Resource Scheduling for Local Computer Systems with a Multiaccess Network

BENJAMIN W. WAH, SENIOR MEMBER, IEEE, AND JIE-YONG JUANG, MEMBER IEEE

Abstract — **Resource scheduling maps requests to a pool of resources to optimize a combination of resource usage, response time, network congestion, and scheduling overhead. The overhead of collecting the necessary status information for the scheduler is usually high, which results in the use of outdated status information and a degradation of performance. In this paper, we study resource scheduling based on a distributed state-dependent discipline for a system of processors connected by a local multiaccess network. The scheduling problem is reduced to the identification of the extremum from a set of physically dispersed random numbers. We propose a method of utilizing the primitive operations of collision detection and broadcast in multiaccess networks to efficiently distribute status information and to identify the extremum. The optimal performance of identifying the extremum is found to be constant on the average independent of the number of contending processors. The protocol can be implemented either by minor hardware modification of existing multiaccess-network interfaces or in software.**

Index Terms — **Broadcast, collision detection, distributed state-dependent resource scheduling, extremum search, load balancing, multiaccess network, resource sharing.**

I. Introduction

THE decreasing cost, the growth in technology, and the diversification of applications have caused computer systems to evolve from being centralized to being distributed. A *distributed computer system* may possess a large number of general- and special-purpose autonomous processing units interconnected by a network. Besides allowing communication among devices, the network facilitates *resource sharing* that allows a local task to be migrated and processed at a remote location of the network [36]. Tasks are migrated because the local processor does not have the required computational capabilities or data or has failed. A task may also be processed remotely if the expected turnaround time is better.

A resource can be logical, such as a shared database, or can be physical, such as a special-purpose functional unit. In this paper, a *resource* refers to a processor receiving the migrated task, and a *request generator* refers to the processor generating the migrated task. A computational device may play the roles of both request generator and resource. For example, a processor may migrate a task to compute the fast Fourier transform to a special-purpose processor, while accepting a task from another heavily loaded processor. In a system with load balancing, excess load in a heavily loaded processor is offloaded to another lightly loaded processor. In a multiprocessor system with a pool of identical (or sets of identical) systolic arrays performing special functions such as matrix inversion and sorting, a task is directed to any one or more of the free resources. Resource sharing is also an important element in data-flow machines. Tasks in node store are sent to a pool of identical processors for processing.

In resource scheduling, tasks are scheduled to any one or more or a pool of free resources that can optimize a combination of resource usage, response time, network congestion, and scheduling overhead. Resource scheduling decisions can be made in a centralized or a distributed manner. A *centralized* decision implies that status information is collected, and decisions to schedule are made at one location. An example would be a system with a job scheduler at one location that collects jobs and dispatches them to resources for processing. Theoretical studies on centralized load balancing have been made by Foschini [9], Chow and Kohler [7], Towsley [32], and Ni and Hwang [25]. The major problem lies in the overhead of collecting status information and jobs. If this overhead is large, scheduling decisions are frequently based on inaccurate and outdated status information, which could be detrimental to performance.

In contrast, a *distributed* resource scheduling scheme does not limit the scheduling intelligence to one processor. It avoids the bottleneck of collecting status information and jobs at a single site and allows the scheduler to react quickly to dynamic changes in the system state. We have studied distributed resource scheduling in which the status information of resources is propagated through the network and is available to all request generators [36], [16]. Requests are sent into the network without any destination tags, and the network is responsible to route the requests to the best available resource.

Resource scheduling can also be classified as deterministic or probabilistic [7]. A decision based on the current state of the system is *deterministic* or *state dependent*. For this type of decision, system performance is optimized by either minimizing or maximizing a system parameter such as response time, system time, or throughput. A decision is *probabilistic* if an arriving job is dispatched to the resources according to a set of branching probabilities that were collected from previous experience. Deterministic strategies have been found to perform better than probabilistic ones be-

Manuscript received May 7, 1985; revised August 27, 1985. This work was supported by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW.

B. W. Wah is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

J.-Y. Juang is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

0018-9340/85/1200-1144$01.00

cause the latter are sometimes insensitive to dynamic changes in system load. However, they usually have higher overhead and are more difficult to implement.

Distributed state-dependent resource scheduling has been found to give a good tradeoff between resource usage and scheduling overhead. However, the scheduler is often implemented on a network model that is independent of the hardware characteristics. Besides simplifying the design of strategies, a simple network model allows the software to be transportable, and permits many applications of different requirements to share the same network. A protocol hierarchy is used which allows the scheduler to interact with the lower levels through system calls. As a result, the status information needed in the control and interchange of data among sites must be formulated into messages that are recognized by the lower levels of the hierarchy. This mismatch between the characteristics of the physical network and the requirements of the scheduler results in inefficiency and increased complexity of the control strategies. To improve the performance, the capabilities of the network must be matched against the requirements of the scheduler.

An example is shown in the load balancing strategy of the Purdue Engineering Computer Network [13] which is a system of computers connected by a hybrid of Ethernet and point-to-point networks. The load balancing decisions are distributed: each processor decides whether to send its job for remote execution. A processor polls other processors for status information about their loads, decides which processor has the lowest load, and sends the job for remote processing if the turnaround time is shorter. This polling results in $O(n)$ messages for each job load-balanced where n is the number of computers in the system. More efficient solutions are proposed in this paper to reduce this overhead.

In this paper, we have studied distributed state-dependent resource scheduling for a local computer 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 *broadcast capability. Carrier-sense-multiaccess networks* with *collision detection* (CSMA/CD networks) belong to this class, and are exemplified by the Ethernet [30] [Fig. 1(a)].

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 be detected and stopped immediately. The time for a processor to assert that there is 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 [Fig. 1(b)]. Many contention-resolution protocols have been proposed and implemented [3], [5], [6], [10], [14], [15], [17], [18], [21]–[23], [33], [35]. They are distinguished by the different transmission control.

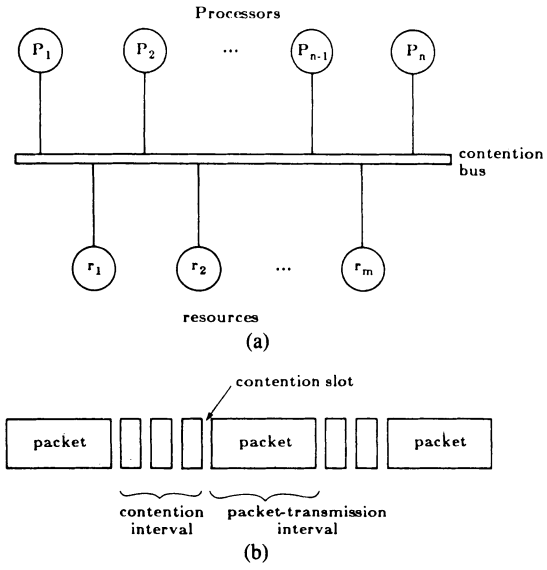


Fig. 1. (a) A resource sharing system connected by a single multiaccess bus. (b) The operations of a contention bus with alternating phases.

In this paper, we have studied the resource allocation problem in resource sharing systems connected by a contention bus. In Section II, a special class of resource allocation schemes is characterized. Optimal resource scheduling in this class of problems can be reduced to finding the extremum among a set of physically distributed random numbers. An efficient distributed window-search scheme is described in Section III. The proposed scheme can be integrated into the contention-resolution protocol of the bus or can be implemented by explicit message transfers. Section IV details the evaluation of alternative window-search schemes, and Section V draws conclusions.

II. 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 let 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. Since 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

$$\min_{(p,r)\in P\times R} H(x_p, y_r) \tag{1}$$

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

In general, the cost function H depends on the character-

istics of tasks and resources, as well as the interconnection network. It may be very complex and difficult to optimize. We will only stuay a special class of the cost functions that are monotonic with respect to x_p and y_r. That is,

$$\frac{\partial}{\partial x_p} H(x_p, y_r) \text{ is either positive or negative for all } x_p \text{ and } y_r \tag{2a}$$

$$\frac{\partial}{\partial y_r} H(x_p, y_r) \text{ is either positive or negative for all } x_p \text{ and } y_r . \tag{2b}$$

The above conditions imply that for a given resource, the cost is minimized by servicing a task of the highest priority (if (2a) is negative) or one with the lowest priority (if (2a) is positive). Similarly, for a given request, the cost is minimized by choosing a resource of the highest preference (if (2b) is negative) or one with the lowest preference (if (2b) is positive). For instance, if $(\partial/\partial x_p)H(x_p, y_r) \leq 0$ and $(\partial/\partial y_r)H(x_p, y_r) \geq 0$, then it follows directly from (1) and (2) that

$$\min_{(p,r)\in P\times R} H(x_p, y_r) = H(\max_{p\in P}(x_p), \min_{r\in R}(y_r)) . \tag{3}$$

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-resolution protocols in CSMA networks are designed to resolve contentions in using the channel. Since 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-Serve 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 [11], [26], [27], [31]. 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 [26]. This assumes that the highest priority level is equally likely to be any one of the P priority levels. Neither of the above 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 above criteria.

5) Resource Sharing of a Pool of Identical Resources: The priority of a request generator is an integer between one and P. The preference of a resource can be a random number in [0, 1] indicating its status (zero indicates that it is busy; any number between zero and one 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 step 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 [35], [2]. 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 the above examples, only linear functions on x_p and y_r are defined. In general, they can be any function satisfying (2a) and (2b).

A general organization of a resource scheduler is shown in Fig. 2. 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, $P = 6$ and $K = 2$ in Fig. 2.

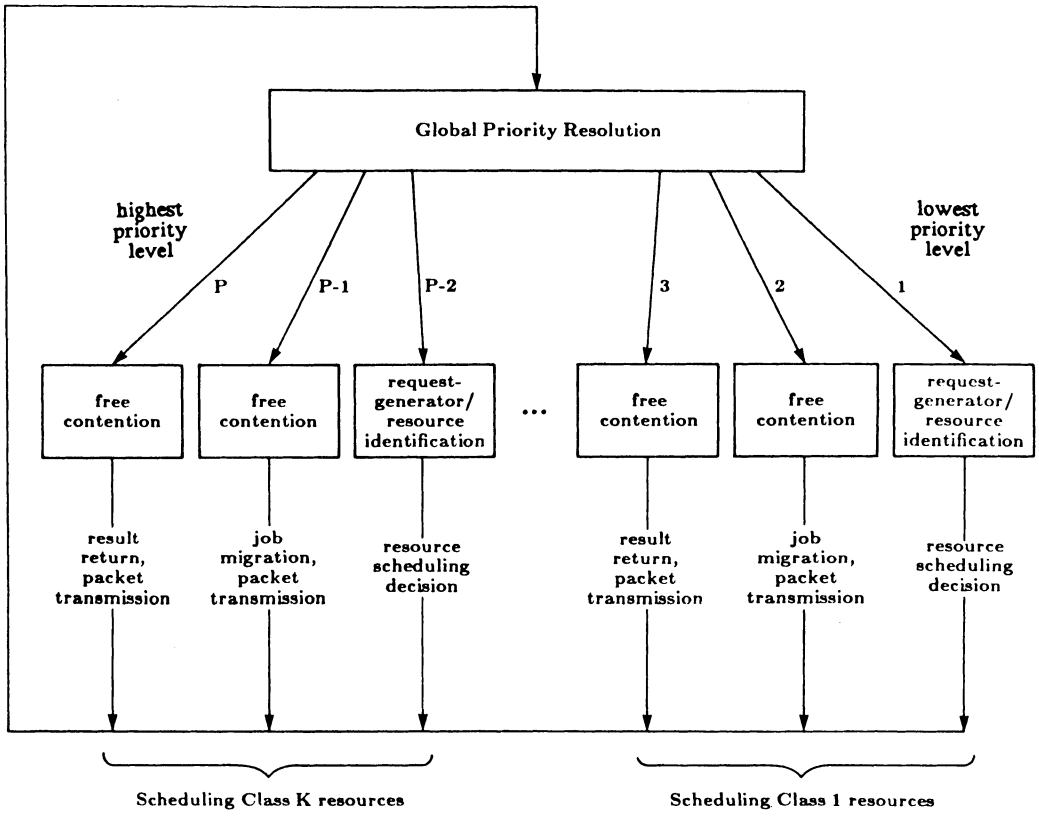


Fig. 2. A protocol to support resource sharing of multiple classes of resources connected by a multiaccess bus.

Class 1 tasks refer to load balancing operations, and class 2 tasks refer to message transmissions. Generally, within a class of resource scheduling problems, 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.

III. A Distributed Minimum-Search Algorithm

The operations of the resource scheduler in Fig. 2 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 paper. Specifically, the identification of the task with the highest priority in Fig. 2 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 zero and one. 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, are load-dependent, but perform 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 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 $\{x_1, \cdots, x_n\}$ in the interval between L and U, and that y_i is the ith smallest of the x_j's. 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 is 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 Fig. 3.

Fig. 4 illustrates the steps involved in the window-search scheme. Initially, five stations are ready, and they sense that the bus is free. Each 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 and 5 are eliminated in the first iteration. As stations 1, 2, and 4 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 [18] was studied by Hluchyj [14], 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 [23]. This protocol is a generalization of Gallagher's procedure [10] which is itself based on an idea from Hayes [12] and Capetanakis [4]. Towsley and Venkatesh [33] and Kurose and Schwartz [20] 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 [23]. 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 are more than one station with a parameter in the window, then a collision will be detected. If there is exactly one station with a parameter in the window, then a successful transmission will be detected. If there is no station with a parameter in the window, then an empty slot will be detected. Each iteration of the protocol in Fig. 3 will be completed in one contention slot. Hardware implementation will be discussed in Section IV-G after the window-control algorithms are presented.

In systems where modification to existing hardware is

```
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
                else
                     /* some other parameters are inside window, stop contending */
                     contending := false ]
      ]
      return (failure)
]
```

Fig. 3. Procedure illustrating the basic steps executed in each station for contending the channel with a three-state collision-detection mechanism.

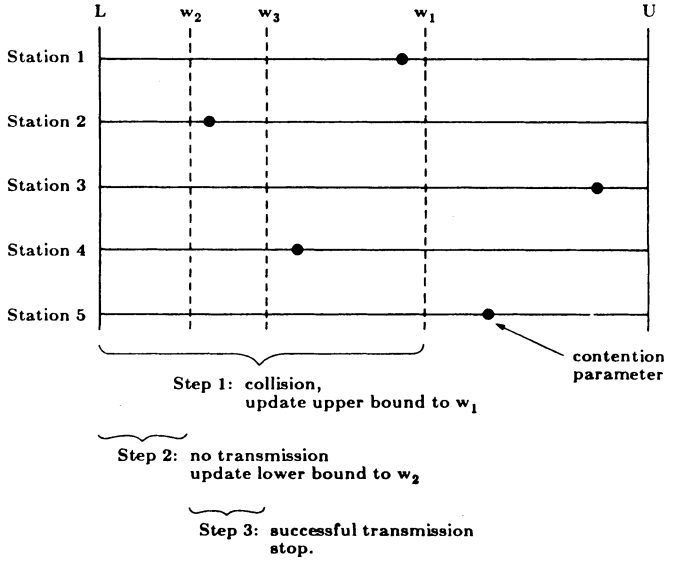


Fig. 4. An example illustrating the updates of the global window to isolate the station with the minimum contention parameter. (Braces indicate windows used in different steps.)

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 to performance. The window will be adjusted according to the minimum of the two broadcast contention parameters.

IV. 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 IV-E. The performance is worse when the channel load is estimated. Lastly, issues on finding the distribution functions and implementation are discussed.

A. 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 difference 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 [Fig. 5(a) and (b)]. On the other hand, if the window is reduced to a size smaller than the difference between y_1 and y_2, and the bounds of the window are updated according to the procedure in Fig. 3, then the minimum will always be isolated in such a window. This is illustrated in Fig. 5(c). 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 difference between y_1 and y_2. Assuming that k steps are required, the following condition holds:

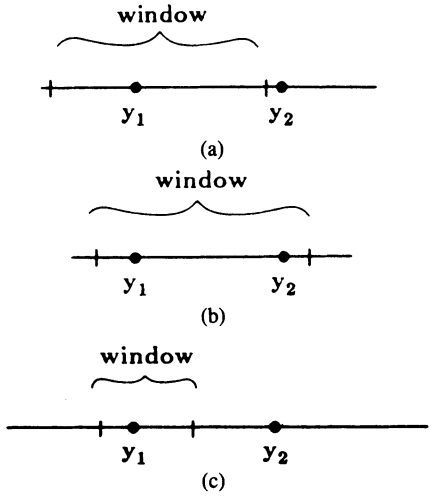


Fig. 5. Possible sizes and positions of a window during a contention step. (a) Contention is resolved by a window with size greater than $(y_2 - y_1)$. (b) Contention is not resolved by a window with size greater than $(y_2 - y_1)$. (c) Contention is always resolved if size of window is smaller than $(y_2 - y_1)$ and lower bound of window is smaller than y_1.

$$2^{-k} < y_2 - y_1 < 2^{-(k-1)}. \tag{4}$$

Taking the logarithm of the inequality in (4) and rearranging it,

$$\left[-\frac{\log_e(y_2 - y_1)}{\log_e 2}\right] < k < \left[1 - \frac{\log_e(y_2 - y_1)}{\log_e 2}\right]. \tag{5}$$

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 [8], if the y_i's are uniformly distributed in (0, 1], then the joint probability density function of y_1 and y_2 is

$$f_{Y_1Y_2}(y_1, y_2) = \begin{cases} \dfrac{n!}{(n-2)!}(1 - y_2)^{n-2} \cdot & \text{for } 1 \geq y_2 > y_1 \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

From (5) and (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.

$$E(K) < \int_0^1 \int_0^{y_2} \left[1 - \frac{\log_e(y_2 - y_1)}{\log_e 2}\right] \frac{n!}{(n-2)!}(1 - y_2)^{n-2}\, dy_1\, dy_2$$

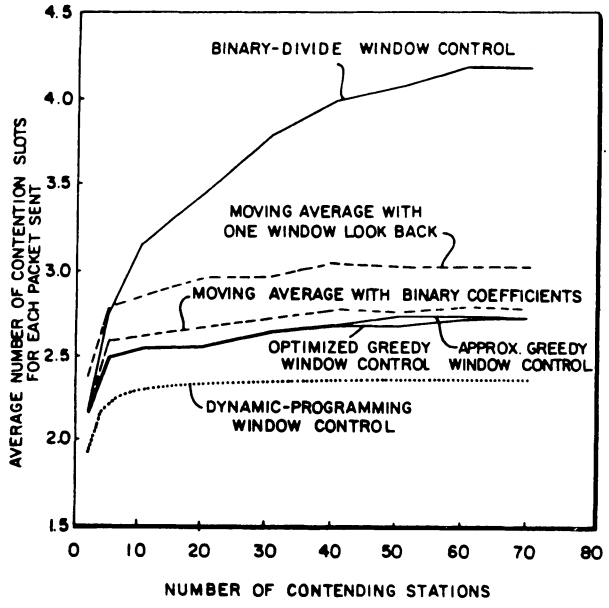


Fig. 6. Performance of the window protocol with different window-control and load-estimation methods. (Solid lines assume that the channel load is exactly known; for dashed lines, the channel load is evaluated from previous experience.)

$$= 1 - \frac{n!}{(n-2)!\log_e 2}\int_0^1\left[\int_0^{y_2}\log_e(y_2 - y_1)\,dy_1\right] \cdot (1-y_2)^{n-2}\,dy_2 . \quad (7)$$

Since $\int_0^{y_2}\log_e(y_2 - y_1)\,dy_1 = y_2\log_e y_2 - y_2$, (7) can be simplified as

$$E(K) < 1 - \frac{n!}{(n-2)!\log_e 2} \cdot \int_0^1 (1-y_2)^{n-2}(y_2\log_e y_2 - y_2)\,dy_2 . \quad (8)$$

The following integration can be evaluated to become

$$\int_0^1 (1-y_2)^{n-2} y_2 \log_e y_2\,dy_2 = H_n - H_{n-1} \quad (9)$$

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

$$H_n = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{i} . \quad (10)$$

The harmonic mean is approximately equal to $\log_e n + \gamma + O(1/n)/n$ [19] where γ is a constant. Hence, from (7)–(10), we obtain

$$E(k) < 1 - \frac{n(n-1)}{\log_e 2}\left[\left(\frac{\log_e n}{n} - \frac{\log_e(n-1)}{n-1}\right) + \left(\frac{1}{n} - \frac{1}{n-1}\right)\right] . \quad (11)$$

Since $\log_e n \simeq \log_e(n-1)$ for large n, (11) may be reduced to

$$E(k) < 1 + \frac{n(n-1)}{\log_e 2}\frac{(1+\log_e n)}{n(n-1)} < 3 + \log_2 n . \quad (12)$$

Hence,

$$E(K) = O(\log_2 n) . \quad (13)$$

In addition to the above 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 identified. A 95 percent confidence interval of ±0.1 was used in the simulations. The results are plotted in Fig. 6. 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.

B. 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 contention 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 above definitions that

$$\ell(w, n, a, b) + g(w, n, a, b) + r(w, n, a, b) = 1 . \quad (14)$$

As the Principle of Optimality is satisfied, the problem of minimizing the expected total number of iterations is reduced

to that of finding w which 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

$$N(a,b) = \min_{a<w<b} \{1 + 0 \cdot g(w,n,a,b) + N(a,w) \cdot \ell(w,n,a,b) + N(w,b) \cdot r(w,n,a,b)\}. \quad (15)$$

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_i's lie in $(a,b]$ and no x_i is smaller than a. This condition is designated as event A.

$$A = \{\text{at least two } x_i\text{'s are in } (a,b], \text{ given that all } x_i\text{'s are in } (a,U]\}.$$

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

$B = \{\text{exactly one of the } x_i\text{'s is in } (a,w], \text{ given that all } x_i\text{'s are in } (a,U]\}$

$C = \{\text{no } x_i \text{ is in } (a,w], \text{ given that all } x_i\text{'s are in } (a,U]\}$

$D = \{\text{more than one } x_i \text{ is in } (a,w], \text{ given that all } x_i\text{'s are in } (a,U]\}$.

From these events, the probabilities can be expressed as

$$g(w,n,a,b) = \Pr\{B \mid A\} = \frac{\Pr\{A \cap B\}}{\Pr\{A\}} \quad (16)$$

$$r(w,n,a,b) = \Pr\{C \mid A\} = \frac{\Pr\{A \cap C\}}{\Pr\{A\}}. \quad (17)$$

The set $A \cap B$ represents the event 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 $F_i(x)$ (respectively, $f_i(x)$) be the distribution (respectively density) function that governs the generation of x_i, $1 \le i \le n$ where n is the number of contending stations. Then event A occurs with probability

$$\Pr(A) = \frac{\prod_{i=1}^{n}[1 - F_i(a)] - \sum_{i=1}^{n}\left\{[F_i(b) - F_i(a)]\prod_{\substack{j=1\\ j\neq i}}^{n}[1 - F_j(b)]\right\} - \prod_{i=1}^{n}[1 - F_i(b)]}{\prod_{i=1}^{n}(1 - F_i(a))}. \quad (18)$$

The first and last terms of (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,

$$g(w,n,a,b) = \frac{\sum_{i=1}^{n}\left\{(F_i(w) - F_i(a)) \times \left[\prod_{\substack{j=1\\ j\neq i}}^{n}(1 - F_j(w)) - \prod_{\substack{j=1\\ j\neq i}}^{n}[1 - F_j(b)]\right]\right\}}{\Pr(A)\prod_{i=1}^{n}(1 - F_i(a))}. \quad (19)$$

$$r(w,n,a,b) = \frac{\prod_{i=1}^{n}[1 - F_i(w)] - \sum_{i=1}^{n}\left[[F_i(b) - F_i(w)]\prod_{\substack{j=1\\ j\neq i}}^{n}[1 - F_j(b)]\right] - \prod_{i=1}^{n}[1 - F_i(b)]}{\Pr(A)\prod_{i=1}^{n}(1 - F_i(a))}. \quad (20)$$

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 smaller than δ, the probability 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 one for discrete distributions. The results of evaluation are plotted in Fig. 6, 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 [28] as shown in the simulation results in Fig. 7. It must be pointed out that 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 always start out with the assumption that there is one contending station. However, the advantage of the window protocol is that the channel load can be estimated easily from previous windows (Section IV-E), provided that the arrival rate does not change abruptly, and that the degradation in performance with estimated loads is negligible. In the case that the channel load cannot be estimated and that 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 [29], [1]. The prob-

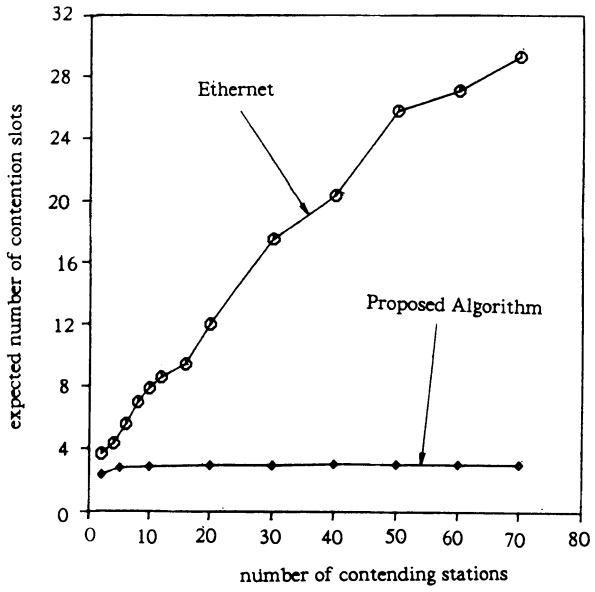


Fig. 7. Comparison of Ethernet's binary exponential backoff protocol to the proposed window protocol.

lem was formulated into a *finite* recursion, and an 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 (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 IV-G.

C. 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

$$g(w,n,a,b) = K[F(w) - F(a)]\{[1 - F(w)]^{n-1} - [1 - F(b)]^{n-1}\} \quad (21)$$

where $K = n/\{\Pr(A)[1 - F(a)]^n\}$. It can be shown that (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 $(\partial/\partial w) \cdot g(w, n, a, b) = 0$ and solve for w. This derivation leads to the following equation if $f(w) \neq 0$:

$$[1 - F(w)]^{n-1} - [1 - F(b)]^{n-1} = (n - 1)[F(w) - F(a)][1 - F(w)]^{n-2}. \quad (22)$$

Let $z = 1 - F(w)$; (22) becomes

$$z^{n-1} - \frac{(n-1)[1 - F(a)]z^{n-2}}{n} - \frac{[1 - F(b)]^{n-1}}{n} = 0. \quad (23)$$

It can be shown that a real root of (23) exists and satisfies the inequality $(1 - F(b)) < z_o < (1 - F(a))$. There is no closed-form solution to (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

$$w_o = F^{-1}(1 - z_o). \quad (24)$$

The performance of the greedy scheme is measured by the average number of iterations expended before the minimum is identified. The performance as obtained by simulations is suboptimal and approaches an average of 2.7 iterations to resolve contentions (see Fig. 6). The computational overhead to solve (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, (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 which 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.

D. Approximate Greedy Window Control

The approximate greedy window-control scheme is similar to the optimal greedy window-control scheme except that an approximate equation on success probability is used. Equation (21) may be rewritten as

$$g(w,n,a,b) = K[F(w) - F(a)][F(b) - F(w)][1 - F(w)]^{n-2} \sum_{i=0}^{n-2} v^i \tag{25}$$

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 substituting the term $(\sum_{i=0}^{n-2} v^i)$ by $(n-1)$. That is,

$$\hat{g}(w,n,a,b) = K'[F(w) - F(a)][F(b) - F(w)][1 - F(w)]^{n-2} \tag{26}$$

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

$$\frac{f(w)}{F(w) - F(a)} + \frac{f(w)}{F(w) - F(b)} + \frac{(n-2)f(w)}{F(w) - 1} = 0 \tag{27}$$

or, equivalently,

$$[F(w)]^2 + C[F(w)] + D = 0 \tag{28}$$

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 (28) in the window $(F(a), F(b)]$ is given by

$$F(w_a) = \frac{-C - \sqrt{C^2 - 4D}}{2}. \tag{29}$$

The approximate window w_a as calculated from (29) gives a performance that is nearly as good as that of the optimal greedy scheme (see Fig. 6). The computational overhead to calculate (29) is independent of n and can be done in less than 100 μs on the DEC VAX 11/780.

E. 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 tth message is transmitted, the window $(L, w(t)]$ that successfully isolate 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

$$LK(\hat{n}(t), w(t), 0) = \Pr(0 < Y_1 < w(t) < Y_2) = \hat{n}(t)w(t)(1 - w(t))^{\hat{n}(t)-1}. \tag{30}$$

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

$$\hat{n}(t) = \left\lceil \frac{-1}{\log_e(1 - w(t))} \right\rceil \qquad 0 < w(t) < 1. \tag{31}$$

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 tth message has been transmitted. The average number of iterations to resolve contentions using this load estimation method, together with the optimal greedy window-control scheme, is 3.1 as shown in Fig. 6.

Since 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

$$f_{Y_1}(y_1 \mid 0 < Y_1 < w < Y_2) = \frac{\int_w^1 f_{Y_1Y_2}(y_1, y_2)\, dy_2}{\int_0^w \int_w^1 f_{Y_1Y_2}(y_1, y_2)\, dy_2\, dy_1}. \tag{32}$$

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

$$f_{Y_1Y_2}(y_1, y_2) = n(n-1)(1 - y_2)^{n-2}. \tag{33}$$

Substituting (33) into (32) yields

$$f_{Y_1}(y_1 \mid 0 < Y_1 < w < Y_2) = \frac{1}{w}. \tag{34}$$

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 autoregressive-moving-average (ARMA) model can be applied to obtain an estimated window based on all previous windows, $w(1), w(2), \cdots, w(t)$. A simple example is to compute a moving average $w_{mv}(t)$ using the following formula:

$$w_{mv}(t) = \frac{w_{mv}(t-1) + w(t)}{2}. \tag{35}$$

The value of $w_{mv}(t)$ is then used in (30) to estimate the channel load. The performance of using ARMA load estimation and optimal greedy window control is very close to that when the channel load is exactly known (see Fig. 6).

F. 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. Since 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 load balancing, a single site is responsible for collecting the distribution function on local response times and distributing them to other sites. 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 ith priority level, and let t_i be the arrival time of the most recent packet in the ith 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 ith priority level is

$$p_i = 1 - \int_{T-t_i}^{\infty} \lambda_i e^{-\lambda_i t} \quad dt = e^{-\lambda_i (T-t_i)} \tag{36}$$

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

$$F_i(k) = \begin{cases} 0 & k < i \\ e^{-\lambda_i (T-t_i)} & i \leq k \leq P \\ 1 & k > P \end{cases} \tag{37}$$

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 topic under investigation.

G. Implementation of the Window Protocol on Ethernet Interfaces

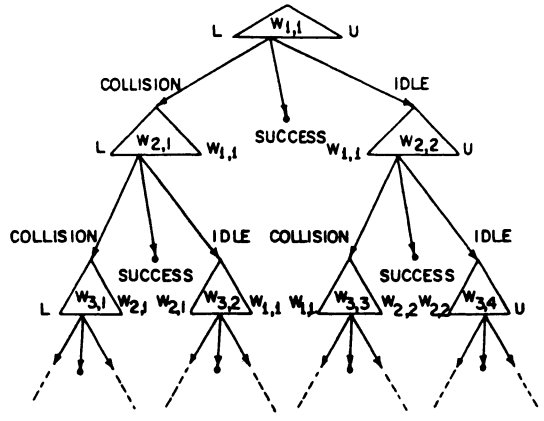
We have presented four window-control protocols in this paper. Window control using dynamic programming requires a high computational overhead, while the other window-control algorithms require less computations but give poorer performance. The implementation on Ethernet-type interfaces has a stringent real-time requirement because each contention slot has a duration of less than 60 μs on a 10-Mbit/s network [39]. The direct computation of the binary-divide and approximate greedy window-control (using a lookup table to compute the square root) 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 (15) constitute a binary decision tree [Fig. 8(a)]. 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 transmission 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 Fig. 8(b). 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.

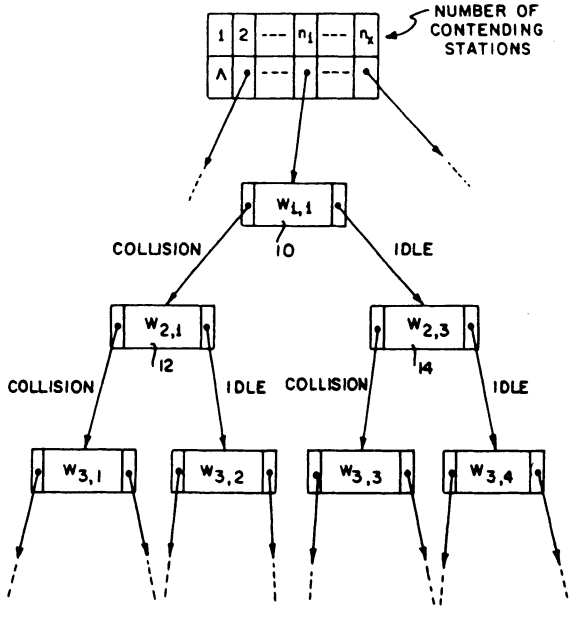
One problem with the lookup-table method lies in the large memory space required. Since the average number of iterations is small, some subtrees can be pruned to reduce the memory space without significant degradation to performance. Window boundaries in the pruned subtrees have to be obtained by interpolation 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 [37], [40]. 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 1–100 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 for a channel load between one and seventy stations (assuming that the channel load is exactly known).

The balanced binary tree in the above 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



(a)



(b)

Fig. 8. Lookup-table implementation of dynamic-programming window control. (a) Binary decision tree. (b) Corresponding data structure.

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 one 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 one to sixty stations is 7.3 kbytes.

V. Conclusions

In this paper, 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. This load-independent behavior 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 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 paper, the proposed algorithm can be extended to resolve contentions for multiple multiaccess or bit-parallel buses [24], [37], [34], [16], maintain consistency and process queries in distributed databases [38], and unify many existing adaptive CSMA protocols [15].

REFERENCES

[1] K. Arrow, L. Pesotchinsky, and M. Sobel, "On partitioning a sample with binary-type questions in lieu of collecting observations," *J. Amer. Statist. Ass.*, vol. 76, no. 374, pp. 402–409, June 1981.

[2] K. M. Baumgartner and B. W. Wah, "The effects of load balancing on response time for local computer systems with a multiaccess network," in *Proc. Int. Conf. Commun.*, June 1985, pp. 10.1.1–10.1.5.

[3] T. Berger, N. Mehrauari, D. Towsley, and J. K. Wolf, "Random multiple access and group testing," *IEEE Trans. Commun.*, vol. COM-34, pp. 767–779, July 1984.

[4] J. Capetanakis, "The multiple access broadcast channel: Protocol and capacity considerations," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, 1977.

[5] ——, "Tree algorithm for packet broadcast channels," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 505–515, Sept. 1979.

[6] ——, "Generalized TDMA: The multiaccessing tree protocol," *IEEE Trans. Commun.*, vol. COM-27, pp. 1479–1484, Oct. 1979.

[7] Y. C. Chow and W. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. C-28, pp. 334–361, May 1979.

[8] H. A. David, *Order Statistics*. New York: Wiley, 1970.

[9] G. J. Foschini, "On heavy traffic diffusion analysis and dynamic routing in packet switched networks," in *Computer Performance*, K. M. Chandy and M. Reiser Eds. New York: North-Holland, 1977.

[10] R. G. Gallagher, "Conflict resolution in random access broadcast networks," in *Proc. AFOSR Workshop Commun. Theory Appl.*, Sept. 17–20, 1978, pp. 74–76.

[11] Y. I. Gold and W. R. Franta, "An efficient collision-free protocol for prioritized access-control of cable radio channels," *Comput. Networks*, vol. 7, pp. 83–98, 1983.

[12] J. H. Hayes, "An adaptive technique for local distribution," *IEEE Trans. Commun.*, vol. COM-26, pp. 1178–1186, Aug. 1978.

[13] K. Hwang *et al.*, "A Unix-based local computer network with load balancing," *IEEE Computer*, vol. 15, pp. 55–66, Apr. 1982.

[14] M. G. Hluchyj, "Multiple access communication: The finite user population problem," Massachusetts Inst. Technol., Cambridge, Tech. Rep. LIDS-TH-1162, Nov. 1981.

[15] J. Y. Juang and B. W. Wah, "Unified window protocol for local multiaccess networks," in *Proc. 3rd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, Apr. 1984, pp. 97–104.

[16] J. Y. Juang, "Resource allocation in computer networks," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, Aug. 1985.

[17] L. Kleinrock and F. A. Tobagi, "Packet switching in radio channels: Part I—Carrier sense multiple access modes and their throughput-delay characteristics," *IEEE Trans. Commun.*, vol. COM-23, pp. 1400–1416, Dec. 1975.

[18] L. Kleinrock and Y. Yemini, "An optimal adaptive scheme for multiple access broadcast communication," in *Proc. Int. Conf. Commun.*, 1978, pp. 7.2.1–7.2.5.

[19] E. M. Reingold, J. N. Nievergelt, and N. Deo, *Combinatorial Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[20] J. F. Kurose and M. Schwartz, "A family of window protocols for time constrained applications in CSMA networks," in *Proc. 2nd Joint Conf. Comput. Commun. Soc.*, 1983, pp. 405–413.

[21] J. F. Kurose, M. Schwartz, and Y. Yemini, "Multiple-access protocols and time-constrained communication," *ACM Comput. Surv.*, vol. 16, no. 1, pp. 43–70, Mar. 1984.

[22] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, pp. 395–404, July 1976.

[23] J. Mosley and P. Humblet, "A class of efficient contention resolution algorithms for multiple access channels," *IEEE Trans. Commun.*, vol. COM-35, pp. 145–157, Feb. 1985; also Lab. for Inform. Dec. Sci., Massachusetts Inst. Technol., Cambridge, Tech. Rep. LIDS-P-1194, Dec. 1982.

[24] A. K. Mok and S. W. Ward, "Distributed broadcast channel access," *Comput. Networks*, vol. 3, pp. 327–335, 1979.

[25] L. M. Ni and K. Hwang, "Optimal load balancing strategies for a multiple processor system," in *Proc. 10th Int. Conf. Parallel Processing*, Aug. 1981, pp. 352–357.

[26] L. M. Ni and X. Li, "Prioritizing packet transmission in local multiaccess networks," in *Proc. 8th Data Commun. Symp.*, Cape Cod, MA, 1983.

[27] N. Shacham, "A protocol for preferred access in packet-switching radio networks," *IEEE Trans. Commun.*, vol. COM-31, pp. 253–264, Feb. 1983.

[28] J. F. Shock *et al.*, "Evolution of the Ethernet local computer network," *IEEE Computer*, vol. 15, pp. 10–27, Aug. 1982.

[29] M. Sobel and P. A. Groll, "Group testing to eliminate efficiently all defectives in a binomial sample," *Bell Syst. Tech. J.*, pp. 1179–1252, Sept. 1959.

[30] A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[31] F. A. Tobagi, "Carrier sense multiple access with message-based priority functions," *IEEE Trans. Commun.*, vol. COM-30, Jan. 1982.

[32] D. Towsley, "Queuing network models with state-dependent routing," *J. Ass. Comput. Mach.*, vol. 27, no. 2, pp. 323–337, Apr. 1980.

[33] D. Towsley and G. Venkatesh, "Window random access protocols for local computer networks," *IEEE Trans. Comput.*, vol. C-31, pp. 715–722, Aug. 1982.

[34] D. Towsley and J. K. Wolf, "On adaptive polling algorithms," *IEEE Trans. Commun.*, vol. COM-32, pp. 1294–1298, Dec. 1984.

[35] B. W. Wah and J. Y. Juang, "Load balancing on local multiaccess networks," in *Proc. 8th Conf. Local Comput. Networks*, Oct. 1983, pp. 56–66.

[36] B. W. Wah, "A comparative study of distributed resource sharing on multiprocessors," *IEEE Trans. Comput.*, vol. C-33, pp. 700–711, Aug. 1984.

[37] B. W. Wah and J. -Y. Juang, "An Efficient contention-resolution protocol for local multiaccess networks," pending U.S. Patent application, serial number 06/652645, Sept. 1984.

[38] B. W. Wah and Y. N. Lien, "Design of distributed databases on local computer systems with multiaccess network," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 606–619, July 1985.

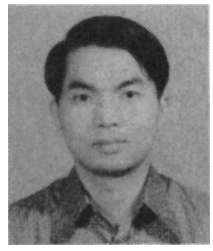
[39] Digital Equipment Corp., Intel Corp., and Xerox Corp., *Ethernet: Local Area Network Data-Link Layer and Physical Layer Specifications*, Version 1.0, Sept. 30, 1980.

[40] B. W. Wah and W. Q. Li, "Interface design for efficient multiaccess networks," under preparation.

Benjamin W. Wah (S'74–M'79–SM'85) received the B.S. and M.S. degrees in electrical engineering and computer science from Columbia University, New York, NY, in 1974 and 1975, respectively, the M.S. degree in computer science, and the Ph.D. degree in engineering, both from the University of California, Berkeley, in 1976 and 1979, respectively.

He was on the Faculty of the School of Electrical Engineering, Purdue University, West Lafayette, IN, between 1979 and 1985. He is now an Associate Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana. His current research interests include parallel computer architectures, computer networks, artificial intelligence, distributed databases, and theory of algorithms.

Dr. Wah received the IEEE Outstanding Paper Award in 1981. He has been a Distinguished Visitor of the IEEE Computer Society since 1983, and serves the Program Co-Chairman of the 1986 IEEE Data Engineering Conference.



Jie-Yong Juang (S'82–M'85) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1976, the M.S. degree in computer science from the University of Nebraska, Lincoln, in 1981, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1985.

He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL. His research interests include computer architectures, computer networks, distributed processing, and parallel processing.

Dr. Juang serves as the Tutorial Chairman of the 1985 IEEE COMPSAC Conference.