F(w_o) = [F(a) + F(b)]/2$. If $F(y)$ is uniformly distributed in (0, 1], then $w_o = (a + b)/2$. The binary-divide control rule can also be used as a heuristic for window control with general distribution functions. It can be interpreted as one that always predicts that there are two contending stations. As a result, it performs well when the channel is lightly loaded, and degrades to have an $O(\log_2 n)$ performance when the channel load is heavy.

7.5.4. Approximate Greedy Window Control

The approximately greedy window-control scheme is similar to the optimal greedy window-control scheme except that an approximate equation on success probability is used. Eq. (7.21) may be rewritten as

$$g(w, n, a, b) = K[F(w) - F(a)][F(b) - F(w)][1 - F(w)]^{n-2} \sum_{i=0}^{n-2} v^i \tag{7.25}$$

where $v = [1 - F(b)]/[1 - F(w)]$. A function $\hat{g}(w, n, a, b)$ that has a maximum very close to that of $g(w, n, a, b)$, can be obtained by replacing the term $[\sum_{i=0}^{n-2} v^i]$ with $(n - 1)$. That is,

$$\hat{g}(w, n, a, b) = K'[F(w) - F(a)][F(b) - F(w)][1 - F(w)]^{n-2} \tag{7.26}$$

where $K' = (n - 1)K$. By solving $(\partial/\partial w) \log_e \hat{g}(w, n, a, b)] = 0$, we obtain

$$\frac{f(w)}{F(w) - F(a)} + \frac{f(w)}{F(w) - F(b)} + \frac{(n - 2)f(w)}{F(w) - 1} = 0 \tag{7.27}$$

or, equivalently,

$$[F(w)^2 + C[F(w)] + D = 0 \tag{7.28}$$

where

$$C = \frac{(n - 1)[F(a) + F(b)] + 2}{n}$$

$$D = \frac{F(a) + F(b) + (n - 2)F(a)F(b)}{n}$$

A solution to Eq. (7.28) in the window $(F(a), F(b)]$ is given by

$$F(w_a) = \frac{-C - \sqrt{C^2 - 4D}}{2} \tag{7.29}$$

The approximate window w_a as calculated from Eq. (7.29) gives a performance that is nearly as good as that of the optimal greedy scheme (see Figure 7.12). The computational overhead to calculate Eq. (7.29) is independent of n and can be done in less than 100 μs on the DEC VAX 11/780.

7.5.5. Load Estimations

Before the window-control protocol is carried out, the number of contending processors must be estimated from the distributions of the contention parameters and the statistics of previous channel activities. This information is essential in estimating an initial window and in controlling the dynamic changes in window sizes in the current contention period. A method based on maximum-likelihood estimation is described here.

After the t-th message is transmitted, the window $(L, w(t)]$ that successfully isolates the station with the minimum is known to all processors. A maximum-likelihood estimate of $n(t)$, the number of stations that have participated in the contention, can be computed from a likelihood function on the probability of success that the minimum lies in $(L, w(t)]$. Assuming that the contention parameters are independently and uniformly distributed in (0, 1], the likelihood function is derived as

$$LK(\hat{n}(t), w(t), 0) = Pr(0 < Y_1 < w(t) < Y_2)$$
$$= \hat{n}(t)w(t)(1 - w(t))^{\hat{n}(t)-1} \tag{7.30}$$

$LK(n(t), w(t), 0)$ is maximized at

$$\hat{n}(t) = \left\lceil \frac{-1}{\log_e(1 - w(t))} \right\rceil \quad 0 < w(t) < 1 \tag{7.31}$$

The number of contending stations to transmit the $(t + 1)$-th message can be obtained by adding to $\hat{n}(t)$ the difference between the possible arrivals after the t-th message has been transmitted. The average number of iterations to resolve contentions using this load-estimation method is 3.1 as shown in Figure 7.12.

Because the extremum is readily available when contention is resolved, this information can be "piggybacked" in the packet transmitted. Hence, an alternative estimate is based on the density function of this statistics. The conditional density of y_1 is

$$f_{Y_1}(y_1 \mid 0 < Y_1 < w < Y_2) = \frac{\int_w^t f_{Y_1Y_2}(y_1, y_2)\, dy_2}{\int_0^w \int_w^t f_{Y_1Y_2}(y_1, y_2)\, dy_2 dy_1} \tag{7.32}$$

Because the contention parameters are independently and uniformly distributed in (0, 1],

$$f_{Y_1Y_2}(y_1, y_2) = (n - 1)(1 - y_2)^{n-2} \tag{7.33}$$

Substituting Eq. (7.33) into Eq. (7.32) yields

$$f_{Y_1}(y_1 \mid 0 < Y_1 < w < Y_2) = \frac{1}{w} \tag{7.34}$$

This result shows that the distribution of y_1 is determined once the window $(0, w]$ is known. Therefore, no new information is gained by using this first-order statistic in estimating n.

The accuracy on load estimation can be improved by using information on previous windows that successfully isolate a single station. A technique in time-series analysis called Auto-Regressive-Moving-Average (ARMA) model can be applied to obtain an estimated window based on all previous windows, $w(1), w(2), \ldots, w(t)$. A simple example is computing a moving average, $w_{mv}(t)$, using the following formula:

$$w_{mv}(t) = \frac{w_{mv}(t - 1) + w(t)}{2} \tag{7.35}$$

The value of $w_{mv}(t)$ is then used in Eq. (7.30) to estimate the channel load. The performance of using ARMA load estimation is very close to that when the channel load is exactly known (see Figure 7.12).

7.5.6. Estimating the Distribution Functions of Contention Parameters

In applications such as load balancing and finding the highest-priority class, the distribution functions from which the contention parameters are generated are unknown and have to be estimated dynamically. Generally, the distribution functions are assumed, and parameters of the distribution functions are estimated from statistics collected. Because information on the distribution functions is essential and must be consistent for all sites to optimize the window search, independent monitoring of local information and information broadcast on the bus may be insufficient and may lead to unstable operations.

For loading balancing, a single site is responsible for collecting the distribution functions on local response times and distributing them to other sites (3). For scheduling transmissions with the highest-priority level, information on the priority levels of messages transmitted can be observed on the bus. As an example, let λ_i be the arrival rate of messages to the i-th priority level and t_i be the arrival time of the most recent packet in the i-th level that has been transmitted. Assuming a Poisson process for the packet arrivals, the probability that at least one station has a message in the i-th

priority level is

$$p_i = 1 - \int_{T-t_i}^{\infty} \lambda_i e^{-\lambda_i t}\, dt = e^{-\lambda_i (T-t_i)} \tag{7.36}$$

where T is the current time. The distribution that a station generates a message in the i-th priority level is

$$F_i(k) = \begin{cases} 0 & k < i \\ e^{-\lambda_i (T-t_i)} & i \leq k \leq P \\ 1 & k > P \end{cases} \tag{7.37}$$

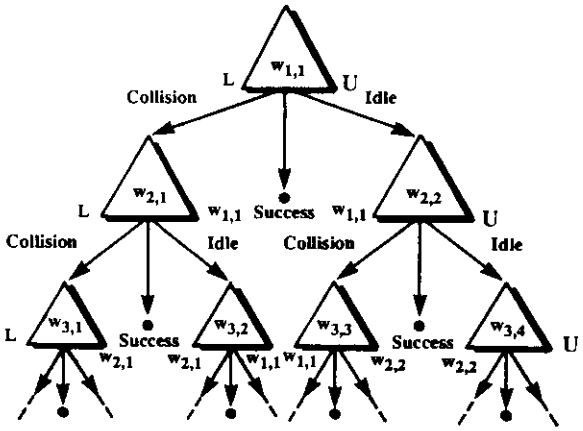
where P is the total number of priority levels in the system. The arrival time of a packet may be acquired by piggybacking this information on the packet transmitted. The packet arrival rate may be estimated by observing the packet arrival times.

The proposed window-control algorithms are quite robust with respect to changes in the distribution functions. Experiments on variations of the parameter of a Poisson distribution did not lead to any significant degradation in performance. However, there is always a delay between the time that the distribution function is changed and the time that this change is propagated to all sites. The optimization in the window protocol may be unstable if changes cannot be disseminated in time. The method for estimating the distribution functions is highly problem-dependent and is currently a problem under investigation.

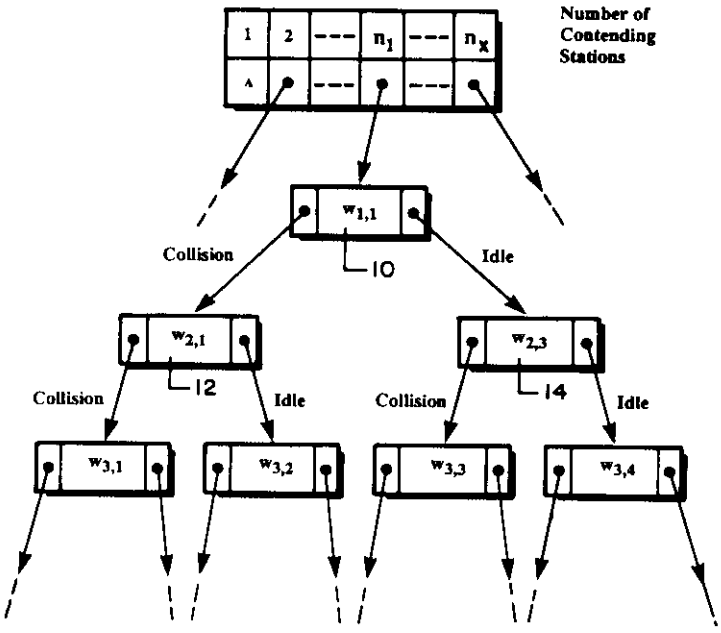
7.5.7. Implementation of the Window Protocol on Ethernet Interfaces

We have presented four window-control protocols in this section. Window control using dynamic programming requires a high computational overhead, while the other window-control algorithms require less computations but give poorer performance. The implementation on Ethernet-type interfaces has a stringent real-time requirement because each contention slot has a duration of less than 60 μs on a 10-Mbps network (12). The direct computation of the binary-divide and approximately greedy window-control schemes can satisfy this timing requirement. In this section, we describe a lookup-table method for implementing the dynamic-programming window control.

The sequence of windows evaluated by dynamic programming can be precomputed and stored in a lookup table. Given a channel load n, the sequence of optimal windows derived from Eq. (7.15) constitutes a binary decision tree (Figure 7.14a). The root of a subtree represents a window. The optimal window for the next iteration will reside in the left subtree if collision is detected in the current iteration. It will be in the right subtree if no trans-



(a) Binary Decision Tree



(b) Corresponding Data Structure

Figure 7.14 Lookup-table implementation of dynamic-programming window control. (a) Binary decision tree. (b) Corresponding data structure.

mission is detected. A set of binary trees, each of which corresponds to a channel load, can be constructed and stored as a lookup table in each station. The data structure for implementing the binary decision tree is shown in Figure 7.14b. The optimal window in each iteration can be retrieved efficiently in real time. The windows are evaluated based on a uniform distribution of the contention parameters. In applications where the contention parameters have identical but nonuniform distributions, they must be transformed by the distribution function into the uniform distribution before the lookup table is used.

One problem with the lookup-table method lies in the large memory space required. Since the average number of iterations is small, some subtrees can be pruned to reduce the memory space without significant degradation to performance. Window boundaries in the pruned subtrees have to be obtained by interpolation techniques. Likewise, for those channel loads for which no decision trees are stored, interpolation has to be used to obtain the window boundaries.

The lookup-table method has been designed on existing Ethernet interfaces (65, 68). A microcontroller, Intel MCS 8396, is placed between the Ethernet-protocol chip, Intel 82586, and the collision-detection chip, Intel 82501. A decision tree of four levels as evaluated by dynamic programming is used, and the microcontroller switches to binary-divide window control when more than four contention slots are needed. Sixteen-bit random numbers are used for the contention parameters and the entries of the decision tree. The channel load is assumed to vary from 1 to 100 stations. Hence, the total space required for storing the lookup table is 3 kbytes, which can fit in the 8-kbyte read-only memory of the MCS 8396. The performance of the truncated decision-tree method is less than 3.0 contention slots when $n \leq 100$ (Figure 7.12) as the number of slots to resolve contention is normally less than four.

The balanced binary tree in the preceding implementation simplifies the data structure. However, the performance can be improved if a skewed binary tree is used. The reasoning behind the skewed tree is that when a collision occurs, the left subtree is traversed and the size of the interval containing the minimum is small. In this case, a binary-divide control works well. On the other hand, when no transmission is detected, the right subtree is traversed and the size of the interval containing the minimum is not reduced significantly. In this case, the binary-divide control does not work well. Experimental results indicate that less than 2.5 slots are required to resolve a contention when a skewed binary tree with a height equal to n and a height of 1 for the left subtree of every nonterminal node (n is exactly known) is used. This means that $2n$ words are required for every dynamic programming tree. The total memory space required for n ranging from 1 to 60 stations is 7.3 kbytes.

7.6. Conclusions

In this chapter, we have shown that a class of resource-allocation problems for a local computer system connected by a multiaccess bus can be reduced to the problem of determining the extremum from a set of physically distributed random numbers. A distributed algorithm to identify the extremum in a constant average time independent of the number of contending stations is proposed. The correspondence between the properties of our design and the proposed methodology is summarized in Table 7.1. The load-independent behavior of the proposed algorithm is important because the number of contending stations to identify the extremum is usually large. Most existing contention-resolution algorithms, such as the Binary Exponential Backoff algorithm of Ethernet, are load-dependent and cannot be used to identify the extremum. The proposed algorithm can be implemented in hardware on a contention bus with the collision-detection capability. The overhead in each iteration is the time for a contention slot. On the other hand, it can also be implemented in software on existing multiaccess networks. In this case, two messages have to be transmitted in each iteration. It must be pointed out that the proposed window control is optimal in the sense of minimizing the number of iterations before the extremum is found, but is not optimal in minimizing the expected delay or maximizing the average throughput of the network.

The proposed algorithm requires the reliable transmission of collision and broadcast information to all processors. This may be difficult if the channel is noisy. Incorrect information received may cause indefinite contentions and the inability to identify the extremum. The problem can be resolved by broadcasting the extremum after it is found. Further, the proposed algorithm has a predictable average behavior. Significant deviation from this behavior can be used to indicate an unreliable channel.

Besides the resource-sharing applications discussed in this section, the

Table 7.1 Application of the Methodology to Design the Proposed Resource-Allocation Scheme in a Single Contention-Bus Network

Methodology	*Design*
Optimal allocation	Request of highest priority Resource of highest preference
Distributed algorithm	Distributed minimum-search
Primitive operation	Window search
Implementation	Collision detection
Results	No explicit message transfer 2.4 contention slots (optimal)

proposed algorithm can be extended to resolve contentions for multiple multiaccess or bit-parallel buses (31, 43, 61, 65), maintain consistency and process queries in distributed databases (66), and unify many existing adaptive CSMA protocols (28).

Problems

1. What are the advantages and disadvantages of maintaining a queue at each resource? Discuss the issue with respect to the processing speed of the resource and message delays in the resource-sharing interconnection network.
2. In a resource-sharing system with a central scheduler, status information of a request is obtained by transferring messages through a message-transfer subsystem. Assuming N requests are pending for service, how many message transfers are necessary for the scheduler to determine the request of the highest priority? If the distributed minimum-search algorithm is applied, how many message transfers are necessary?
3. If the resource-sharing interconnection network comprises multiple contention buses, multiple requests can be transmitted simultaneously. As a result, t requests of the highest priority have to be identified when there are t buses available. Modify the distributed minimum-search algorithm such that multiple buses can be utilized to search these requests in parallel.
4. After the successful completion of the distributed minimum-search algorithm, three events can be identified: (1) no x_i is in the interval $[L, a]$; (2) the minimum of x_is is in the interval $(a, w]$; and (3) the second minimum is in $(w, b]$. Formulate a maximum likelihood estimate of $\hat{n}$ based on these three events.
5. Suppose that the number of processors involved in the distributed minimum-search procedure is governed by a Poisson process. Use this a priori information together with the three events described in Problem (4) to formulate a Bayes estimate of $\hat{n}$.
6. In dynamic-programming window control, a boundary condition is included that results in a truncated dynamic programming tree. Show that the number of nodes in a truncated tree is proportional to the number of contending stations.

References

1. Arrow K, Pesotchinsky L, Sobel M: On partitioning a sample with binary-type questions in lieu of collecting observations. *Journal of the American Statistical Association*, Vol. 76, No. 374, June 1981, pp. 402–409.

2. Baer JL: *Computer Systems Architecture*. Computer Science Press, Rockville, MD, 1980.
3. Baumgartner KM, Wah BW: The effects of load balancing on response time for local computer systems with a multiaccess network. *Proc. International Conference on Communications, IEEE*, June 1985, pp. 10.1.1–10.1.5.
4. Berger T, Mehrauari N, Towsley D, Wolf JK: Random multiple access and group testing. *Trans. on Communications, Vol. COM-34, No. 7, IEEE*, July 1984, pp. 769–779.
5. Briggs FA, Fu KS, Hwang K, Wah BW: PUMPS architecture for pattern analysis and image database management. *Trans. on Computers, IEEE*, Vol. C-31, No. 10, Oct. 1982, pp. 969–983.
6. Capetanakis J: The Multiple-Access Broadcast Channel: Protocol and Capacity Considerations, Ph.D. Thesis, Massachusetts Institute of Technology, 1977.
7. Capetanakis J: Tree algorithm for packet broadcast channels. *Trans. on Information Theory, IEEE*, Vol. IT-25, No. 5, Sept. 1979, pp. 505–515.
8. Capetanakis J: Generalized TDMA: The multi-accessing tree protocol. *Trans. on Communications, IEEE*, Vol. COM-27, No. 10, Oct. 1979, pp. 1479–1484.
9. David HA: *Order Statistics*. John Wiley & Sons, New York, 1970.
10. Day JD: Resource sharing protocols. *Computer, IEEE*, Vol. 10, No. 9, Sept. 1977, pp. 47–56.
11. Dennis JB: Data flow supercomputers. *Computer, IEEE*, Vol. 13, No. 11, Nov. 1980, pp. 48–56.
12. Digital Equipment Corp., Intel Corp., and Xerox Corp., Ethernet: Local Area Network Data-Link Layer and Physical Layer Specifications, Version 1.0, Sept. 30, 1980.
13. Dijkstra EW: Cooperating Sequential Processes. In Genuys F (ed): *Programming Languages*, Academic Press, New York, 1968.
14. Dubois M, Briggs FA: Effects of cache coherency in multiprocessors. *Trans. on Computers, IEEE*, Vol. C-31, No. 11, Nov. 1982.
15. Enslow PH: Multiprocessor organization. *Computing Surveys, ACM*, Vol. 9, March 1977, pp. 103–129.
16. Feng TY: A survey of interconnection networks. *Computer, IEEE*, Dec. 1981, pp. 12–27.
17. Frankovich JM: A bandwidth analysis of baseline networks. *Proc. International Conference on Distributed Computing Systems, IEEE*, Oct. 1982, pp. 572–578.
18. Fuller SH, Harbison SP: The C.mmp Multiprocessor, Technical Report, Carnegie-Mellon University, Pittsburgh, PA, 1978.
19. Gallagher RG: Conflict resolution in random access broadcast networks. *Proc. AFOSR Workshop Communication Theory and Applications*, Sept. 17–20, 1978, pp. 74–76.
20. Gold YI, Franta WR: An efficient collision-free protocol for prioritized access-control of cable radio channels. *Computer Networks*, North-Holland, Amsterdam, Vol. 7, pp. 83–98.
21. Hansen PB: Distributed processes: A concurrent programming concept. *Communications of ACM*, Vol. 21, Nov. 1978, pp. 934–941.
22. Hayes JH: An adaptive technique for local distribution. *Trans. Communications, IEEE*, Vol. COM-26, No. 8, Aug. 1978.

23. Hoare CAR: Monitor: An operating system structure concept. *Communications of ACM*, Vol. 17, No. 10, Oct. 1974, pp. 549–557.
24. Hoare CAR: Communicating sequential processes. *Communication of ACM*, Vol. 21, No. 8, Aug. 1978, pp. 666–667.
25. Hluchyj MG: Multiple Access Communication: The Finite User Population Problem, Massachusetts Institute of Technology, Cambridge, MA, Nov. 1981.
26. Hwang K, et al: A Unix-based local computer network with load balancing. *Computer, IEEE*, Vol. 15, No. 4, April 1982, pp. 55–66.
27. Jayaraman B, Keller RH: Resource expressions for applicative language. *Proc. 1982 International Conference on Parallel Processing, IEEE*, Aug. 1982, pp. 162–167.
28. Juang JY, Wah BW: Unified window protocol for local multiaccess networks. *Proc. Third Annual Joint Conference of the IEEE Computer and Communication Societies, IEEE*, April 1984, pp. 97–104.
29. Juang JY, Wah BW: A multi-access bus-arbitration scheme for VLSI-densed distributed systems. *Proc. National Computer Conference*, AFIPS Press, Vol. 53, July 1984, pp. 13–22.
30. Juang JY, Wah BW: Optimal scheduling algorithms for resource sharing interconnection networks. *Proc. Eighth International Computer Software and Applications Conference, IEEE*, Nov. 1984, pp. 217–225.
31. Juang JY: Resource Allocation in Computer Networks, Ph.D. Thesis, Purdue University, West Lafayette, IN, Aug. 1985.
32. Kleinrock L: *Queueing Theory, 1*, Addison-Wesley, Reading, MA, 1972.
33. Kleinrock L, Tobagi FA: Packet switching in radio channels: Part 1—carrier sense multiple access modes and their throughput-delay characteristics. *Trans. on Communications, IEEE*, Vol. COM-23, No. 12, Dec. 1975, pp. 1400–1416.
34. Kleinrock L, Yemini Y: An optimal adaptive scheme for multiple access broadcast communication. *Proc. International Conference on Communications, IEEE*, 1978, pp. 7.2.1–7.2.5.
35. Kung HT: Why systolic architectures. *Computer, IEEE*, Vol. 15, No. 10, Jan. 1982, pp. 37–46.
36. Kurose JF, Schwartz M: A family of window protocols for time-constrained applications in CSMA networks. *Proc. Second Joint Conference of Computer and Communication Societies, IEEE*, 1983, pp. 405–413.
37. Leinbaugh DW: High-level specifications of resource sharing. *Proc. International Conference on Parallel Processing, IEEE*, Aug. 1981, pp. 162–163.
38. Leinbaugh DW: Selector: High-level resource schedulers. *Trans. on Software Engineering, IEEE*, Vol. SE-10, No. 11, Nov. 1984, pp. 810–824.
39. Liskov BH, et al: CLU Reference Manual (Lecture notes in Computer Science), 114, Springer-Verlag, 1981.
40. Liskov BH: On linguistic support for distributed programs. *Trans. on Software Engineering, IEEE*, Vol. SE-8, No. 3, May 1982, pp. 203–210.
41. Manner R: Hardware task/processor scheduling in a polyprocessor environment. *Trans. on Computers, IEEE*, Vol. C-33, No. 7, July 1984, pp. 626–636.
42. Metcalfe RM, Boggs DR: Ethernet: Distributed packet switching for local computer networks. *Comm. of the ACM*, Vol. 19, No. 7, July 1976, pp. 395–404.
43. Mok AK, Ward SW: Distributed broadcast channel access. *Computer Networks*, Vol. 3, 1979, pp. 327–335.

44. Mosley J, Hamblet P: A class of efficient contention resolution algorithms for multiple access channels. *Trans. on Communications, IEEE,* Vol. C-35, Feb. 1985, pp. 145–157.
45. Li MN, Hwang K: Optimal load balancing strategies for a multiple processor system. *Proc. Tenth International Conference on Parallel Processing, IEEE,* Aug. 1981, pp. 352–357.
46. Ni LM, Li X: Prioritizing packet transmission in local multiaccess networks. *Proc. Eighth Data Communications Symposium, IEEE,* 1983.
47. Ousterhout JK, Scelza DA, Sindhu PS: MEDUSA: An experiment in distributed operating system structure. *Communications of ACM,* Vol. 23, No. 2, Feb. 1980, pp. 92–105.
48. Patel JH: Performance of processor-memory interconnections for multiprocessors. *Trans. on Computers, IEEE,* Oct. 1981, pp. 771–780.
49. Reingold EM, Nievergelt JN, Deo N: *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1979.
50. Riche DM, Thompson K: The Unix time-sharing system. *Communications, ACM,* Vol. 17, No. 7, July 1974, pp. 1278–1308.
51. Selinger PG: State-of-the-art issues in distributed databases. *Trans. on Software Engineering, IEEE,* Vol. SE-9, No. 3, May 1983, pp. 218–219.
52. Shock JF, et al: Evolution of the Ethernet local computer network. *Computer, IEEE,* Vol. 15, No. 8, Aug. 1982, pp. 10–27.
53. Silberschatz A: Extending CSP to allow dynamic resource management. *Trans. on Software Engineering, IEEE,* Vol. SE-9, No. 4, July 1983, pp. 527–530.
54. Sobel M, Groll PA: Group testing to eliminate efficiently all defectives in a binomial sample. *Bell Systems Technical Journal,* Sept. 1959, pp. 1179–1252.
55. Shacham N: A protocol for preferred access in packet-switching radio networks. *Trans. on Communications, IEEE,* Vol. COM-31, No. 2, Feb. 1983, pp. 253–264.
56. Tanenbaum AS: *Computer Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1981.
57. Thomas RH: A resource sharing executive for the ARPANET. *Proc. National Computer Conference,* AFIPS Press, 1973, pp. 155–163.
58. Tobagi FA; Carrier sense multiple access with message-based priority functions. *Trans. on Communications, IEEE,* Vol. COM-30, No. 1, Jan. 1982.
59. Towsley D: Queueing network models with state-dependent routing. *Journal of ACM,* Vol. 27, No. 2, April 1980, pp. 323–337.
60. Towsley D, Venkatesh G: Window random-access protocols for local computer networks. *Trans. on Computers, IEEE,* Vol. C-31, No. 8, Aug. 1982, pp. 715–722.
61. Towsley D, Wolf JK: On adaptive polling algorithms. *Trans. on Communications, IEEE,* Vol. COM-32, Dec. 1984, pp. 1294–1298.
62. Wah BW, Hicks A: Distributed scheduling of resources on Interconnection networks. *Proc. National Computer Conference,* AFIPS Press, 1982, pp. 697–709.
63. Wah BW, Juang JY: Load balancing on local multiaccess networks. *Proc. Eighth Conference on Local Computer Networks, IEEE,* Oct. 1983, pp. 56–66.
64. Wah BW: A comparative study of distributed resource sharing on multiprocessors. *Trans. on Computers, IEEE,* Vol. C-33, No. 8, Aug. 1984, pp. 700–711.

65. Wah BW, Juang FY: An efficient contention-resolution protocol for local multiaccess networks. Pending patent application, Sept. 1984.
66. Wah BW, Lien YN: Design of distributed databases on local computer systems with multiaccess network. *Trans. on Software Engineering, IEEE*, Vol. SE-11, No. 7, July 1985, pp. 606–619.
67. Wah BW, Juang JY: Resource sharing for local computer systems with a single multiaccess network. *Trans. on Computers, IEEE*, Vol. C-34, Dec. 1985.
68. Wah BW, Li WQ: Interface design for efficient multiaccess networks, in press.
69. Walker B, et al: The LOCUS distributed operating system. *Proc. Ninth ACM Symposium on Operating System Principles, ACM*, 1983, pp. 49–70.
70. Wang YT, Morris JT: Load sharing in distributed systems. *Trans. on Computers, IEEE*, Vol. C-34, March 1985, pp. 204–217.