# LOAD BALANCING PROTOCOLS ON A LOCAL COMPUTER SYSTEM WITH A MULTIACCESS NETWORK

*Katherine M. Baumgartner and Benjamin W. Wah*

Department of Electrical and Computer Engineering
and the Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Avenue
Urbana, IL 61801

## ABSTRACT

Load balancing has been shown to be effective in reducing the average response time of jobs in local computer systems. In this paper, we study protocols for use in load balancing strategies that can be implemented on *existing* local computer systems connected by multiaccess network. The protocols use the existing broadcast capability of multiaccess networks to implement an efficient search technique for finding the extremum of a set of numbers. Load balancing can be implemented by using numbers which reflect the workloads in the computers. The load balancing strategy is practical and effective because it has a constant average overhead.

INDEX TERMS: Broadcast, collision detection, dynamic programming, load balancing, Markov chains, multiaccess networks.

## 1. 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* (DCS) may possess a large number of general and special-purpose autonomous processing units interconnected by a network. The primary function of the network is to allow communications among devices. A secondary function is resource sharing, a special form of which is load balancing. *Load balancing* uses communication facilities to support remote job execution in a user transparent fashion to improve resource utilization and reduce response time. A decision to load balance is made if the job is likely to be finished sooner when executed remotely than when executed locally. Load balancing is necessary since a job will almost always be waiting for service at one processor while another processor is idle in a DCS with ten or more processors [8].

Load balancing decisions can be made in a centralized or a distributed manner. A *centralized* decision implies that status information is collected, and decisions to load balance 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 stations for processing. Theoretical studies on centralized load balancing have been made by Chow and Kohler [4] and Ni and Hwang [10]. The disadvantage of centralized scheduling is the overhead of collecting processor status information and jobs. If this overhead is large, scheduling decisions are frequently based on inaccurate and outdated status information. In contrast, a *distributed* load balancing 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.

Load balancing can also be classified as state-dependent or probabilistic [4]. A decision based on the current state of the system is *state-dependent*. A decision is *probabilistic* if an arriving job is dispatched to the processors according to a set of branching probabilities that are collected from previous experience or are based on system characteristics. In the case that the branching probabilities are derived from the service rates of processors, the strategy is called *proportional branching* [4]. It was found that a probabilistic strategy for a single job class performed better than a proportional branching strategy with a single arrival stream [10]. An optimal probabilistic algorithm for multiple job classes was found to be easier to implement than state-dependent strategies. An optimal probabilistic load balancing algorithm with multiple arrival streams has also been shown [11]. Other research on load balancing include studies characterizing state-dependent load balancing, determining appropriate state information, and proposing efficient algorithms [2, 5, 7, 8, 15] and topology-dependent strategies [3, 14].

State-dependent load balancing is implemented on the Purdue Engineering Computer Network, which is a system of computers connected by a hybrid of Ethernet and point-to-point links [6]. The load balancing decisions are distributed: each processor decides whether to send its jobs 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.

Some results of these previous studies are as follows.

(1) A network with load balancing performs better than a network without load balancing.

(2) State-dependent load balancing strategies result in better performance than probabilistic strategies, but the overhead associated with implementing them is higher.

(3) Probabilistic strategies are sometimes insensitive to dynamic changes in system load and may result in suboptimal performance.

(4) Load balancing decisions considering the state of the source only do not have the potential for performance improvement that decisions considering the state of the server do [15].

(5) Extensive state information is not needed for effective load balancing and can be detrimental to system performance [5].

(6) Status information used in a state-dependent decision must be readily available. Decisions based on outdated or inaccurate status information could degrade the performance.

(7) Load balancing increases network load which can impede message transmissions.

This study considers load balancing on local computer systems connected by CSMA/CD networks. These networks have a broadcast bus topology that allows only one job or message to be sent across the network at a time. Response time is the amount of time elapsed from job submission to job completion and is an indication of the processor load. An efficient load balancing strategy will result in a minimum in response time and send a minimum of state information across the network. Due to the constraint of sending one job at a time across the bus, one such strategy is to send a job from the processor with the maximum load to the processor with the minimum load if the overhead of sending these jobs is small. This paper proposes window protocols for distributed search that can be implemented on *existing* CSMA/CD networks. These protocols can be used to implement an efficient load balancing strategy.

Research supported by National Science Foundation Grant DMC 85-19649.

International Conference on Parallel Processing, St. Charles, Illinois, August 1987.

The organization of this paper is as follows. The section following this introduction details a window protocol that can be used for distributed extremum search on bus networks and requires hardware modification to existing network interfaces. Section Three explains the extensions to this protocol for implementation on existing networks without hardware changes. Also described in Section Three is the implementation of a load balancing strategy using the distributed search. Section Four describes the implementation of and performance evaluation of a load balancing strategy using the distributed search, and conclusions are made in Section Five.

## 2. WINDOW PROTOCOL FOR DISTRIBUTED EXTREMUM SEARCHES

This section is divided into two parts. The first gives some background on various contention-resolution protocols. The second part describes a window protocol used for distributed extremum search. An example of the protocol is shown.

### 2.1. Contention Resolution

*Carrier-sense-multiaccess networks* with *collision detection* (CSMA/CD) are a type of local-area networks with packet switching and a bus topology [12]. CSMA/CD networks evolved from CSMA networks that have *listen-before-talk* protocols to avoid overlapping transmissions. The collision-detection ability of CSMA/CD networks allows processors to additionally *listen-while-talk*, so collisions resulting from simultaneous transmissions can be detected and stopped immediately.

There are three types of protocols for contention resolution in CSMA/CD networks. Collision-free protocols strictly schedule bus accesses, so no collisions occur. Contention protocols function at the other extreme allowing processors to transmit whenever they find the bus idle. When collisions occur because of simultaneous transmissions, processors stop transmitting, wait for some prescribed amount of time, and try again. The backoff algorithm of Ethernet [9] is an example in this class. The disadvantage of collision-free protocols lies in the overhead of waiting for transmission, while the disadvantage of contention protocols is the time wasted during collisions. A third type of contention-resolution protocol is the limited-contention protocol. This type of protocol chooses a processor for transmission from among those waiting to transmit based on a priori information, such as the channel load. The Virtual-Window Protocol proposed by Wah and Juang [13] is an example of a limited-contention protocol.

### 2.2. The Virtual-Window Protocol with Three-State Collision Detection

The *Virtual-Window Protocol* (VWP) uses a three-state collision-detection mechanism. After each attempted broadcast, there are three possible outcomes: *collision* (more than one broadcast), *idle* (no broadcast), and *success* (exactly one broadcast).

Stations wishing to transmit packets participate in a *contention period* that consists of a number of *contention slots*. Each station generates a random number called a *contention parameter* that is used for the entire contention period. The parameter is in an interval with upper and lower bounds U and L, respectively. Successive choices of smaller intervals in each contention slot attempt to isolate the minimum contention parameter. The steps performed by each station in a contention period are as follows.

```
procedure virtual_window_protocol;
/* choose_window()     function returning upper bound for the
 *                     next window
 * contention_state()  procedure returning state of the network
 * contend()           procedure to contend and broadcast
 * lb_window           lower bound for window to be chosen
 * ub_window           upper bound for window to be chosen
 * window_bd_up        actual upper bound of window chosen
 */
contending = true;
lb_window = L;
ub_window = U;
while contending do {
   /* operations performed in one contention slot */
   window_bd_up =
         choose_window(lb_window, ub_window);
   if (contention_parameter ≤ window_bd_up)
         and (contention_parameter > lb_window) then {
      contend(contention_parameter);
      contention_state(state);
      if (state = idle) then
         lb_window = window_bd_up;
      else if (state = collision) then
         ub_window = window_bd_up;
      else if (state = success) then
         contending = false;
   } else
      contending = false;
}
```

For regular message transfers, each station has equal chance of being chosen for transmission, so the contention parameters are random numbers generated from a uniform distribution on the interval (0,1]. The stations maintain a common window (interval) for contention. In a contention slot, stations having contention parameters within the window broadcast a short signal to contend for the channel. If a collision or no transmission occurs, the window boundaries are adjusted in parallel at all stations for the next contention slot. Stations having contention parameters outside the window stop contending and wait for the next contention period. The above steps are repeated until a single station is isolated in the window. This station is the winner and is allowed to transmit its packet. The distribution of the contention parameters and an estimate of the channel load are used to update the window efficiently, so the number of contention slots is kept to a minimum.

An example of the VWP is shown in Figure 1. There are five processors contending, and station i has contention parameter $x_i$. In this example, $x_1 = 0.48$, $x_2 = 0.90$, $x_3 = 0.35$, $x_4 = 0.30$, and $x_5 = 0.75$. These contention parameters were chosen arbitrarily, but for different purposes they may reflect processor loads or priorities. The windows chosen in these examples are not the optimal windows but are chosen to illustrate the operation of the protocols. $w_1$, the upper bound for the first window chosen, is 0.51. All stations with contention parameters less that or equal to 0.51 are allowed to broadcast, in this case stations 1, 3, and 4. The result of this contention slot is a collision, the interval to be searched is updated to (0, 0.51], and stations 2 and 5 are eliminated from the contention. $w_2$, the upper bound for the next window, is 0.25. The result of the second contention slot is idle (no broadcast), so the interval is updated to (0.25, 0.51]. No stations were eliminated as a result of this contention slot. For the third contention slot, the upper bound of the window is chosen to be 0.32. The result is a successful transmission, so station 4 is isolated and "wins" the contention.

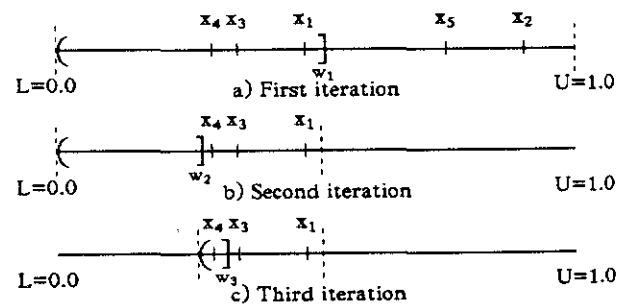



Figure 1. Example of the Virtual-Window Protocol. The dashed lines indicate the portion of the interval being searched during the current contention slot. The current window, enclosing stations eligible to contend, is delimited by ( ].

The window-selection process is formulated as a dynamic programming problem, and details have been shown elsewhere [14]. Analyses and simulations have shown that contention can be resolved in an average of 2.4 contention slots, independent of the number of contending stations and the distribution function of the contention parameters, if the parameters are independent and identically distributed [13].

The VWP can be implemented easily by minor hardware modifications of an existing Ethernet interface [14]. The global window can be maintained by updating an initially identical window with a common algorithm and using the identical information broadcast on the bus. Assuming that the information broadcast is received correctly by all stations, the global window will be synchronized at all sites.

## 3. WINDOW PROTOCOLS WITH TWO-STATE COLLISION DETECTION

The VWP locates the extremum of a set of independent contention parameters using information gathered during three-state collision detection. An iteration of the protocol is a contention slot. Additional hardware modifications to the network interface are required.

The VWP cannot be implemented easily at the applications level. Many existing networks do not make three-state collision-detection information available to the applications software because a contention slot is a small amount of time (50 to 100 microseconds) relative to the time required to propagate information through all levels of software to the applications level (hundreds of microseconds). At the applications level, each station has an independent *search parameter*, and an iteration is a *broadcast slot* which is a contention resolution at the network interface followed by a broadcast of a message to all stations. A broadcast slot has two possible outcomes, *idle* (no stations attempt to broadcast), or *transmission* (one or more stations attempt to broadcast resulting in contention resolution, and one station broadcasts its search parameter). A broadcast slot may consist of a number of contention slots, and information about each contention slot is not sent to the applications level.

There are a number of differences between searching at the applications level and the network level.

(1) The contention parameters are the search parameters for the VWP, which is not the case for the window protocol at the applications level.

(2) An iteration of the VWP is a contention slot with three possible outcomes: idle, collision, and success. An iteration of the window protocol at the applications level is a broadcast slot with two possible outcomes: transmission and idle.

(3) An iteration of the VWP takes less time than an iteration of the window protocol at the applications level. Normally, a contention slot takes tens of microseconds, while broadcasting a short message takes hundreds of microseconds.

Since the information available for window selection is different at the applications level, the decision process has to be modified. Three possible window-search strategies to identify the minimum are described below. They are the one-broadcast strategy, the two-broadcast strategy, and the combined strategy. The identification of the maximum is similar and is not described. For each strategy, the algorithm, an example, the technique for making window choices, and an implementation are shown. In contrast to the VWP, dynamic programming methods to optimize window choices are not used here because the Principle of Optimality is not satisfied. The performance of these strategies are compared using the number of broadcast slots they require to isolate the minimum search parameter.

### 3.1. One-Broadcast Strategy

The one-broadcast strategy allows a maximum of one broadcast slot per iteration. Starting with an interval (L,U], each station has a search parameter $x_i$ in the interval. The stations maintain a global window on the interval. Stations with parameters within the window attempt to broadcast their search parameters, and if there are one or more parameters in the window, there will be a contention resolution followed by a broadcast of one of the search parameters. In that case the upper bound of the interval will be updated to the value broadcast. If there are no parameters within the window, the lower bound of the interval is updated to the upper bound of the window used, and the protocol continues. The minimum is identified when the lower bound of the interval is equal to the upper bound. The steps each station performs are outlined below.

```
procedure one_broadcast_strategy;
/* choose_window()      function returning choice for upper bound of
 *                      next window
 * broadcast_state()    procedure returning results of broadcast slot
 * broadcast()          procedure to contend and broadcast
 * parameter_broadcast  search parameter broadcast if broadcast_state
 *                      returns transmission
 * lb_window            lower bound for window to be chosen
 * ub_window            upper bound for window to be chosen
 * window_bd_up         actual upper bound of window chosen
 */
searching = true;
lb_window = L;
ub_window = U;
while searching do {
   window_bd_up = choose_window(lb_window, ub_window);
   if (search_parameter ≤ window_bd_up) and
         (search_parameter > lb_window) then {
      broadcast(search_parameter);
      broadcast_state(state, parameter_broadcast);
      if (state = idle) then
         lb_window = window_bd_up
      else if (state = transmission) then
         ub_window = parameter_broadcast;
   } else
      searching = false;
   if (ub_window = lb_window) then
      searching = false;
}
```

An example of the one-broadcast strategy is shown in Figure 2. The stations and parameters are the same as the example in Figure 1. For the first iteration (Figure 2a), the upper bound of the window chosen is 0.51. Stations 1, 3, and 4 attempt to broadcast their parameters. Suppose that station 3 is the winner and transmits. The next interval to be searched is $(0,x_3)$. Let the upper bound of the next window chosen be 0.33. Only station 4 trys to transmit its parameter, and $x_4$ is broadcast. The search has not concluded even though $x_4$ is the minimum because the fact that it was the only station broadcasting is not available to station 4 or to the other stations. The next window chosen is 0.25. There is, of course, no broadcast. This process will continue until the bounds of the window isolate $x_4$, and the minimum is globally known.

The choice of the window in each broadcast slot is based on the probabilities of the two states, transmission and idle, which

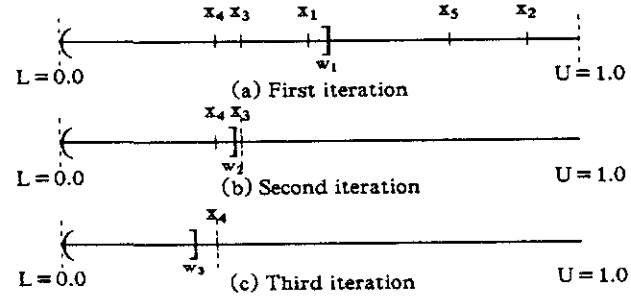



Figure 2. Example of the Window Protocol using the one-broadcast strategy. The dashed lines indicate the portion of the interval being searched during the current broadcast slot. The current window is delimited by ( ].

are dependent on previous broadcasts. If a previous broadcast slot resulted in the transmission of a value, say $x_{b1}$, then any subsequent transmissions must be less than $x_{b1}$. This implies that any subsequent $x_i$s broadcast were eligible to broadcast during the iteration that $x_{b1}$ was broadcast, but lost the contention. The probability of the subsequent transmissions must be conditioned on the fact that any $x_i$s in the current window did not broadcast when they were eligible during previous iterations. The choice of the window is, thus, dependent on previous broadcasts, hence, the choice cannot be optimized by dynamic programming methods because the Principle of Optimality is not satisfied.

Assume that station i has an independent search parameter $x_i$ with distribution $F(x)$ and density $f(x)$. The following definitions are used to formulate the problem of choosing the upper bound of the next window as a recurrence after k broadcasts.

$N_{E1}(a,b,v^k,q^k)$:
the minimum expected number of broadcast slots to isolate the minimum $x_i$ using a one-broadcast strategy, given that there have been k previous broadcasts with values and corresponding upper bounds of windows stored in the k-element arrays $v^k$ and $q^k$, respectively;

$\phi_E(a,b,w,v^k,q^k)$:
the probability of a transmission on the interval (a,w], given that there have been k previous broadcasts with values and corresponding upper bounds of windows stored in the k-element arrays $v^k$ and $q^k$, respectively;

$\theta_E(a,b,w,v^k,q^k)$:
the probability of idle on the interval (a,w], given that there have been k previous broadcasts with values and corresponding upper bounds of windows stored in the k-element arrays $v^k$ and $q^k$, respectively.

The notation $v^k$ and $q^k$ indicates a set of k values broadcast and the corresponding upper bounds of windows used. $v_k$ and $q_k$ are the $k^{th}$ value and the corresponding upper bound of window used. It follows directly from the above definitions that

$$\phi_E(a,b,w,v^k,q^k)+\theta_E(a,b,w,v^k,q^k)=1.0. \tag{1}$$

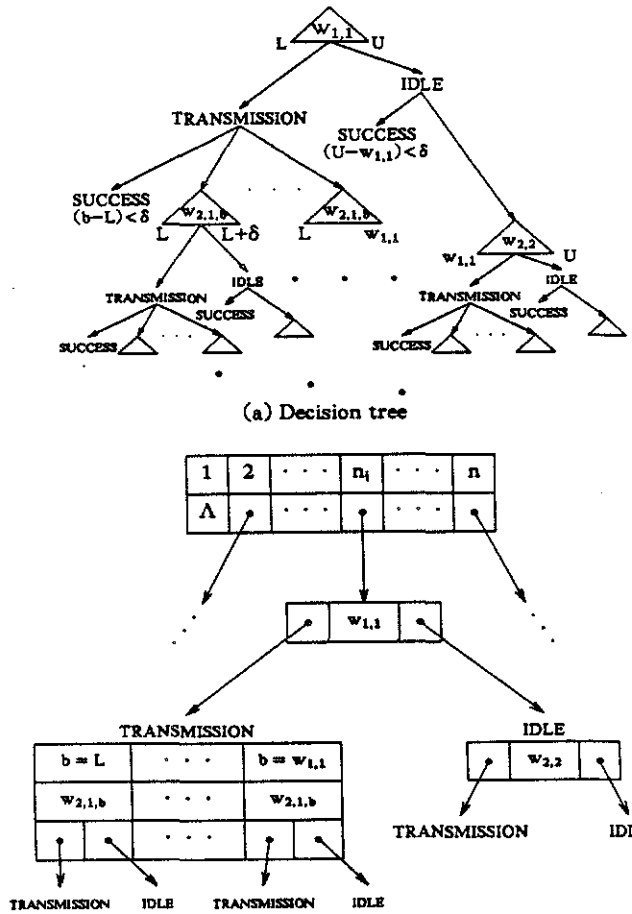
After k successful broadcasts, there are 2(k+1) subintervals on the interval (a,U]. They are (a, w], (w, $v_k$], ($v_k$, $q_k$], ($q_k$, $v_{k-1}$], ..., ($v_1$, $q_1$], and ($q_1$, U]. For reference, they can be numbered from left to right and from 1 to 2(k+1). Let s be a set of elements $\{s_i\}$, where $s_i$ is the number of $x_i$s in the $i^{th}$ subinterval. Let S be the set of s that are possible with the previous windows and values broadcast, and let I be a subset of S such that $s_1=0$. The set I is the subset of S that corresponds to a distribution of $x_i$s, such that there will be no $x_i$s in (a, w] and that the result of the broadcast slot is idle. Then

$$\theta_E=\frac{\sum_{s\in I}\left[\Pr[\text{arrangement } s]\prod_{i=1}^{k}\Pr\left[\begin{array}{l}v_i \text{ broadcast with a}\\ \text{window upper bound } q_i\end{array}\middle|\, s\right]\right]}{\sum_{s\in S}\left[\Pr[\text{arrangement } s]\prod_{i=1}^{k}\Pr\left[\begin{array}{l}v_i \text{ broadcast with a}\\ \text{window upper bound } q_i\end{array}\middle|\, s\right]\right]}. \tag{2}$$

The probability of a given arrangement is found using the distribution function $F(x)$. Let $b(a,b,i)=[F(b)-F(a)]^i$ then

$$\Pr[\text{arrangement } s]=\binom{n}{s_1}b(a,w,s_1)\binom{n-s_1}{s_2}b(w,v_k,s_2)\cdots\binom{n-\sum_{i=1}^{2k}s_i}{s_{2k+1}}b(v_1,q_1,s_{2k+1})b(q_1,U,s_{2k+2}).$$

$\Pr(v_k \text{ broadcast with a window upper bound } q_k|s)$ is easily determined because each station in the subinterval search has equal probability of winning and broadcasting in a broadcast slot, so




(a) Decision tree




(b) Data structure

Figure 3. Decision tree and data structure for the exact solution of the one-broadcast strategy. Each triangle in the decision tree indicates where a window choice is made. The data structure shows a method for storing the windows.

$$\Pr\left[\begin{array}{l}v_i \text{ broadcast with a}\\ \text{window upper bound } q_i\end{array}\middle|\, s\right]=\frac{1}{\sum_{j=1}^{2(k-i+1)}s_j}.$$

Using a conditional density function

$$f_c(a,w,x_b)=\frac{f(x_b)}{\Pr(a<x_b\leq w)}=\frac{f(x_b)}{F(w)-F(a)}, \tag{3}$$

the choice of the upper bound of the next window is formulated as a recurrence. Let

$$\gamma(a,w,v^k,q^k)=\int_a^w f_c(a,w,x_b)N_{E1}(a,x_b,v^{k+1},q^{k+1})dx_b$$

then

$$N_{E1}(a,b,v^k,q^k)=\min_{a<w<b}\left\{1+\phi_E(a,b,w,v^k,q^k)\gamma(a,w,v^k,q^k)+\theta_E(a,b,w,v^k,q^k)N_{E1}(w,b,v^k,q^k)\right\}, \tag{4}$$

with $N_{E1}(a,b,v^k,q^k)=1$ for $<$all b$=$a. The first term on the right hand side of Eq. (4) counts the current broadcast. The second term is the expected number of additional broadcast slots to isolate the minimum if the current broadcast slot results in a

transmission. $\gamma$ is the weighted average number of broadcast slots for the value broadcast, $x_b$, and the probability that this value was broadcast. The third term is the number of additional broadcast slots if the current broadcast slot is idle.

Boundary conditions must be set to terminate the evaluations after a reasonable number of broadcast slots. In practice, the $x_i$s may represent indistinguishable physical measures when their difference is less than $\delta$. It is assumed that when the window size is smaller than $\delta$, the probability that two stations have generated parameters in this interval is so small that contention can always be resolved in one step. The boundary condition becomes $N_{E1}(a,b,v^k,q^k)=1$ for all $(b-a)<\delta$

Using a $\delta=1/(10n)$, the evaluation of the recurrence equation is complex. The complexity becomes apparent when the window choices are shown in a decision tree in Figure 3. Each triangle in the decision tree corresponds to a decision point. The labels on the two lower corners show the lower and upper bounds of the interval to be searched, and the contents of the triangle, $w_{a,b,c}$, is the window upper bound for the current broadcast slot. The subscripts of the window upper bound indicate the iteration number, whether the last iteration outcome was transmission or idle (transmission = 1, idle = 2), and the value broadcast if the last iteration was a transmission. There are two branches from each decision point corresponding to the two possible outcomes in each broadcast slot. Starting from the root, if the broadcast slot results in a transmission of b, then the search will terminate if $(b-L)<\delta$; otherwise, the search will continue with a new decision point corresponding to the interval (L,b]. If the broadcast slot is idle, then the search will terminate if $(U-w_{1,1})<\delta$. Otherwise, the search continues with the interval $(w_{1,1},U]$. The data structure shown in Figure 3b is used to store the information in the decision tree. The top of the structure contains the roots of decision trees with different numbers of processors. For a given $n_i$, there is an initial window $w_{1,1}$, and two pointers to substructures corresponding to the two outcomes, transmission and idle. Note that the substructure for a transmission contains windows for each of the possible values that can be transmitted on the subinterval.

The tree shows the final window choices, but during computation of the best window choice for each decision point, all possible choices of windows have to be tried. For each possible window, there can either be a transmission or no transmission. If there is a transmission, all possible values within the window must be considered as the possible value broadcast. Each level of the tree indicates the outcome of an iteration. For every decision in the exact solution, the entire tree above the current decision point, which is determined by $v^k$ and $q^k$, must be taken into consideration in computing the next set of branching probabilities. With $\delta=1/(10n)$, and $n=5$, there are 69,007,690 decision points, and for $n=6$, the number increases to 8,501,194,726. The number of decision points increases so rapidly that the problem becomes intractable. Fortunately, reasonable results can be obtained using a heuristic decision based on the current upper and lower bounds only.

In the approximate solution, the probabilities of transmission and idle are assumed to be independent of previous broadcasts and are computed without information from previous broadcasts. The following definitions are used.

$N_{A1}(a,b)$:
the minimum expected number of broadcast slots to isolate the minimum $x_i$ using an approximate solution of the one-broadcast strategy, given that all $x_i$s are in (a,U], and that at least one $x_i$ is in (a,b];

$\phi_A(a,b,w)$:
the probability of a transmission on the interval (a,w], given that all $x_i$s are in (a,U], and that at least one $x_i$ is in (a,b];

$\theta_A(a,b,w)$:
the probability of no transmission on the interval (a,w], given that all $x_i$s are in (a,U], and that at least one $x_i$ is in (a,b].

It is obvious that

$$\phi_A(a,b,w)+\theta_A(a,b,w)=1.0. \tag{5}$$

There are two cases to consider when calculating $\theta_A(a,b,w)$, namely, $b=U$ and $b\neq U$. When $b=U$, it is uncertain whether there is a $x_i$ at b, and the arrangements of the n $x_i$s must be considered, so

$$\theta_A(a,U,w)=\frac{(F(U)-F(w))^n}{(F(U)-F(a))^n}. \tag{6a}$$

When $b\neq U$, there must be a station at b, since b is only updated to a value of $x_i$ in the event of a transmission. In this case we are only concerned with the placement of at most (n−1) of the $x_i$s.

$$\theta_A(a,b,w)\Big|_{b\neq U}=\frac{(F(U)-F(w))^{n-1}}{(F(U)-F(a))^{n-1}}.$$

The recurrence for choosing the window is

$$N_{A1}(a,b)=\min_{a<w<b}\left\{1+\phi_A(a,b,w)\left[\int_a^w f_c(a,w,x_b)N_{A1}(a,x_b)dx_b\right]+\theta_A(a,b,w)N_{A1}(w,b)\right\}. \tag{7}$$

Again, the three terms on the right hand side of the above equation count the current broadcast slot, additional broadcast slots in the event of a transmission, and additional broadcast slots if the current broadcast slot is idle.

The assumption that contention can be resolved in one step when the window size is smaller than $\delta$ holds, so the boundary condition $N_{A1}(a,b)=1$ for all $(b-a)<\delta$ is used again. The decision tree is the same as for the exact solution, but there is a savings since many of the nodes at different levels are duplicates. The data structure for storing the windows is simply a two-dimensional array. The number of decision points for the approximate solution is determined by the values of a and b. The total number of unique nodes with $\delta=1/(10n)$ is $((10n)^2+30n)/2$, which is determined by counting the decision points indicated by the above recurrences. For $n=5$ and $n=6$, the numbers of decisions points are 1325 and 1890, respectively, and the complexity of the solution is considerably reduced from the exact solution. The performance results of the one-broadcast strategy will be discussed in Section 3.4.

### 3.2. Two-Broadcast Strategy

A problem in the one-broadcast strategy is that it may take a large number of iterations to determine that there are no stations in the interval, even after the station with the minimum search parameter has broadcast. An alternative is to allow up to two broadcast slots per iteration and use the second slot to determine whether there are any stations with search parameters smaller than the parameter broadcast in the first slot. Initially, as with previous strategies, the interval is (L,U], and each station has a search parameter $x_i$ on the interval. A global window is determined, and stations with parameters within the window attempt to broadcast their search parameters. If there are no search parameters in the window, the strategy proceeds as in the one-broadcast strategy: the lower bound of the interval is updated to the window's upper bound, and the protocol continues. The difference between the one- and two-broadcast strategies occurs when there are parameters within the window. In this case, the upper bound is updated to $x_{b1}$, the value broadcast, and a second broadcast slot is allowed for all stations with $x_i<x_{b1}$. If the second slot is idle, $x_{b1}$ is the minimum, and the algorithm terminates. If there is a broadcast, the next iteration begins with $x_{b2}$, the second value broadcast, as the upper bound of the interval. Note that $x_{b2}$ is smaller than $x_{b1}$. The steps for each iteration are shown below.

```
procedure two_broadcast_strategy;
/* same parameters as procedure one_broadcast_strategy
*/
searching = true;
lb_window = L;
ub_window = U;
while searching do {
    window_bd_up = choose_window(lb_window, ub_window);
    if (search_parameter ≤ window_bd_up) and
          (search_parameter > lb_window) then {
        broadcast(search_parameter);
        broadcast_state(state, parameter_broadcast);
        if (state = idle) then
            lb_window = window_bd_up;
        else if (state = transmission) then{
            /* second broadcast attempt */
            if (search_parameter < parameter_broadcast) and
                  (search_parameter > lb_window) then {
                broadcast(search_parameter);
                broadcast_state(state, parameter_broadcast);
                if (state = idle) then
                    searching = false;
                else if (state = transmission) then
                    ub_window = parameter_broadcast;
            } else
                searching = false;
        }
    } else
        searching = false;
    if (ub_window = lb_window) then
        searching = false;
}
```

An example of the two-broadcast strategy is shown in Figure 4. The stations and contention parameters are the same as in previous examples. The first window chosen is (0.0.51], and as in the one-broadcast case, stations 1, 3, and 4 contend. Assume that station 1 wins and broadcasts $x_1$. To determine whether there are stations with parameters less than $x_1$, all stations with parameters less than $x_1$, namely, stations 3 and 4, are allowed to contend. Assuming station 3 wins, it is globally known that $x_1$ is not the minimum, and the interval to be searched next is updated to $(0, x_3)$. Let the next window chosen be 0.31. Station 4 is the only one allowed to broadcast, so it wins the contention. In the subsequent contention period, there is no broadcast, so it is globally known that $x_4$ is the minimum.

Each $x_i$ has search parameter $x_i$ with distribution $F(x)$ and density $f(x)$. The following definition is used.

$N_{E2}(a,b,v^k,q^k)$:
the minimum expected number of attempted broadcasts to isolate the minimum $x_i$ for an exact solution using the two-broadcast strategy, given that there have been k previous broadcasts with values and corresponding upper bounds of windows stored in the k-element arrays $v^k$ and $q^k$, respectively.

With $\phi_E$ and $\theta_E$ as before, and

$$\gamma(a,w,v^k,q^k)=\int_a^w f_c(a,w,x_{b1})\left[1+\phi_E(a,x_{b1},x_{b1},v^k,q^k)\int_a^{x_{b1}} f_c(a,x_{b1},x_{b2})N_{E2}(a,x_{b2},v^k,q^k)dx_{b2}\right]dx_{b1},$$

the recurrence for the two-broadcast case is

$$N_{E2}(a,b,v^k,q^k)=\min_{a<w<b}\left\{1+\phi_E(a,b,w,v^k,q^k)\gamma(a,w,v^k,q^k)+\theta_E(a,b,w,v^k,q^k)N_{E2}(w,b,v^k,q^k)\right\} \quad (8)$$

with the condition that $N_{E2}(a,b)=1$ for all $b=a$.

The same physical limitations exist as with the one-broadcast strategy, so the $x_i$s are considered indistinguishable when their difference is less than $\delta$, and the boundary condition is $N_{E2}(a,b)=1$ for all $(b-a)<\delta$. The decision tree implementing the above strategy is the same as that shown in Figure 3.

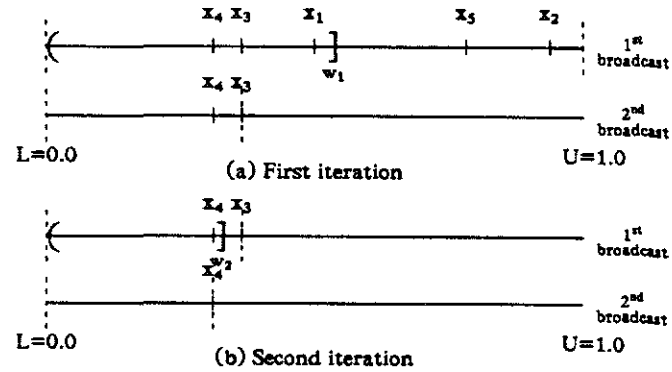



Figure 4. Example of the window search using the two-broadcast strategy. The dashed lines indicate the portion of the interval being searched during the current broadcast slot. The current window is delimited by ( ].

The exact solution becomes intractable when the number of processors is large as in the one-broadcast case, so an approximation that does not use the history information is adopted here. For the approximate solution, let

$N_{A2}(a,b)$:
the minimum expected number of attempted broadcasts to isolate the minimum $x_i$ for an approximate solution using the two-broadcast strategy, given that all $x_i$s are in (a,U], and that at least one $x_i$ is in (a,b].

and use $\phi_A$, and $\theta_A$ as before. Let

$$\gamma(a,w)=\int_a^w f_c(a,w,x_{b1})\left[1+\phi_A(a,x_{b1},x_{b1})\int_a^{x_{b1}} f_c(a,w,x_{b2})N_{A2}(a,x_{b2})dx_{b2}\right]dx_{b1}.$$

The recurrence is

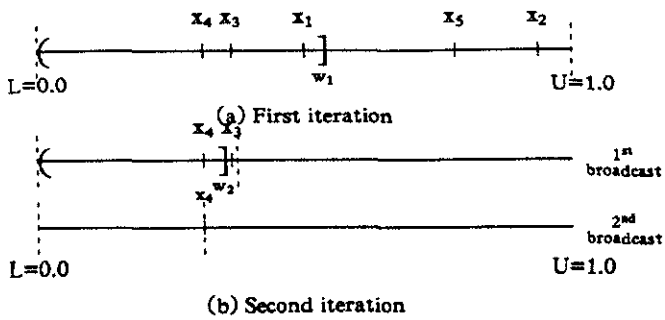
$$N_{A2}(a,b)=\min_{a<w<b}\left\{1+\phi_A(a,b,w)\gamma(a,w)+\theta_A(a,b,w)N_{A2}(w,b)\right\} \quad (9)$$

with the condition that $N_{A2}(a,b)=1$ or all $(b-a)<\delta$. The data structure for the approximate solution and the number of decision points are the same as those for the one-broadcast strategy. The performance of the two-broadcast strategy will be discussed in Section 3.4.

### 3.3. Combined Strategy

A problem in the two-broadcast strategy is that the window for the second broadcast slot is chosen suboptimally when there are stations with search parameters smaller than the current broadcast value. Therefore, a better solution is to combine the one-broadcast and two-broadcast strategies and to make a decision in each iteration whether one broadcast or two broadcasts will be used. The objective is to minimize the expected number of future broadcasts. The procedure is a combination of the one-broadcast and two-broadcast procedures shown above and is illustrated by the following example shown in Figure 5. For the first iteration, the one-broadcast strategy is used with a window of (0, 0.51]. For the second iteration, the two-broadcast strategy is used with a window of (0, 0.33], and on the second broadcast, it becomes globally known that $x_4$ is the minimum.

The recurrence formulation to optimize the window in each iteration has two parts corresponding to the two strategies. Let $N_{AC}(a,b)$ be the the minimum expected number of broadcasts to isolate the minimum $x_i$ using a combined strategy and an approximate solution, given that all $x_i$s are in (a,U], and that at least one $x_i$ is in (a,b]. The recurrence formulation for the com-




(b) Second iteration

Figure 5. Example of the window search using the combined-broadcast strategy. The dashed lines indicate the portion of the interval being searched during the current broadcast slot. The current window is delimited by ( ].

bined strategy is expressed in terms of the one- and two-broadcast strategies. If

$$\text{one-broadcast} = 1 + \phi_A(a,b,w)\left|\int_a^w f_c(a,w,x_b)N_{AC}(a,x_b)dx_b\right| + \theta_A(a,b,w)N_{AC}(w,b),$$

and

$$\gamma(a,w) = \int_a^w f_c(a,w,x_{b1}) \times \left|1 + \phi_A(a,x_{b1},x_{b1})\int_a^{x_{b1}} f_c(a,x_{b1},x_{b2})N_{AC}(a,x_{b2})dx_{b2}\right| dx_{b1}$$

then

$$\text{two-broadcast} = 1 + \phi_A(a,b,w)\gamma(a,w) + \theta_A(a,b,w)N_{AC}(w,b),$$

and the recurrence is

$$N_{AC}(a,b) = \min_{a<w<b}\left|\min(\text{one-broadcast}, \text{two-broadcast})\right|. \qquad (10)$$

The number of decision points for the approximate solution is the same as that of the previous two strategies. The decision tree is the same as that shown in Figure 3, but the data structure differs slightly from that used for the previous cases because the strategy (one-broadcast or two-broadcast) must be stored in addition to the window for each iteration.

### 3.4. Simulation Results for the Approximate Distributed Search

The simulation results for the distributed window search using the three strategies are shown in Figure 6. The windows were generated using the equations derived in Sections 3.1 through 3.4. The broadcast parameters were generated from a uniform distribution in (0,1], and sufficient cases were simulated until a confidence interval of 0.95 was reached. The number of broadcast slots is bounded by 2.7 for the two-broadcast strategy, and by 2.6 for the one-broadcast and combined strategies. The two-broadcast strategy is not as good as the one-broadcast strategy because, although it can reduce the number of broadcast slots after the minimum has been identified, it uses suboptimal window choices for earlier broadcast slots. The combined strategy always chooses the one-broadcast strategy, so their results are identical. The overhead for the combined strategy is higher than that of the one-broadcast strategy because the strategy for each decision point must be stored in addition to the window choices. For these reasons, the one-broadcast strategy is superior to the other two.

The proposed scheme is practical as a result of the constant expected number of broadcast slots. The time required for a

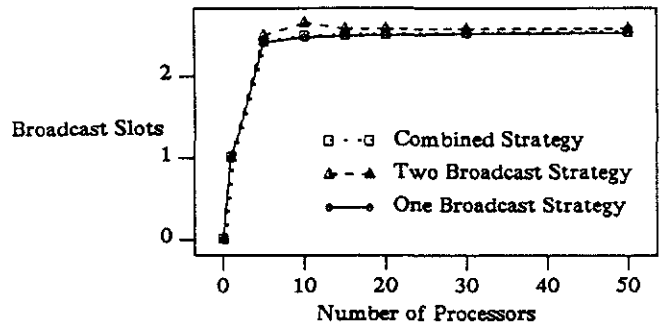



Figure 6. Results of the simulations for different window-search strategies.

contention slot is approximately 50 microseconds, and the time required to broadcast a search parameter may be estimated at approximately 100 microseconds. It follows that each broadcast slot would require on the order of 220 microseconds if 2.4 contention slots [14] were required to resolve contention. If it takes 120 microseconds to resolve contention and 100 microseconds to transmit a 1-Kbyte packet, then the overhead of each load balancing decision to identify the maximally and the minimally loaded processors is equivalent to transmitting 5.2 1-Kbyte packets.

## 4. IMPLEMENTATION OF LOAD BALANCING USING A DISTRIBUTED WINDOW SEARCH

The objective of load balancing is to evenly distribute jobs to processors, so the system load is balanced. This will minimize the time when a job is waiting for service at a processor while other processors are idle. An optimal load balancing strategy in a distributed system would try to distribute jobs to processors in a way similar to a multi-server system with a single queue, such that jobs never wait while processors are idle. This is not possible for a bus system because delay is incurred when jobs are sent across the network, and only one packet can be sent across the network at a time. For load balancing to be practical, it is necessary to efficiently identify the maximally and the minimally loaded processors and send a job from the maximally loaded processor to the minimally loaded processor if the turn-around time will be shorter. This allows the system to use the current status information to make an accurate load balancing decision.

There are four types of tasks that require the use of the network: regular message transfers, identifications of the maximally and the minimally loaded processors (max/min identification), job migrations, and result returns. The relative priorities of these tasks must be determined. Regular message transfer is assigned the highest priority since it is the original purpose of the network, and load balancing operations should not interfere with it. The priorities of the remaining tasks are determined by considering the relative overheads in terms of the additional total system delay (the sum of the delays of all jobs) incurred.

First, the relative priority between the identification of the $i^{th}$ maximally/minimally-loaded-processor pair and the migration of the job between the $(i-1)^{th}$ pair is determined. Two cases are considered. The first is when there are idle processors. If job migration is done first, then the job can begin execution immediately upon arrival at the destination processor, so its delay is increased by the time to send it across the network. In contrast, if the max/min identification is done first, then the same job will incur the delay of that operation as well. Clearly, migrating the job first is better. The second case is when there are no idle processors. The ordering of the tasks is not critical in this case, since migrating the job first does not immediately contribute to reducing the job delay as the job may not begin execution upon arrival at the destination. This result is true for any job migration and max/min identification, and performing the

migration first will result in a total delay equal to or smaller than performing the max/min identification first. As a result, job migration should have higher priority over max/min identification.

Next, the relative priority between result return and job migration is considered. When there are no idle processors, delay is added directly to the job waiting for result return. Since the job waiting for job migration will not be able to begin execution immediately upon arrival at the destination processor, result return should take precedence. When there are idle processors, both the delay for job migration and the delay for result return will add directly to the overall delay. Performing the result return first always results in performance equal to or better than performing job migration first. It is also easy to see that result return should always have precedence over max/min identification by a similar argument.

In summary, the priority orderings for tasks using the bus network is (1) regular message transfer, (2) result return, (3) job migration, and (4) max/min identification.

The load balancing strategy consists of two steps that are executed repeatedly. The first is to determine which of the current potential tasks has the highest priority, and the second step is to execute that task. Since the number of priority levels is small, the stations can resolve the highest priority level in a linear fashion. Due to the ordering of the priorities, only one job will ever be waiting for job migration, but potentially more than one may be waiting for result return.

All three types of contention-resolution protocols discussed in Section 2.1 can be used for performing load balancing operations with varying degrees of efficiency. Regardless of whether the decision is made in a centralized or a distributed manner, load information must be collected at decision locations. For an n-processor system, if the scheduler utilizes the message-passing subsystem for routing status information, then $(n-1)$ point-to-point transmissions of processor status information are required for a centralized decision, and n broadcasts of load information are required for a distributed decision. Status information can be propagated more efficiently with the distributed window search by using contention parameters that reflect processor loads. As estimated in the last section, an average of 5.2 1-Kbyte packet times are required to make a load balancing decision.

The values for the windows can be stored in a table as described in Section 3. The table space required to store the entire tree for varying processor loads is large, but acceptable results can be achieved by storing only the first four levels because contention is usually resolved in less than four iterations. Window boundaries in the pruned subtrees and windows for channel loads with no decision trees stored are obtained by interpolations. The storage space to store the decision trees for the one-broadcast strategy with loads varying from one to 100 processors is 5 Kbytes if windows are stored using 16 bits. The performance in this case is indistinguishable when more levels are used.

The average response time for a system using this load balancing strategy has been derived using an approximate queueing network and verified with simulation results [1]. The results show that at low traffic intensities, situations for which load balancing is beneficial do not occur frequently. When the load balancing interval is large, the effectiveness of load balancing is reduced as a result of the communication delay incurred. Consequently, load balancing has the greatest benefit at moderate traffic intensities and at load balancing intervals that are smaller than the average service time.

## 5. CONCLUSIONS

In this paper, we have presented an efficient technique for a distributed extremum search and a load balancing protocol using this technique. The search technique can be implemented on *existing* CSMA/CD networks at the applications level. This is important because it is typically not possible to make hardware modifications to existing networks. The maximum or the minimum of a set of numbers can be identified in a small bounded number of broadcast slots on the average. Since the search technique has a constant average behavior, a load balancing algorithm using this efficient search technique is feasible. Performance of the load balancing was found to be favorable at low to moderate traffic intensities, and load balancing intervals (the total time to perform max/min identification, job migration, and result transfer) that are small relative to the average service time. The results of the analysis can be used to determine a system size that allows load balancing to be beneficial.

## REFERENCES


[1] K. M. Baumgartner and B. W. Wah, "The Effects of Load Balancing on Response Time for Local Computer Systems with a Multiaccess Network," *Proc. Int'l. Conf. Communications*, June 1985, pp. 10.1.1-10.1.5.

[2] T. C. K. Chou and J. A. Abraham, "Load balancing in Distributed Systems," *IEEE Trans. Software Engineering*, Vol. SE-8, July 1982, pp. 401-412.

[3] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *IEEE Computer*, Nov. 1980, pp. 57-68.

[4] Y. C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. Comp.*, Vol. C-28, May 1979, pp. 354-361.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Engineering*, Vol. SE-12, May 1986, pp. 662-675.

[6] K. Hwang, W. Croft, B. W. Wah, F. A. Briggs, W. R. Simmons, and C. L. Coates, "A Unix-Based Local Computer Network With Load Balancing," *IEEE Computer*, Vol. 15 , Apr. 1982, pp. 55-66.

[7] A. Kratzer and D. Hammerstrom, "A Study of Load Leveling," *Proc. IEEE COMPCON*, Fall 1980, pp. 647-654.

[8] M. Livney and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Proceeding of the ACM Computer Network Performance Symposium*, 1982, pp. 47-55.

[9] R. M. Metcalfe and D. R. Boggs, "Ethernet : Distributed Packet Switching for Local Computer Networks," *CACM*, Vol. 19, July 1976, pp. 395-404.

[10] L. M. Ni and K. Hwang, "Optimal Load Balancing Strategies for a Multiple Processor System," *Proc. Int'l. Conf. Parallel Processing*, Aug. 1981, pp. 352-357.

[11] A. N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *Journal of the ACM*, Vol. 32, Apr. 1985, pp. 445-465.

[12] A. S. Tanenbaum, *Computer Networks*, Prentice Hall Incorporated, New Jersey, 1981.

[13] B. W. Wah and J. Y. Juang, "An Efficient Protocol For Load Balancing on CSMA/CD Networks," *Proc. Eighth Conference on Local Computer Networks*, Oct. 1983, pp. 55-61.

[14] B. W. Wah and J. Y. Juang, "Resource Scheduling for Local Computer Systems with a Multiaccess Network," *IEEE Trans. Comp.*, Dec. 1985, pp. 1144-1157, also in "Efficient Contention Resolution Protocols for Local Multiaccess Networks," U.S. Patent 4,630,264, Dec. 16, 1986.

[15] Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. Comp.*, Vol. C-34, Mar. 1985, pp. 204-217.