IEEE International Conference on Communications
Chicago, IL June 23-26, 1985

THE EFFECTS OF LOAD BALANCING ON RESPONSE TIME FOR LOCAL COMPUTER SYSTEMS WITH A MULTIACCESS NETWORK

Katherine M. Baumgartner and Benjamin W. Wah
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

ABSTRACT

Load balancing has been shown to be effective in reducing the average response time of jobs in a distributed computer system. In this paper, we derive the average response time due to load balancing for computers connected by a local CSMA/CD network. Based on a contention-resolution protocol that can identify the stations with the minimum and the maximum response times efficiently, jobs are sent from the maximally loaded processor to the minimally loaded processor. The average response time is analyzed by an approximate queuing network and is compared against simulation results. It is found that the benefits of load balancing are maximum at moderate traffic intensities and load balancing intervals that are short compared to job service time.

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* 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 communication 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. 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.

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 [ChK79] and Ni and Hwang [NiH81]. The major problem in centralized scheduling is 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 If this overhead is large, scheduling decisions are frequently based on inaccurate and outdated status information, which could be detrimental to performance.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* 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 deterministic or probabilistic [ChK79]. 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 processors according to a set of branching probabilities that are collected from previous experience. In the case that branching probabilities are derived from the service rates of processors, the strategy is called *proportional branching* [ChK79].

Deterministic load balancing strategies result in better performance; however, the overhead associated with implementing them is higher than that associated with probabilistic strategies. It was found that a probabilistic strategy for a single job class [NiH81] yielded better performance than a proportional branching strategy. An optimal probabilistic algorithm for multiple job classes was found to be easier to implement than deterministic strategies. However, probabilistic strategies are sometimes insensitive to dynamic changes in system load and may result in suboptimal performance.

Load balancing is implemented on the Purdue Engineering Computer Network (ECN) which is a system of computers connected by a hybrid of Ethernet and point-to-point links [Hwa82]. 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.

A common result of the previous studies on load balancing is that a network of computers with load balancing performs better than one without load balancing. Load balancing, however, may have the following side effects:

(1) Status information used in a deterministic decision must be readily available; otherwise, decisions based on outdated or inaccurate status information could degrade the performance.
(2) Load balancing increases network load which can impede message transmissions.

Consequently, load balancing is effective when the overheads of broadcasting status information and sending jobs and results are relatively small.

This study considers load balancing on a local computer system connected by a CSMA/CD multiaccess network. The objective is to determine *the effects of load balancing on the average response time*. These effects are a function of the *traffic intensity*, which is the ratio of the job arrival rate to the job service rate. As the traffic intensity changes, the decision to load balance must be adjusted. The results obtained can guide the design of better load balancing strategies.

There are six sections following this introduction. Section Two gives background about CSMA/CD networks and contention resolution. Section Three describes the protocol for making load balancing decisions in the network. Sections Four and Five explain the methods for the analyses and simulations

Research 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.

International Conference on Communications, 1985.

respectively. Section Six summarizes the results, and Seven draws conclusions.

2. CARRIER-SENSE-MULTIACCESS NETWORKS

Carrier-sense-multiaccess networks with *collision detection* (CSMA/CD networks) are a type of local area networks with packet switching and a bus topology [Tan81]. 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.

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 by allowing processors to transmit whenever they find the bus idle. When collisions occur due to simultaneous transmissions, processors stop transmitting, wait for some prescribed amount of time, and try again. The backoff algorithm of Ethernet is an example in this class. The disadvantage of collision-free protocols is 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 of the workload. The Virtual-Window Protocol proposed by Wah and Juang [WaJ83,JuW84] is an example of a limited-contention protocol.

The *Virtual-Window Protocol* shown in Figure 1 functions as follows. Stations wishing to transmit 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. For regular message transfers, each station has equal chance of being chosen for transmission, so the contention parameters can be generated from a uniform distribution between 0 and 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. This continues 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 in an efficient manner, so the number of contention slots is kept to a minimum. Analyses and simulations have shown that contention can be resolved in a minimum average of 2.4 contention slots independent of the number of contending stations and the distribution function of the contention parameters, provided that the parameters are independent and identically distributed.

All three of these types of contention-resolution protocols 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 Virtual-Window Protocol by using contention parameters that reflect processor loads. The distribution of response times at different stations can be used in the protocol to identify the stations with the maximum and the minimum response times efficiently.

```
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,
      $\hat{n}$ - estimated channel load,
      lb_window - lower bound of window to be searched
            (minimum is L),
      ub_window - upper bound of window to be searched
            (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_window := L;
      ub_window := U;
      $r_i$ := random(L,U);
      $\hat{n}$ := estimate();
      w := window( lb_window, ub_window, $\hat{n}$);
      contending := true;

      while (contending) do [
            if ($r_i \geq$ lb_window and $r_i \leq$ w) then [
                  transmit_signal();
                  /* test for unique station in the window */
                  state := detect();
                  if state = idle then
                  /* update lower bound of window */
                        lb_window := w
                  else if state = collision then
                  /* update upper bound of window */
                        ub_window := w
                  else /* successful isolation of minimum */
                        return(lb_window, ub_window);
                  w := window( lb_window, ub_window, $\hat{n}$) ]
            else
                  contending := false ]  /* stop contenting */
]
```

Figure 1. Procedure illustrating the basic steps executed in each station for contending the channel with a three-state collision-detection mechanism.

3. LOAD BALANCING PROTOCOL

Multiaccess networks have a broadcast bus topology that allows one job 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. To have the maximum reduction in response time, a job must be sent from the maximally loaded processor to the minimally loaded processor.

The load balancing protocol consists of two steps that are repeated continually. The set of stations with the highest-priority task is identified in the first step, then the actions to proceed with this task are performed in the second step. There are four tasks associated with load balancing using the Virtual-Window Protocol. They are regular message transfer, result return, job migration, and identification of the processors to be involved in a load balancing operation. These tasks are ordered according to the following priority levels.

(1) regular message transfer (highest priority);
(2) result return;
(3) job migration; and

(4) identification of the maximally and the minimally loaded processors (lowest priority).

Task 1 has the highest priority because regular message transfer is the primary function of the network, so load balancing should not degrade this communication. Result return has priority over job migration because any delay in returning results contributes to an increase in response time, while an earlier transmission of a job to a remote processor may not reduce the job response time unless the remote processor is idle. Task 3 has priority over Task 4 because a job must be completely sent before it can be processed, and the delay in completing a job transfer may tie up valuable buffer space unnecessarily and reduce the processor utilization. The Virtual-Window Protocol may be used for priority resolution, identification of the maximally and minimally loaded processors for a load balancing operation, and contention resolution for message, job, and result transfers.

4. RESPONSE TIME ANALYSIS FOR A LOAD BALANCED NETWORK

4.1. Assumptions

The analytical model is derived with the following assumptions:

(1) the stations are modeled as M/M/1 systems [Kle75] connected by a single bus;
(2) each station can accommodate at most ℓ jobs (including the one in service);
(3) the bus service time is exponentially distributed and each load balanced job is serviced by the bus only once;
(4) the network is only used for load balancing and not for regular message transfers;
(5) the times for identifying the maximally and the minimally loaded processors and for resolving contentions are negligible.

The finite queue limit was chosen to model a realistic implementation. The bus service time, or *load balancing interval*, takes into account network contention, so the additional complexity of modeling contention can be avoided. The last assumption was made because the Virtual-Window Protocol can resolve contentions in a small bounded amount of time on the average. If another protocol were used, or the decision made in a centralized manner, the time required for the load balancing operation would have to be adjusted. For most other protocols, this interval would be increased.

The analytical model is a two-processor system modeled as M/M/1/ℓ servers and connected by a bus modeled as an M/M/1/1 server. Load balancing involves only two processors at a time when using a bus network, the maximally and minimally loaded processors. In comparing a two-processor system with a system with more processors and the same bus service rate, the effect is to decrease the per-processor access to the bus. A system with more than two processors corresponds to a two-processor system with a longer load balancing interval. Therefore, a system with more than two processors can be approximated by a two-processor system with a proportionately smaller bus service rate.

4.2. Description of a System State

The state of each station is specified by the number of jobs in its queue. The queue limit imposed results in $\ell+1$ possible states for a station since it can have from zero to ℓ jobs queued (Assumption 2). The bus can be in any one of the n(n−1) states in load balancing a job from one station to another, or can be idle. The state of the system depends on the states of the n connected stations and the state of the network. A good representation would be a sorted sequence of (n+1) numbers, $(q_1, q_2, \ldots, q_n, b)$, in which the q_i (greater than or equal to q_{i+1}) corresponds to the number of queued jobs in one of the stations, and b indicates the state of the bus. Let $T(n,\ell)$ be the number of states for the n stations, each with a queue limit of ℓ, then $T(n,\ell) = \sum_{j=1} T(n-1,j)$, $n>1$, with boundary conditions $T(n,1)=1$ and $T(1,\ell)=\ell$. $T(n,\ell)$ equals the number of permutations of $\ell+1$ numbers whose sum is n, and has a closed form: $T(n,\ell) = \begin{pmatrix} n+\ell \\ n \end{pmatrix}$. As a result of not distinguishing the stations with equal number of queued jobs, the number of states for the bus can be reduced to $(\ell+1)^2+1$ if $n>(\ell+1)$. For a system with n=8 and $\ell=3$, T(8,3) is 165, the number of states for the bus is 17, and the total number of states is 9405. The number of states can be further reduced because some states accounted for in this model will have zero probability, specifically those states in which load balancing would be advantageous and the bus is idle. Additionally, when jobs are load balanced, they use buffer space at both the source and destination processors for the duration they are on the bus, hence reducing the queue limit at the source and destination stations by one when load balancing is carried out.

An additional reduction in the worst-case number of states is achieved when a two-processor model is used. For example, T(2,10) = 66, and the number of states is 3∗66=198. The state representation for a two-processor model is a triplet, (q_1, q_2, b), where q_1 and q_2 are the queue lengths of Processor 1 and Processor 2 respectively, and b is the state of the bus. The bus status is either 0 indicating that it is idle, or 1 indicating that the job on the bus has a destination of Processor 1, or 2 that indicating the job on the bus has a destination of Processor 2.

4.3. Computing the Average Response Time

The method used to find the average response time has three steps. The first is to use the state described above to determine the flow balance equations. The steady state probability vector, π, is found by solving the system of flow equations, and the boundary condition that the probabilities sum to one. If F is the matrix of coefficients of the flow-balance and boundary-conditions equations, then

$$F\pi = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \pi = F^{-1}\begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

Lastly, the response time can be obtained from the steady state probabilities.

Using the state described in the last section, there are five events that will cause a state transition. These events include an arrival to Processor One, an arrival to Processor Two, a departure from Processor One, a departure from Processor Two, and a completion of bus service. There may be fewer events for a given state. For example, there can be no arrivals when the queue is full, nor departures when the queue is empty, or bus completions when the bus is idle. Bus service (load balancing) is initiated when the bus is idle and the queue lengths are such that load balancing would be advantageous.

An example of a flow diagram for a system with a queue limit of two is shown is Figure 2. All permutations of states for the two stations are shown. For the system shown, the bus service time is very small compared to the processor service time, so jobs are load balanced whenever the queue lengths differ by two or more. Note that State (2, 1, 1) does not exist because the job on the bus uses buffer space at both the source and destination stations, hence reducing the queue limit in both stations to one. Also, an arrival to Queue 1 in State (1, 0, 0) results in State (1, 0, 2), as load balancing has been automatically initiated.

If the probability of State (q_1, q_2, b) is $P_{q_1,q_2,b}$ and the arrival rate to each processor is λ, the service rate for each processor is μ, and the bus service rate is μ_B, then the flow

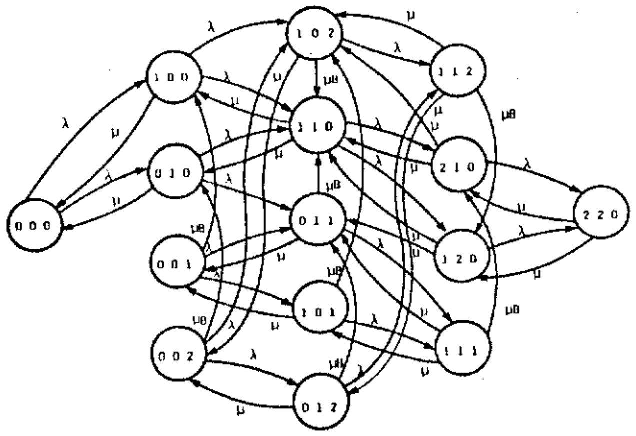

Figure 2. Flow diagram for a system with n=2 and ℓ = 2 (each state is a triplet with the first two numbers indicating the state of the stations and the third number indicating the state of the bus).

balance equation for a typical state with $q_1 > q_2$ is

$$(2\lambda + 2\mu + \mu_B)P_{q_1,q_2,2} = \lambda(P_{q_1-1,q_2,2} + P_{q_1,q_2-1,2}) \qquad (2)$$

$$+ \mu(P_{q_1+1,q_2,2} + P_{q_1,q_2+1,2}) + \mu_B(P_{q_1+1,q_2,1} + P_{q_1+1,q_2,2})$$

The coefficients in Eq. (2) are determined by the flows in and out of State $P_{q_1,q_2,b}$. Once F is found, π, the steady-state probability vector, is found by solving Eq. (1). The limiting factor in the analysis is the size of Matrix F, which has to be inverted.

The average response time is found using Little's formula, which states that the average number of jobs in the system is equal to the product of the average arrival rate and the average waiting time (response time) [Kle75]. The average number of jobs in the system is the sum of the number of jobs in each state weighted by the state probabilities.

$$E[\text{jobs in system}] = \sum_{\text{state } i} (\text{jobs in state } i) \times Pr(\text{state } i)$$

The average arrival rate found by taking a weighted average of the arrival rate to each state.

$$E[\text{arrival rate}] = \sum_{\text{state } i} (\text{arrival rate state } i) \times Pr(\text{state } i) \quad (3)$$

The response time is

$$E[\text{response time}] = \frac{E[\text{jobs in system}]}{E[\text{arrival rate}]}. \qquad (4)$$

5. SIMULATION OF A LOAD BALANCED NETWORK OF COMPUTERS

The first part of this section describes SMPL, a simulation language, and the second part describes the model used to simulate the network of computers with load balancing.

5.1. Description of the Simulation Language

The simulations were performed using SMPL, a simple portable simulation language, on a DEC VAX 11/780 computer [Mac80]. SMPL is an event driven language that was designed to model discrete-event systems. The language is a package of subroutines that handle queue management and statistic gathering. A SMPL simulation model consists of *facilities* and *tokens*, where facilities are static, and tokens are active and move from facility to facility. An event corresponds to a token reserving or releasing a facility.

To simulate a network of queues using SMPL, the model must first be initialized, which involves declaring facilities, initializing tokens, and scheduling initial events. After initialization, events that include the reservation and release of facilities are caused and scheduled until the simulation is completed. When a reservation of a facility is caused, the following sequence of actions occurs: (1) if the facility is busy (a token has previously reserved the facility) the token is queued; (2) otherwise, the facility is reserved and a release is scheduled at some future time depending on the service characteristics of the facility. The sequence of actions associated with the release of a facility are: (1) releasing the facility; (2) scheduling a reserve for a queued token at this facility; and (3) scheduling a reserve for the released token at its next facility if necessary.

5.2. Description of the Simulation Model

There is one difference between the assumptions made for the analysis and the simulations. For the analysis, job migration and result return for one job are accounted for in one bus service time (the load balancing interval). Two bus service times are used for the simulation, one for job migration and one for result return. These times are assumed to be exponentially distributed, each has a rate equal to one half of the load balancing interval. Other assumptions remain unchanged.

Each station connected to the bus is modeled by two facilities. The producer models users generating jobs with a Poisson distribution and average rate λ. The server models the actual service of jobs, which is Poisson distributed with rate μ. The bus is modeled by one facility with a queuing space of one (including the job in service). The service time at this facility corresponds to the time to send a job or result across the network, and is Poisson distributed with rate $\mu_B/2$.

The tokens are the active part of the model. The path that a token takes through the passive model depends on whether or not the job is involved in a load balancing operation. All jobs are generated at one of the stations. If a job is not involved in a load balancing operation, it is routed to the server at that station for local execution. Otherwise, it is routed to the bus for job migration, to a server at another station for remote execution, and to the bus again for result return when execution is completed. Before a job is sent for remote execution, buffer space is reserved for it at the destination's queue.

Load balancing is initiated whenever the bus becomes idle. At that point, if the load is not evenly distributed and, if load balancing is beneficial, a job is migrated from the maximally loaded processor to the minimally loaded processor. At the completion of remote execution, the results are returned. The priorities of transfer follow those described in Section 3.

6. RESULTS

Before presenting the results, some performance limitations will be considered. A system without load balancing consists of autonomous $M/M/1/\ell$ subsystems. On the other hand, the best possible system would be one with n parallel servers of average rate μ each, a single queue with limit $n \times \ell$ (including jobs in service), and a single arrival stream of jobs with rate $n \times \lambda$. It is expected that a system with load balancing will result in a smaller average response time than that of the independent $M/M/1/\ell$ system, but a larger average response time than that of the $M/M/n/n\ell$ system.

The queue limit imposed also creates a performance limitation. Jobs arriving at a queue with rate λ and finding the queue full have to be discarded. For Markovian queues, the effective arrival rate is given by Eq. (3).

The simulations and analyses were performed for a range of load balancing intervals and traffic intensities. A service rate of 0.1 jobs per second was used, and the arrival rate was varied to change the traffic intensity. The queue limit, ℓ, was set to be five. The average load balancing interval was varied from a small value up to one at which no load balancing occurred. The results are summarized in Figures 3, 4, and 5.

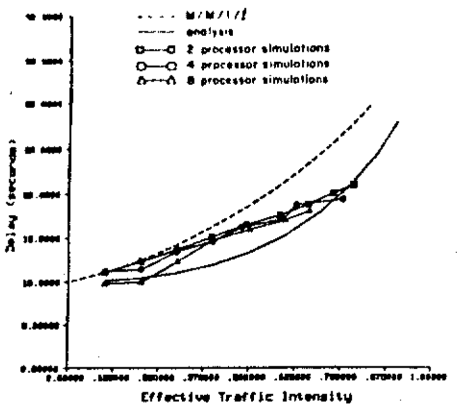

Figure 3. Delay for system with $\ell=5$, $\mu=0.1$ at a small load balancing interval for n=2, n=4, and n=8.

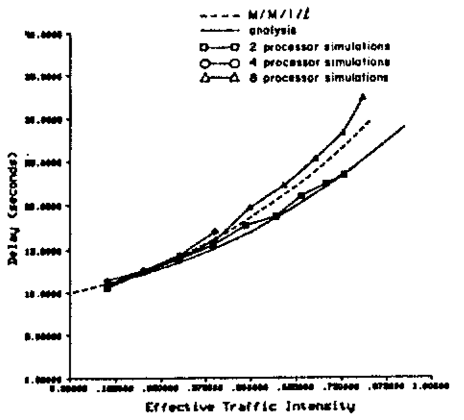

Figure 4. Delay for system with $\ell=5$, $\mu=0.1$ at a moderate load balancing interval for n=2, n=4, and n=8.

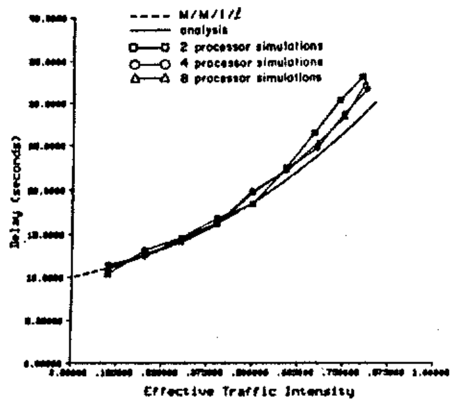

Figure 5. Delay for system with $\ell=5$, $\mu=0.1$ at a large load balancing interval for n=2, n=4, and n=8.

These figures show that the delays associated with the analyses are slightly lower than those with the simulations for the eight-processor case. The reason for this discrepancy is due to the irregularities in the load balancing interval for the simulations. In the simulations, the bus is used to transmit jobs and results that are exponentially distributed, and load balancing is initiated when the bus is free. The interval between two load balancing operations is, therefore, irregular. When there is a long interval between two load balancing operations, there is more imbalance in the workload which causes the average response time to increase. However, when the load balancing interval is short, the workload may be already well balanced, and the benefits of load balancing are small. It may be possible to more accurately analyze systems with more than two processors by using a hyper-exponential distribution for the bus service time instead of an exponential service distribution. That would introduce larger variances into the distribution of the service time of the bus and more accurately predict the delay.

These figures also show that load balancing has the greatest benefit at moderate traffic intensities and at load balancing intervals that are smaller than the average service time. 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 delay incurred.

7. CONCLUSIONS

In this paper, we have presented an approximate queuing model to analyze the reduction in average response time due to load balancing for a local computer system connected by a multiaccess bus. Due to an effective contention-resolution protocol, the overhead of propagating status information for the load balancing protocol is negligible. We found that load balancing is beneficial when the traffic intensity is moderate, and the load balancing interval is of the same magnitude or smaller than the average service time. At high traffic intensities, instabilities in performance can result with load balancing.

Our study can lead to the design of a good load balancing strategy. From our results, we found that load balancing should be used sparingly at high traffic intensities. A threshold that requires a minimum savings as a function of load can be set to reduce the amount of load balancing at high loads. Our results can also be applied to determine the effects of connecting the stations by multiple busses and to allow multiple jobs to be load balanced simultaneously.

REFERENCES

[ChK79] Y. C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. on Comput.*, Vol. C-28, May 1979, pp. 334-361.

[Hwa82] K. Hwang, et al., "A Unix-Based Local Computer Network With Load Balancing," *IEEE Computer*, Vol. 15, April 1982, pp. 55-66.

[JuW84] J. Y. Juang and B. W. Wah, "Unified Window Protocol for Local Multiaccess Networks," *Proc. of 3'rd Annual Joint Conf. of IEEE Computer and Communications Societies*, April 1984, pp. 23.

[Kle75] L. Kleinrock, *Queuing Systems Volume 1: Theory*, John Wiley, New York, 1975.

[Mac80] M. H. MacDougall, *SMPL - A Simple Portable Simulation Language*, Technical Report, Amdahl Corporation, 1980.

[NiH81] L. M. Ni and K. Hwang, "Optimal Load Balancing Strategies for a Multiple Processor System," *Proc. of 10'th Int'l Conf. on Parallel Processing*, August 1981, pp. 352-357.

[Tan81] A. S. Tanenbaum, *Computer Networks*, Prentice Hall Inc., New Jersey, 1981.

[WaJ83] B. W. Wah and J. Y. Juang, "Load Balancing on Local Multiaccess Networks," *Proc. of 8'th Conf. on Local Computer Networks*, October 1983, pp. 55-61.