Analysis of a Class of Distributed Queues with Applications*

Mart L. Molle[†]

ABSTRACT

Recently we have developed a class of media access control algorithms for different types of Local Area Networks. A common feature of these LAN algorithms is that they represent various strategies by which the processors in the LAN can simulate the availability of a centralized packet transport facility, but whose service incorporates a particular type of changeover time known as 'moving server' overhead. First we describe the operation of moving server systems in general, for both First-Come–First-Served and Head-of-the-Line orders of service, together with an approach for their delay analysis in which we transform the moving server queueing system into a conventional queueing system having proportional waiting times. Then we describe how the various LAN algorithms may be obtained from the ideal moving server system, and how a significant component of their performance characteristics is determined by the performance characteristics of that ideal system. Finally, we evaluate the compatibility of such LAN algorithms with separable queueing network models of distributed systems by computing the interdeparture time distribution for M/M/1 in the presence of moving server overhead. Although it is not exponential, except in the limits of low server utilization or low overhead, the interdeparture time distribution is a weighted sum of exponential terms with a coefficient of variation not much smaller than unity. Thus, we conjecture that a service centre with moving server overhead could be used to represent one of these LAN algorithms in a product form queueing network model of a distributed system without introducing significant approximation errors.

Technical Report #88-14

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

May 1988

* This research was supported by the Natural Sciences and Engineering Research Council of Canada, under grant A5517.

[†] This work was carried out while the author was on Sabbatical leave from the Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4.

Analysis of a Class of Distributed Queues with Applications*

Mart L. Molle[†]

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A4

Abstract — **Recently we have developed a class of media access control algorithms for different types of Local Area Networks. A common feature of these LAN algorithms is that they represent various strategies by which the processors in the LAN can simulate the availability of a centralized packet transport facility, but whose service incorporates a particular type of changeover time known as 'moving server' overhead. First we describe the operation of moving server systems in general, for both First-Come–First-Served and Head-of-the-Line orders of service, together with an approach for their delay analysis in which we transform the moving server queueing system into a conventional queueing system having proportional waiting times. Then we describe how the various LAN algorithms may be obtained from the ideal moving server system, and how a significant component of their performance characteristics is determined by the performance characteristics of that ideal system. Finally, we evaluate the compatibility of such LAN algorithms with separable queueing network models of distributed systems by computing the interdeparture time distribution for M/M/1 in the presence of moving server overhead. Although it is not exponential, except in the limits of low server utilization or low overhead, the interdeparture time distribution is a weighted sum of exponential terms with a coefficient of variation not much smaller than unity. Thus, we conjecture that a service centre with moving server overhead could be used to represent one of these LAN algorithms in a product form queueing network model of a distributed system without introducing significant approximation errors.**

I. Introduction

Distributed computer systems present a significant challenge to performance analysts, in terms of finding both good designs and also good analytical models for such systems. One reason for the apparent difficulty of modelling distributed systems it that, almost by definition, some type of Local Area Network (LAN) is employed as a central element of the design, and all communications among the various processing elements of the system must take place over the LAN. Consider, for example, that Sun Microsystems Inc. has trademarked the phrase "The Network *Is* the Computer" for use in its advertising.

* This research was supported by the Natural Sciences and Engineering Research Council of Canada, under grant A5517.

[†] This work was carried out while the author was on Sabbatical leave at the Department of Information and Computer Science, University of California Irvine, Irvine, CA 92717

Unfortunately, the performance characteristics of a LAN depends critically on the details of an internal, very low level distributed algorithm (sometimes called the *media access algorithm*), which determines when each processor can send each packet over the LAN's communication channel. These media access algorithms must resolve the 'interference' between the packets queued at different processors for transmission over the common channel, which they accomplish by various "tricks" based on careful synchronization of the actions of different processors, cyclic polling of individual processors, randomization, etc. Because of their complexity and dependence on detailed timing information for correctness, accurate analytical models of LANs operating in isolation have often been difficult to obtain. Furthermore, even if the traffic handling characteristics of the LAN in isolation *are* known, they often turn out to be unsuitable for integrating the LAN into a larger scale performance model as the service centre that processes use to travel from one processor to another.

If the peculiarities in the performance characteristics of LAN access algorithms could be eliminated, then the task of integrating LANs into larger scale performance models would be dramatically simplified. Thus, in some sense the ideal LAN would behave exactly like an conventional queueing system—a "distributed queue" of jobs located at various processors and all waiting for delivery to some other processor. Thus, it has been our goal to develop LAN access algorithms that emulate this distributed queueing ideal as closely as possible, given the fact that the customers are dispersed in space. That is, we must be prepared to pay some "price" for this spatial dispersion, in terms of time and/or bandwidth (to learn about the spatial information), and possibly some "scheduling errors" (when we choose to "guess" rather than to pay). Our strategy for achieving this goal has been to create algorithms that correspond to a distributed simulation of an idealized distributed queueing system that we call a *moving server queue.* It is interesting to note that our algorithms reduce to well known single server queueing systems with common service orderings (such as FCFS, HOL, etc.) in the limit as the dispersion of the customers (and hence the overhead in the simulation) vanishes.

II. On 'Moving Server' Queueing Systems and their Analysis

In a moving server queueing system, a server traverses an infinite tour along which he encounters customers requiring service. For now, let us assume that the server, when he is not busy serving a customer, moves along the tour at one of two different rates depending upon his position with respect to some constantly advancing reference point. When the server is caught up with the reference point, he advances in step with it; when he is behind (because he had to stop to serve a customer) he speeds up by a factor of η, $\eta > 1$, in order to catch up. In the sequel, we shall normalize rates so that the speed of the reference point is unity.

Notice that moving server systems differ from conventional queueing systems of the type M/M/1, M/G/1, etc. Instead of asking the customers to come to the server (and join a central queue to obtain his service), we require the server to come to the customers, charging ourselves walk time overhead in the process. Furthermore, although our model resembles both cyclic multiqueue systems [1] (used to model polling systems or token ring LANs) and the classical 'machine repairman' problem in Operations Research, the novelty on our approach is in having the server traverse an *open* tour rather than a closed cycle. As a consequence, the customer arrival process resembles that found in an open queueing network model [2] in the sense that the distribution of the number of customers (or, equivalently, their aggregate total service requirements) encountered by the moving server in one segment of the tour is unaffected by the time he spent traversing any earlier segments of the tour. This is quite different from cyclic multi-queue systems, where the number of customers encountered at each queue (or per unit distance in the continuous case [3]) depends on the time that has elapsed since his last visit to this queue.

The random variable of particular interest in the analysis of moving server systems is the difference between the time the advancing reference point encounters a customer requiring service (which we will call the *arrival time*) and the time the server encounters that customer. We shall refer to this random variable as the *moving server delay,* $\tilde{W}$. In this section we will show that, by a suitable transformation of the moving server system, a conventional single server queueing system is obtained in which the waiting time is proportional to the moving server delay. Thus, we are free to apply the entire body of queueing theoretic literature in attempting to solve for the performance of a moving server queueing system that corresponds to some particular LAN algorithm of interest.

A. First-Come—First-Served Order of Service

Consider the sequence of customer waiting times, $W^{(1)}$, $W^{(2)}, \ldots,$ in a conventional first-come—first-served single server queueing system. Assume that $\mathbf{x}^{(k)}$ and $A^{(k)}$ are the service time for the kth customer and the interarrival time between the $(k-1)$st and kth customers, respectively, the distributions of which are unimportant for our present discussion. In particular, we make *no* assumption about independence, nor do we assume particular interarrival or service time distributions. Clearly

$$W^{(k)} = \max\{W^{(k-1)} + \mathbf{x}^{(k-1)} - A^{(k)}, 0\}, \tag{1}$$

since the kth customer enters service at either the $(k-1)$st customer's departure time or at his own arrival time, whichever occurs last.

Now consider a corresponding sequence of moving server delays $\tilde{W}^{(1)}$, $\tilde{W}^{(2)}, \ldots,$ in a moving server queueing system. Clearly $\tilde{W}^{(k)} = 0$ if and only if the moving server has caught up to the reference point when he encounters the kth customer. Otherwise, the kth customer must wait until the $(k-1)$st customer's departure time (so the server can start moving again), and then for enough additional time for the server, walking at rate η, to reach the kth customer's arrival time, namely $A^{(k)}/\eta$. Thus

$$\tilde{W}^{(k)} = \max\{\tilde{W}^{(k-1)} + \mathbf{x}^{(k-1)} - A^{(k)} + A^{(k)}/\eta, 0\}. \tag{2}$$

Comparing Eqs. (1) and (2), we see that waiting times in a moving server system contain an extra term corresponding to "walk time" overhead, where a customer's entry into service may be delayed even though the server is *not* offering service to any customer. However, unlike most models of queueing systems with "changeover time" overhead, ours is *proportional* to the corresponding customer's interarrival time, rather than some independent random variable.

Define

$$\beta = \frac{\eta}{\eta - 1} \tag{3}$$

to be a constant that depends on the moving server's speedup factor, η. Multiplying both sides of Eq. (2) by β and recognizing that $-A^{(k)} + A^{(k)}/\eta = -A^{(k)}/\beta$, we have that

$$\beta \tilde{W}^{(k)} = \max\{\beta \tilde{W}^{(k-1)} + \beta \mathbf{x}^{(k-1)} - A^{(k)}, 0\}. \tag{4}$$

Notice that after a change of variable from $\beta \tilde{W}^{(\cdot)}$ to $W^{(\cdot)}$, Eq. (4) has exactly the same form as Eq. (1), and thus represents the relation between successive customer waiting times in a conventional queueing system, which we will call the *synthetic queueing system.* The transformation from moving server system to synthetic system preserves the sequence of interarrival times $A^{(1)}, A^{(2)}, \cdots$ while substituting a proportional increase from $\mathbf{x}^{(k)}$ to $\beta \mathbf{x}^{(k)}$, respectively, in each customer's service time in place of the walk time overhead. Since $W^{(1)} = \tilde{W}^{(1)} \triangleq 0$, the observation above implies that

$$\tilde{W}^{(k)} = W^{(k)}/\beta \tag{5}$$

must hold for all k. And since this correspondence holds for each customer taken individually, it must hold for the mean and distribution of the waiting time in the two systems as well. Hence Eq. (5) allows us to compute the statistics of waiting time in the moving server queueing system from those of the synthetic queueing system.

The increase in service times that results from transforming a moving server system into a corresponding synthetic queueing system has an interesting physical interpretation. Recall that while the server stops to offer service to some customer, he is being *left further behind* by the reference point at rate unity; whenever the server is moving at rate η, he is *gaining* on the reference point at the rate of $\eta-1$. Thus, to maintain his relative position with respect to the reference point over the long run, the system must satisfy a "global balance" condition, which states that the server must spend $1/(\eta-1)$ time units moving at the accelerated rate for every time unit spent serving a customer. One way to accomplish this would be to impose a "local balance" condition, where we define the kth customer's service time to be

$$\tilde{\mathbf{x}}^{(k)} = \mathbf{x}^{(k)} + \frac{\mathbf{x}^{(k)}}{\eta - 1} = \beta \mathbf{x}^{(k)},$$

the sum of his actual service time together with enough walk time overhead for the server to regain his former position with respect to the reference point. In the synthetic system, we assume that a customer's entire service time is processed without interruption, while in the moving server system, the component corresponding to walk time overhead is preempted if the server encounters subsequent customers in his walk.

B. Head-of-the-Line Priority Service

Now consider the generalization of the problem to a moving server system with P Head-of-the-Line (HOL) priority classes. In this case, we assume that the system contains P separate moving servers, each serving customers from a single class. However, we allow only one of the servers at a time to serve customers: whenever *one* of the servers encounters a customer belonging to its respective priority class along the tour, *all* P servers stop until the service is complete. In addition, the movements of the servers are constrained so that the servers from classes p and below remain stopped (and thus they are unable to reach any new lower priority customers) until the class $p+1$ server, moving at rate η_{p+1}, has caught up with the reference point. Thus, we say that the class p server is *busy* whenever he is stopped (either because some customer is being served, or because the class $p+1$ server is still busy) or walking at rate η_p, and that he is *idle* otherwise. Notice that from the point of view of the class p server, a customer (from any priority class) whose actual service time is $\mathbf{x}$ represents $\beta_p \mathbf{x}$ units of 'work',

$$\beta_p \overset{\Delta}{=} \prod_{i=p}^{P} \frac{\eta_i}{\eta_i - 1}, \tag{6}$$

out of which he stops (as if he were offering service to the customer) for $\beta_{p+1}\mathbf{x}$ time units, and is moving at rate η_p for the rest.

As for the non-priority case above, we solve for the class p waiting time in the moving server HOL queueing system with the aid of a synthetic queueing system. In this case, however, we will employ a different synthetic system for each priority class, since the *same* unit of actual service time represents β_P units of work to the class P server, versus β_{P-1} units of work to the class $P-1$ server, etc. The p th synthetic system is a conventional HOL priority queueing system (i.e., without any walk time overhead) that serves the same set of customer arrivals as the moving server system. As in the case of FCFS service above, we substitute a proportional increase in each customer's service time by a factor of β_p in place of the walk time overhead due to the moving servers from classes p, $p+1, \ldots, P$. In addition, we replace each lower class customer's arrival time by its *pseudoarrival time* to account for the walk time overhead due to the moving servers from the lower classes $p-1, \ldots, 1$. The pseudoarrival time for a lower class customer is defined as the position of the class p moving server upon its entry into service (since, by definition, the class p server must be moving in step with the reference point when a lower priority customer enters service).

Notice that in both the synthetic and moving server systems, every customer from every priority class whose [pseudo]arrival time is less than the arrival time for a 'tagged' class p customer enters service ahead of it. Thus, recognizing that a lower priority customer's pseudoarrival time is at least as large as its true arrival time, and using our previous results for FCFS ordering, it follows that for each class p customer, the synthetic system overestimates by a factor of β_p/β_{p+1} the component of his waiting time due to customers in the system upon his arrival.

It remains to find the contribution to a 'tagged' class p customer's waiting time due to customers from strictly higher priority classes, but who arrived after him. But this is simply the remainder of a generalized busy period, initiated by the FCFS component of the 'tagged' customer's waiting time, in which only customers from classes $p+1, \ldots, P$ are served. If we now restrict our attention to the M/G/1 queue with HOL priorities,[1] then the distribution of such a generalized busy period is well known, e.g., [4, 5]. It is important to note, however, that in the synthetic system a fraction $1/\eta_p$ of this additional waiting time component represents walk time for the class p server, while he is traversing a segment of the tour *beyond* the point where the 'tagged' class p customer arrived, so we must use $\beta_{p+1}\mathbf{x}_i$, $i=p+1, \ldots, P$, as the service times of higher priority customers in this calculation. Thus, for the case

[1] Notice that in our previous discussion, we substituted pseudoarrival times in place of true arrival times for each customer belonging to a strictly lower priority class. Although it is unlikely that these pseudoarrival times form a Poisson process, it is clear from [4, section 8.6] that the Poisson property for customers belonging to classes $p-1, \ldots, 1$ is not used in the derivation of the waiting time distribution for class p customers. In particular, *every* customer belonging to classes $p-1$ and below (along with some of the customers belonging to classes p and above) initiates a generalized busy period where only customers belonging to classes p and above are served, the analysis of which—and the probability that a 'tagged' class p arrival encounters one of them—depends only on the mean arrival rate of customers belonging to classes $p-1$ and below.

of the moving server M/G/1 HOL priority queue, we have, in principle at least, demonstrated that the distribution of waiting times can be obtained from the corresponding results for a conventional M/G/1 HOL priority queue after a few straightforward manipulations.

In particular, the results above allow us to write down by inspection the mean class p waiting time for an M/G/1 HOL queueing system with moving server overhead, using the well-known results for the conventional M/G/1 HOL queue with class j arrival rate λ_j and service time $\mathbf{x}_j$ [4, 5]. We obtain

$$\tilde{W}_p = \frac{\beta_{p+1} \cdot \beta_p \, W_0}{\left(1 - \beta_p \sum_{j=p}^{P} \overline{\mathbf{x}_j} \, \lambda_j\right)\left(1 - \beta_{p+1} \sum_{j=p+1}^{P} \overline{\mathbf{x}_j} \, \lambda_j\right)} , \tag{7}$$

where

$$W_0 = \frac{1}{2} \sum_{k=1}^{P} \lambda_k \, \overline{X_k^{\,2}} \tag{8}$$

is the mean residual life of the service time for the customer (if any) in service at his arrival, the ratio of the numerator to the left-hand factor in the denominator represents the component of his mean delay due to customers in the queue on his arrival, and the ratio of that component to the right-hand factor in the denominator represents the mean duration of the delay busy period that ends with his entry into service.

III. Application to LAN Media Access Algorithms

In this section, we will show how to receive credit towards the "price" we must pay for the spatial dispersion of the packets in a LAN, for the walk time overhead in a moving server system. That is, we will show how to construct several media access algorithms for different LAN types that are both implementable and have good performance properties, and which operate as "distributed simulations" of the "ideal" moving server systems described in the previous section. To conserve space, we shall concentrate on explaining how the mappings between moving server systems and LAN algorithms are accomplished, and on some general principles by which the delay performance of the resulting LAN algorithms may be obtained. The interested reader is referred to the accompanying references for the details of the analysis for each algorithm.

To simulate the operation of a moving server system on a LAN, we must define the *tour* through time—and possibly also "space", i.e., the LAN— that will be followed by both the reference point and moving server. In defining the tour, we must remember that each of the processors in the LAN must be able to calculate, at least approximately, the times at which both the reference point and the moving server would have encountered each of its packets along the tour. Having done all this for a specific system, we obtain a LAN media access algorithm by defining what each of those processors must do to simulate the local part of the moving server system. In general, the actions of each processor consist of: (i) tagging each of its own newly arriving packets with the point on the tour at which the reference point would have encountered that packet; (ii) tracking the progress of the moving server along the tour; and (iii) executing the "service algorithm" that the moving server would have used upon reaching that point on the tour, resulting in the [eventual] transmission of the packet.

Notice that the three steps listed above are disjoint, and that every packet must pass through each of those steps [and possibly more than once, in the case of some collision based algorithms] before arriving at its destination processor. Thus the steps may be used to partition the time in system for a packet into distinct "phases". The first phase, terminating with the arrival of the reference point, is simple to compute exactly, since the speed at which the reference point walks along the tour is constant. The time spent in the second phase is just the waiting time in the underlying moving server queueing system. The last phase, in which the moving server executes its "service algorithm", is determined by the characteristics of the service algorithm, given a particular distribution of packets waiting at that point on the tour, and it does not depend on the time at which the server reaches that point. Furthermore, if we are careful about our definition of the tour and our selection of the "service algorithm", it may be possible to arrange for these components of a packet's time in system to be mutually independent random variables, thus simplifying the delay analysis.

A. Carrier Sense Multiple Access (CSMA/CD)

Virtual time CSMA [6, 7] is an efficient, moving server based alternative to the well-known non-persistent, 1-persistent and p-persistent CSMA protocols [8] for Ethernet-like broadcast LANs [9]. In contrast to 1-persistent CSMA (which is a "greedy" algorithm, where each processor sends its packets as soon as the medium is sensed to be idle—even though this greatly increases the likelihood of collisions under load) and non-persistent CSMA (which is an "altruistic" algorithm, where requests to transmit packets while the medium is sensed busy are ignored—even though a given packet must endure many random rescheduling delays before it is transmitted), virtual time CSMA places all the packets in a common queue, which it then attempts to serve in global FCFS order.

Here we let the time axis serve as our tour, along which the reference point is assumed to walk at constant speed. In addition to the above "asynchronous" version of the algorithm, a "synchronous" sliding window type algorithm is obtained by assuming that the reference point moves at the rate of one discrete "window" of size w every w time units. In either case, the first phase of the simulation is trivial for each processor to execute, since each packet is simply tagged with its own generation time, rounded up to the next window boundary in the case of the synchronous algorithm. Similarly, the last phase is also trivial: whenever a processor finds that the moving server's position on the tour matches the tag for one of its queued packets, it simply transmits the packet. In the event that a collision occurs, the packet is rescheduled in the usual way for CSMA algorithms, by computing a random backoff delay and adding it to the "tag" value for that packet.

The middle step in the simulation, i.e., tracking the progress of the moving server along the tour, is done in the following way. For the synchronous version of the algorithm, it is easy for all processors to move their local copies of the moving server in step with one another. At the start of the each "slot", each processor advances its server to the next window, transmitting one of its packets if it finds one there, and then waits for a time determined by the channel outcome (i.e., 'idle'/'success'/'collision') in the current slot before advancing to the next window. Like all "simplified window" algorithms, the rules described above introduce "rest periods" whenever the server is in step with the reference point, because the duration of a idle slot is much less than the length of a window. However, this inefficiency is easily avoided by letting the reference point advance in steps of size a, the network propagation time, every a

time units in this case (i.e., the size of the next window is the minimum of w and the lag between the server and the reference point moving in continuous time).

The asynchronous version of the algorithm is handled by giving each processor autonomous control of its local copy of the moving server, which is assumed to start moving at each transition from busy to idle channel, and to stop moving at each transition from idle to busy. Since all processors can monitor the state of the channel (i.e., 'idle'/'busy') at no cost, there is no need for them to exchange positional information about the server. However, the times at which different processors start and stop the motion of their local copies of the moving server can differ by as much as the channel propagation time, a. The speed of the server is piecewise constant, so a simple 'event driven' simulation of the server motion is sufficient. Each time a packet arrives to a processor, it is either transmitted immediately, if the moving server is in step with the reference point, or else tagged with its arrival time and placed in a local queue. At each transition from busy to idle channel, each processor computes the time at which the server would either encounter its next queued packet, if it has one, or else catch up to the reference point, based on the contents of its local queue. If the channel remains idle until that time, the processor initiates the appropriate action, consisting of either transmitting a queued packet (and updating the position of the server correspondingly) or indicating that the server is moving in step with the reference point for the duration of this idle period.

Notice that in both the synchronous and asynchronous versions, the order in which packets from different processors are granted their first transmission attempt is always close to global FCFS. However collisions are still unavoidable with this algorithm whenever $a > 0$. In the synchronous algorithm, we offer "bulk service" to all packets arriving in a window of size w, $w = \eta a$, while in the asynchronous algorithm, the servers at the other processors continue to move (and at the accelerated rate η, if the server is not in step with the reference point) for time a after the first processor starts transmitting its packet. However, we have the option of reducing η so as to trade an increase in channel access delay (together with a corresponding reduction in capacity) for a reduction in the collision probability.

Under the popular—but rarely trusted—Poisson total traffic assumption, it is easy to obtain estimates for the mean time in system for a packet under both the synchronous and asynchronous versions of virtual time CSMA. The time up to the arrival of the reference point is either zero or the residual life of a window, for the asynchronous and synchronous versions, respectively. The difference in the arrival times between the reference point and the moving server corresponds to the waiting time in an M/G/1 queue with moving server overhead (and which operates in discrete time [10] in the case of the synchronous algorithm). An estimate for the remaining time, due to the "service algorithm", may be found by assuming that the average number of attempts required is given by the reciprocal of the probability of success under the Poisson traffic assumption, and that the mean time between successive attempts is the sum of the length of a collision and the mean backoff delay. In [7], the resulting estimates were found to have very good agreement with detailed simulation studies of the synchronous algorithm in spite of our use of the Poisson traffic assumption. We attribute the accuracy of our model to the low collision probability, and hence the low frequency with which the backoff algorithm must be invoked. However, our estimates were of only marginal use with the asynchronous algorithm because of the problem of accounting for the state dependence (i.e., whether or not the server is in step with the reference point) in the collision probability, which manifests itself in terms of both the number of customers found per unit distance along the time line and the service time for those customers. In [11], however, a refinement of the model to account for these state dependencies was found to give accurate estimates for the asynchronous protocol.

B. CSMA/CD with Message Based Priority Classes

Given the discussion in section II.B about moving server systems with HOL priority classes, it should come as no surprise that there is an elegant way to incorporate HOL priority classes into virtual time CSMA [12, 11]. Thus in PVT-CSMA, each processor that generates packets belonging to classes $p, \ldots, P$ simply tracks the current positions of the moving servers from classes $p, \ldots, P$ in response to the channel history in the manner described above. However, it is perhaps not so obvious that unlike most other proposals for adding message based priority classes to CSMA, ours comes at essentially no "cost" since, with proper choices for $\{\eta_p\}$, *no* additional channel overhead is introduced, the collision probability under the Poisson traffic model remains the same, and the overall state-independent mean delay is preserved. All we have done is to rearrange the walk time overhead that was already present in virtual time CSMA so as to permute the order in which packets from different priority classes are transmitted.

An example of the effects of imposing two HOL priority classes is shown in Figure 1. Assuming that each priority class generates half the total traffic, we let $\eta_2 = 2\eta$ and $\eta_1 = 2\eta - 1$ for the two class algorithm. Thus, since

$$\beta \mathbf{x} = \frac{\eta}{\eta - 1} \mathbf{x} = \frac{(2\eta)}{(2\eta - 1) - 1} \frac{(2\eta - 1)}{(2\eta) - 1} \mathbf{x} = \prod_{j=1}^{2} \frac{\eta_j}{\eta_j - 1} \mathbf{x} = \beta_1 \mathbf{x} ,$$

total amount of work (including 'moving server' overhead) for each packet is the same in both systems, and hence the length of the busy period must be preserved. Furthermore, consider the probability that service to some "customer" corresponds to a successful transmission in the two systems. In the non-priority system, there are two cases to consider. First, if the customer's entry into service takes place during a busy period, then we assume that the servers at the remaining processors continue moving at rate η for another a time units, yielding a probability of success of $e^{-a\eta G}$ under the Poisson traffic model. Otherwise the entry into service must have taken place during an idle period, in which case the rest of the processors continue moving at rate 1 for another a time units to give a probability of success of of e^{-aG}. In the two priority system, we have three cases to consider. First, if only the class 2 server was moving at rate η_2, then the probability of success will be $e^{-a\, 2\eta\, G/2} = e^{-a\eta G}$. Next, if the class 2 server was moving at speed 1 and the class 2 server was moving at speed η_1, then the probability of success will be $e^{-a\,(1 + 2\eta - 1)\, G/2} = e^{-a\eta G}$. And finally, if the entry into service took place during an idle period, then the probability of success will be $e^{-a\,(G/2 + G/2)} = e^{-aG}$. Since busy periods occupy the same proportion of the time axis in both systems by construction, and the respective conditional probability of success, given that the transmission took place within or outside a busy period, are identical, both the throughput and required number of transmissions for each class is unchanged.

In [11], we obtained estimates for the mean time in system for the asynchronous version of PVT-CSMA under the Poisson traffic assumption, as described above. As for the non-priority case, we found that good estimates for the moving server component of the delay are obtainable from Eqs. (7–8) by careful substitution of state dependent arrival rate and service time distributions. However, we have as yet been unable to obtain a good model for the synchronous version of the algorithm, in which windows for each priority class may advance at the start of each slot. This is because, although the interarrival times (between successive window boundaries) and service times (i.e., slot lengths) are all discrete, the elementary time unit in the discrete arrival process is both state dependent (i.e., "Is the class p server moving in step with the reference point now?") and class dependent (since the window sizes will in general vary between classes). Thus, finding an estimate of the waiting time synchronous PVT-CSMA, that is more accurate than simply approximating the synchronous algorithm by an asynchronous one, remains an open problem.

C. LAN-Based Tree Conflict Resolution Algorithms

Instead of the CSMA approach, where the moving server simply continues on his walk after each collision, leaving the backoff algorithm to reassign each of the affected packets to a random point further along the tour, the processors in an Ethernet-like broadcast LAN could apply a more complicated "service algorithm" to each window, such as a Tree Conflict Resolution Algorithm (TCRA) of the Capetanakis-Tsybakov-Mikhailov type [13]. Here "service" to a window corresponds to a *conflict resolution interval* (CRI), consisting of a sequence of one or more discrete algorithmic steps from a recursive "divide and conquer" group testing algorithm that terminates when all processors are sure that every packet belonging to the current window must have been transmitted successfully. This is easy to accomplish by having all processors execute in lock-step a synchronous *conflict resolution algorithm,* corresponding to the preorder traversal of a d-ary *conflict resolution tree,* $d \geq 2$. Initially, all packets belonging to the current window are assigned to the root node of the conflict resolution tree, the node from which the traversal begins. Thereafter, at the each step (i.e., slot) during the CRI, all packets assigned to the current node are transmitted by their respective processor. As in the case above for synchronous CSMA algorithms, the time at which to execute the next step (if any) of the CRI depends on the channel outcome ('idle'/'success'/'collision') during the current slot. Furthermore, if that channel outcome is a collision, all processors know that at least 2 packets were assigned to the current node, and extend the tree to include its d successors, and each of those packets is reassigned at random to one of the successors by its processor. Similarly, if the channel outcome at the current step is a non-collision, then all processors know that current node requires no successors. The CRI ends when there are no more nodes to "test" (which is easy for all processors to determine since the remaining number of "untested" nodes increases by $d-1$ at each collision and decreases by 1 at each non-collision).

In [14], we used the approach above to find solve for the distribution of the time in system for packets in such a LAN in the case of constant length windows (which results in the "rest periods" described above for synchronous virtual time CSMA) and Poisson arrivals per window, for the d-ary Capetanakis-type conflict resolution algorithm above. Here the delay until the arrival of the reference point is simply the residual life of a window as before. Furthermore the distribution of CRI lengths and of the time from the start of a CRI until the departure of a 'tagged' packet may be computed (but with some effort!) by taking advantage of the recursive structure of the algorithms. The remaining term, corresponding to the moving server delay, can be found as the waiting time in a discrete time M/G/1 moving server queue in which the (equally spaced) window boundaries correspond to possible customer arrivals, 'customers' correspond to windows that delay the progress of the server (because they contain at least one packet, and which are separated by a geometric number of points corresponding to empty windows), and the service times for those customers are given by the conditional CRI length distribution, assuming that at least one packet was contained in the window. The results from this M/G/1 analysis are exact whenever each of the possible service times (i.e., CRI lengths) we obtain for the synthetic system is an integer multiple of the the elementary time unit in our discrete time queueing system (which is determined by the window size). And in those cases where this discrete embedding condition is not satisfied, we have found [14] that the results still give us an excellent approximation to the mean delay.

Before leaving this example algorithm, we wish to point out that task of solving for its delay was made easier because of the following "separability" condition. Notice that under the Poisson traffic model, the moving server delay must be independent of (any combination of) the residual life of the window at its arrival and the time within its own CRI of its departure, since the former depends only on the arrivals from a Poisson source *before* the start of the current window, while the other two depend on the arrivals *within* a disjoint interval, namely its own window. Furthermore, if the random assignment of packets from a collision node to its d successors is based on random "coin tosses" rather than relative arrival times, then the residual life of the window at a packet's arrival and the time of its departure within its own CRI will also be independent. Thus the distribution of the time in system factors into a product form, corresponding to the phases in the LAN access algorithm. (Our earlier results for virtual time CSMA have a product form as well, but only because we ignored the dependencies between the phases.)

D. Token Rings

So far we have presented examples of moving server based algorithms as applied to Ethernet-like broadcast channels. However, we have shown that the basic concept is equally applicable to token rings [15]. In this case, however, the unidirectional cyclic structure of the LAN leads us to modify our definition of the tour to include a spatial component. In particular, we assume that the reference point travels around the ring at constant speed, labelling each new packet he encounters with his current tour iteration number. The actions of the "server" (i.e., the idle token) in this system consist of walking along the same tour, stopping to transmit packets *labelled with his current tour iteration number* as he encounters them. As before, we assume that the server, when not serving a customer, moves η times faster than the reference point unless he is in step with the reference point.

We call such a system a token ring with *helical window service* because the motion of the reference point along the surface of an infinitely long hollow cylinder representing the evolution of the ring in space and time would inscribe helical threads of constant pitch, which partition time into constant length *windows* from the perspective of an individual processor on the ring. It should be clear that the difference between the times at which the reference point and the idle token arrive at any point on the ring for the k th time can be modelled as a moving server queueing system where "windows" receive FCFS service at the rate of one window per visit by the moving server.

Unlike our previous algorithms, in this system our moving server represents a physical entity (i.e., the idle token), and his motion along the tour actually requires that physical entity to move physically around the ring—and at one of two different speeds! Thus, the method by which the processors track the motions of the reference point and the server along the tour requires some explanation. First, because the reference point moves at constant speed, the time between successive visits to the same processor is deterministic. Thus each processor in isolation can determine the times at which the reference point will make its k th visit to the processor, based on its relative position on the ring with respect to the time origin of tour and the speed with which the reference point moves along the tour. The situation with the server is more complicated, since the time for a signal to travel once around the entire ring is fixed by the characteristics of the ring hardware, and is given by the sum of the interface delays at each processor and the propagation times between each pair of processors around the cycle. By convention this hardware characteristic of the ring is called the *walk time* for the ring. Below we will denote the ratio of the walk time to the mean packet transmission time by a , since it plays a corresponding role to the propagation time in an Ethernet-like broadcast LAN in terms of measuring the "cost" of dispersing the packets among the stations.

Since the operation of a token ring requires the idle token (i.e., our moving server) to circulate around the ring, it should be clear that the *spatial* component of the maximum speed of the moving server along the tour is limited by the walk time of the ring to one revolution every a time units. However, *temporal* component of his maximum speed has no such physical constraints, and may be adjusted arbitrarily by changing the pitch of the "threads" inscribed on the helix by the reference point. In other words, the projection of the server onto the time axis can be moving at rate η if the temporal length of the window is ηa . However, since the temporal component of the speed of the reference point must be unity, by definition, we are left with the problem of slowing down the idle token (i.e., a physical bit pattern) so that it travels η times more slowly than usual whenever the moving server is in step with the reference point. This can be accomplished in the following way. First, inactive stations (i.e., those that have crashed or been taken off line) can simply allow the data on the ring, including idle tokens, to pass by unhindered, in the usual way. Second, the active stations follow the usual token ring protocol with the following exception. Upon detecting the arrival of the idle token on the incoming channel τ time units ahead of the reference point, say, the station initiates a "rest period" on the outgoing channel, of duration τ, thus delaying the arrival of the idle token until the completion of the corresponding window. Thereafter, the station continues in the usual way, either by releasing the idle token, since it now knows there won't be any packets to send in the current window, or by transmitting a busy token, the packet(s) scheduled for transmission in the current window separated by busy tokens, and finally an idle token.

The implementation of the "rest periods" on the outgoing channel can be done as follows. If tokens are represented as reserved bit strings [16], then the rest periods can be created using variable length tokens. Thus, instead of using 1^8 and $1^7 0$ to represent idle and busy tokens, respectively, we can use $1^{7+j}01$ and $1^{7+j}00$, where j is any non-negative integer. Notice that the minimum length token is one bit longer than before (or that bit stuffing, to prevent the appearance of a token in a packet, must be done one bit earlier) and that the delay at each ring interface must be increased to *two* bit times. Alternatively, it is possible to create tokens that are of the same duration as a single bit time using violations of the standard Manchester encoding for data bits [17]. In this case, since the complete token fits within the one bit delay in the station's ring interface, it is a simple matter for the station to "remove" the idle token and output ordinary data bits for the duration of the rest period.

Now consider the service characteristics for such a system from the point of view of an arriving packet. First it must wait to receive a tour iteration number at the next visit of the reference point, which is easily done by 'rounding up' the packet's generation time to the next window boundary. Next, this packet, along with any others that were generated at the same processor during that same window, forms a "train of packets" at the tail of a queue of such "trains", the j th of which will be transmitted after another j arrivals of the idle token. (Notice that some of "trains" ahead of the one containing the tagged packet might contain several packets—or possibly even no packets—so our helical window access rule is clearly different from ordinary, gated and exhaustive service.) And finally, when the moving server arrives on the correct tour iteration, the "train" of packets is transmitted. At this stage, our tagged packet may experience some *queueing delay,* due to packets that are ahead of it in the "train", followed by its own transmission.

In comparing the helical window token ring described above with our CSMA and TCRA based algorithms, we see that the influence of the parameter a (which, as you may recall, is a measure of the "cost" of traffic dispersion in our LAN) on the performance of our algorithm is somewhat different. Since token rings are inherently collision free systems, we are not forced into a tradeoff between moving server delay and the collision probability. Instead, we are given tradeoff between fairness and light traffic response time on one hand (if we choose a small value for η) versus reduced delays—and higher capacity—under heavy load (if we choose a large one). By fairness, we mean that the algorithm guarantees that two packets from different processors will transmitted in chronological order if their generation times differ by at least ηa (and hence that it *approximates* network-wide FCFS transmission of packets up to the resolution of the window size). Thus, we see that the algorithm reduces to a conventional FCFS single server queue as $a \rightarrow 0$ if we select $\eta \approx 1/\sqrt{a}$.

In [15], we obtained exact results for the distribution of the time in system for symmetric (in terms of both the location and traffic generation rate for each processor) systems with Poisson arrivals and a general, independent packet length distribution.[1] We were able to do this because the Poisson traffic assumption, together with the *a priori* partitioning of the traffic according to constant sized windows, is again sufficient to ensure independence between the moving server delay and the (sum of) the residual life of the window at phase (i) and the queueing delay at phase (iii). Furthermore, since the service time for each packet is an i.i.d. random variable by assumption, we see that once again we have obtained the distribution of the time in system as a product of independent factors, namely the sum of the window and queueing delays, the moving server delay, and the service times. And unlike other polling and token passing systems, these results are remarkably easy to obtain.

IV. Do these Algorithms Fit into Models of Distributed Systems?

We have now described a set of four LAN algorithms, all of which represent different strategies for implementing distributed moving server queueing systems. All possess nice properties from both the performance and ease of analysis points of view, such as high throughput, reasonable delay, low variance of delay and "almost" global FCFS (or HOL) service order. Furthermore, due to the nice structure of these algorithms, we have been able to obtain *exact* results for the *distribution* of delay for two of the algorithms, along with simple and yet remarkably accurate approximations to the mean delay for the others. How much can we expect of this class of algorithms, in terms of their suitability for modelling distributed systems?

[1] Again, in the finite population model, for our results to be exact we must assume that the allowable packet lengths are discrete, and chosen so that the token always catches up to the reference point at the location of one of the processors.

Undoubtedly, the best we could hope for would be to discover that the characteristics of one (or more!) of our LAN algorithms satisfies the restrictions for service centres in a separable (i.e., "product form") queueing network model. Thus, to test this hypothesis, we now derive the interdeparture time distribution for our "ideal" prototype LAN algorithm, namely a FCFS queue with moving server overhead. Since it is already known that FCFS queues *without* moving server overhead *already* fail to satisfy these restrictions unless the service times are exponentially distributed (and with a common mean, in the case of multiple class systems), we can restrict our efforts to the M/M/1 queue with moving server overhead.

Recall that in a moving server queue, there is some "changeover time" between the departure of one customer and the entry of the next customer into service to account for the motion of the server along that segment of his tour. Below, we define the interdeparture time in the moving server system to be the sum of the changeover time following the departure of some customer and the *following* customer's actual service time. Although the service time distribution for M/M/1 is trivial, the calculation of these interdeparture times is more complicated because the server may catch up to the reference point before encountering the $(k+1)$st customer, and thus be forced to reduce his speed from η to unity over the remainder of the segment.

To calculate the distribution of this changeover time, we recognize that a fraction $1/\eta$ of the *total* time in system for the k th customer in the synthetic system is actually walk time overhead, which is used to advance the moving server from the position of his own arrival to the point where the busy period would end if a $(k+1)$st customer were not encountered first. This result follows because: (i) we have FCFS scheduling, and so in both the moving server and synthetic systems the k th customer must remain in the system during all of the (actual) service for himself and all customers who arrived ahead of him in the busy period (but not for any service to customers who arrived after him); and also (ii) the amount by which the departure time of the k th customer in the synthetic system overestimates his departure time in the moving server system is exactly the time required for the server to move, at rate η, across the segment of the tour that begins at the position of his arrival and ends at the point where he would have left in the synthetic system—a distance equal to his system time in the synthetic system. Thus, it follows that the server can walk at rate η over the entire distance between the k th and $(k+1)$st arrivals if the interarrival time leading up to the position of the $(k+1)$st arrival is no farther along the tour than the time the k th customer spends in the synthetic system.

Let us now condition on a system time of y, say, and compute the conditional Laplace transform for the changeover time distribution, given that busy period will end (and the speed of the server must drop to unity) if the interarrival time is greater than y. Since the interarrival time distribution is exponential by assumption, we have

$$a(x) = x \; e^{-\lambda x} \; , \qquad A^*(s) = \frac{\lambda}{s+\lambda} \; .$$

Furthermore, it is obvious that as long as $x \leq y$, the changeover time will be x/η; otherwise the it will be $y/\eta + (x-y)$, since the server walks along the remainder of the segment at unit speed. Combining these two cases, we obtain after some manipulations that

$$C^*(s \mid y) = A^*(s/\eta) + e^{-y(\lambda + s/\eta)}[A^*(s) - A^*(s/\eta)] \, .$$

Notice that this expression even has an intuitive justification, namely that the distribution of the changeover time is exponential with rate $\lambda\eta$ unless it exceeds y, which happens with probability $e^{-\lambda y}$, in which case it is the sum of the constant y/η and an exponential tail, but at rate λ instead of $\lambda\eta$. Unconditioning on the system time, y, we have

$$C^*(s) = \int_{y=0}^{\infty} C^*(s \mid y) \, dP[y]$$

$$= \int_{y=0}^{\infty} \Big(A^*(s/\eta) + e^{-y(\lambda+s/\eta)}[A^*(s) - A^*(s/\eta)] \Big) \, dP[y] \, . \tag{9}$$

Since that all but one of the factors of $C^*(s \mid y)$ in Eq. (9) are independent of y, and recognizing the remaining term as the transform of the system time in the synthetic system, evaluated at the point $\lambda + s/\eta$, Eq. (9) may be rewritten as

$$C^*(s) = A^*(s/\eta) + [A^*(s) - A^*(s/\eta)] \, S^*(\lambda + s/\eta) \, .$$

Since we have defined the interdeparture time as the sum of the changeover time (which we obtained by conditioning on the service time of the *previous* customer) and the service time for the *following* customer, and these two service times are independent in M/M/1, the interdeparture time distribution must be convolution of the changeover and service time distributions, i.e., a product in the transform domain. Thus,

$$D^*(s) = B^*(s) \left(A^*(s/\eta) + [A^*(s) - A^*(s/\eta)] \, S^*(\lambda + s/\eta) \right) . \tag{10}$$

Making substitutions into Eq. (10) based on $B^*(s)=\mu/(s+\mu)$, $A^*(s)=\lambda/(s+\lambda)$, and, since we need the system time for the synthetic system, $S^*(s)=\mu(1-\beta\rho)/(\beta s+(1-\beta\rho)$ [18, Eq. (5.117)], Eq. (10) simplifies dramatically to yield

$$D^*(s) = \frac{\lambda}{s+\lambda} \left(\frac{\mu}{\mu + \beta s/\eta} \; \frac{\beta s + \mu}{s + \mu} \right) . \tag{11}$$

Taking the first derivative with respect to s evaluated at $s=0$, we can find the mean interdeparture time is $1/\lambda$, as expected. Differentiating our the expression for $\frac{d}{ds}D^*(s)$ once more, we obtain

$$\overline{d^2} = \frac{2}{\lambda^2} \left(1 - 2(\beta - 1)\rho^2 \right) \leq \frac{2}{\lambda^2}$$

and

$$C_d = \sqrt{1 - 2(\beta-1)\rho^2} \leq 1 \tag{12}$$

as the second moment and coefficient of variation of the interdeparture time distribution. Although we are slightly disappointed to see that these results are in general below the corresponding values for the exponential distribution, we see from Figure 2 that they are quite close. (The boundary curve, namely

$$C_d \le \sqrt{1 - 2(\eta-1)/\eta^2}\,, \qquad \eta \ge 1, \tag{13}$$

corresponds to the stability condition, i.e., $\beta\rho \le 1$, for the synthetic queueing system.)

Notice that pairs curves in Figure 2, intersecting the boundary at $\rho = 1/\beta$ and $\rho = 1-1/\beta$ respectively, appear to have the same shape up to a multiplicative scaling in the horizontal direction. This suggests that normalizing with respect to $\beta\rho$ may lead to some simplifications, and, indeed, it is easy to verify that Eq. (11) is *invariant* under the substitution $(\beta, \rho) \to (\eta, \rho\beta/\eta)$. Notice that $\beta=\eta=2$ represents a fixed point under this substitution rule, and that it also gives us the minimum possible value for Eq. (13), namely $1/\sqrt{2}$. That $\eta=2$ should play a key role in these results is not surprising, since it represents the boundary between values of η where service and changeover time overhead, respectively, become the dominant components of a busy period.

The interdeparture time distribution is easily obtained by inverting the transform in Eq. (11), to give

$$d(t) = \begin{cases} \alpha_1 \lambda e^{-\lambda t} + \alpha_2 \mu e^{-\mu t} + \alpha_3 (\beta-1)\mu e^{-(\beta-1)\mu t} & \beta \ne 2 \\ \dfrac{\lambda(1-2\rho)}{(1-\rho)^2}\left[e^{-\lambda t} - e^{-\mu t}\right] + \dfrac{\lambda\, \mu\, t\, e^{-\mu t}}{1-\rho} & \beta = 2 \end{cases} \tag{14}$$

where

$$\alpha_1 = \frac{1-\beta\rho}{(1-\rho)\,(1-(\beta-1)\rho)}$$

$$\alpha_2 = \frac{(\beta-1)\,\rho}{(\beta-2)\,(1-\rho)}$$

$$\alpha_3 = \frac{(\beta-1)\,\rho}{(\beta-2)\,(1-(\beta-1)\rho)}\,.$$

Notice that for $\beta \ne 2$, Eq. (14) is a convex sum of exponential terms, and that as $\eta \to \infty$ (and hence β approaches unity from above) or as $\rho \to 0$ (and hence $\alpha_1 \to 1$), the distribution converges to an exponential with rate λ, as expected. Furthermore, in the most extreme case, namely $\beta = 2$ and $\beta\rho = 1$, it is easy to see that the departure process reduces to a two stage Erlangian distribution, as it must be since the server alternates between walking and service—each of which is an exponential stage with rate 2λ.

Figure 3 shows the effect of β (or, equivalently, of η due to the invariant substitution rule) on the interdeparture time distribution for two representative values of the normalized utilization. In all cases, we have normalized the arrival rate to unity, obtaining different values of ρ by varying μ. In part (a), we vary both β and and ρ so as to preserve the product $\beta\rho = 0.5$. The resulting graphs show that there is little sensitivity to β for moving server systems operating at 50% of their available capacity, except for the small values of t. In part (b), we show the worst-case situation for each system. That is, for each value of β, we show $d(t)$ when the system is operating at 100% of available capacity. In this case, we see greater sensitivity to the parameter β, but even here the part of the departure distribution beyond the mean takes the form of an exponential tail.

V. Summary and Conclusions

We have described a general class of "distributed" queueing systems that include changeover time overhead to account for the time required for a server, capable of moving at a finite maximum rate, to reach the positions of successive customers arranged along an infinite tour. We have shown that the analysis of the waiting time in such a system can be reduced to the analysis of the waiting time in a conventional queueing system, by transforming the moving server system into a synthetic queueing system (with altered service times for each customer but no changeover time overhead) having proportional waiting times.

The relevance of these results to the study of Local Area Networks was demonstrated by surveying several media access algorithms for different types of LANs that mimic the operation of such a distributed moving server queue, and for which a major component of the delay analysis is readily obtainable from the study of the underlying moving server system. We believe that these results are also relevant to the larger problem of building (and obtaining performance models for) large scale distributed systems, because we expect the "nice" performance characteristics, when studied in isolation, of these LAN algorithms (in comparison to those of other LAN algorithms) to lead to better performing and (more easily analysable) systems. As evidence to support this belief, we showed that the underlying moving server queueing system, upon which these LAN algorithms are based, is close to satisfying the necessary restrictions for a service centre to useable in a product form queueing network model.

The work reported on here can be extended in a variety of of directions. These include further work on the analysis of LAN algorithms in isolation, such as more general models for the arrival and/or service time distributions or the inclusion of various performance enhancements in the algorithms themselves. However, it seems equally important to go more deeply into the study of integrating good LAN models into large scale models of distributed systems, for example, by testing our claims of the suitability of our LAN algorithms against detailed experiments.

References

[1] H. Takagi, *Analysis of Polling Systems,* MIT Press, Cambridge, Mass. (1986).

[2] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis using Queueing Network Models,* Prentice-Hall, Englewood Cliffs (1984).

[3] E. G. Coffman, Jr. and E. N. Gilbert, "A Continuous Polling System with Constant Service Times," *IEEE Transactions on Information Theory* **IT-32**(4), pp.584-591 (July 1986). (Preprint: AT&T Bell Laboratories, Murray Hill, New Jersey 07974, 1984).

[4] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling,* Addison-Wesley (1967).

[5] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications,* Wiley-Interscience, New York (1976).

[6] M. L. Molle, "Unifications and Extensions of the Multiple Access Communications Problem," CSD Report No. 810730 (UCLA-ENG-8118), UCLA Computer Science Department (July 1981). Ph.D. dissertation.

[7] M. L. Molle and L. Kleinrock, "Virtual Time CSMA: Why Two Clocks are Better than One," *IEEE Transactions on Communications* **COM-33**(9), pp.919-933 (September 1985).

[8] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part I — Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics," *IEEE Transactions on Communications* **COM-23**(12), pp.1400-1416 (December 1975).

[9] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM* **19**(7) (July 1976).

[10] J. J. Hunter, *Mathematical Techniques of Applied Probability, Vol. 2 — Discrete Time Models: Techniques and Applications,* Academic Press, New York (1983).

[11] M. L. Molle, *Prioritized Virtual Time CSMA: Head-of-the-Line Priorities Without Added Overhead.* submitted to *IEEE Transactions on Communications.*

[12] M. L. Molle and M.-W. Wu, "Prioritized Virtual Time CSMA: Head-of-the-Line Priorities Without Channel Overhead," *IEEE International Conference on Communications*, pp.35.4.1-35.4.6 (June 1985).

[13] J. L. Massey, "Collision-Resolution Algorithms and Random-Access Communications," in *Multi-User Communications*, ed. G. Longo, Springer-Verlag, New York (1981).

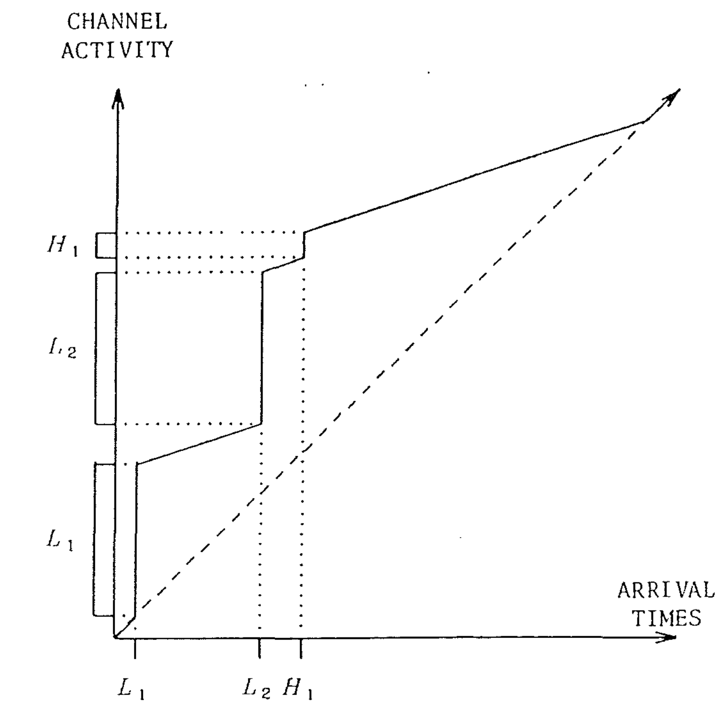
[14] G. C. Polyzos and M. L. Molle, "Delay Analysis of a Window Tree Conflict Resolution Algorithm in a Local Area Network Environment," *ACM SIGMETRICS '87 Conference on Measurement and Modeling of Computer Systems,*, pp.234-241 (May 1987).

[15] F. R. Kschischang and M. L. Molle, *The Helical Window Token Ring.* submitted to *IEEE Transactions on Information Theory.*

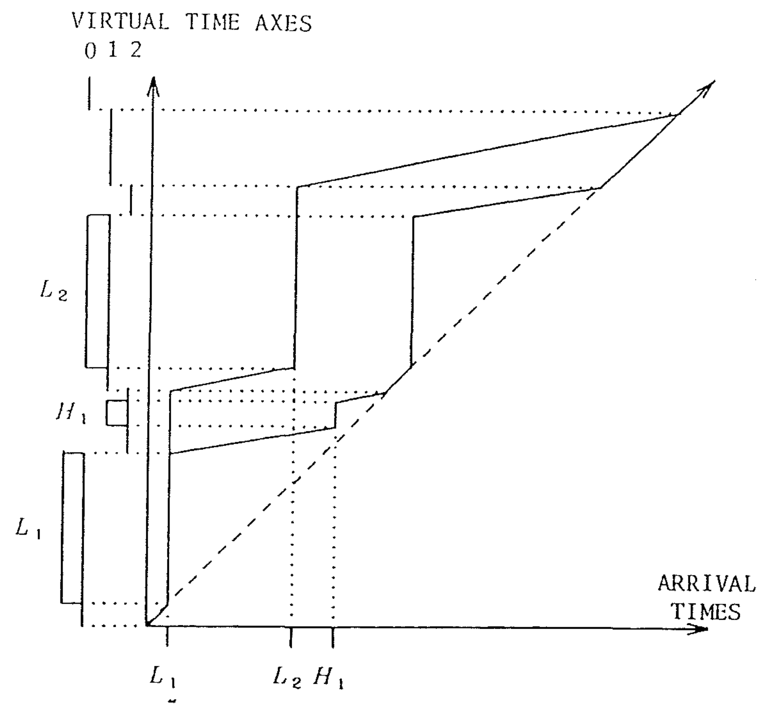
[16] A.. S. Tanenbaum, *Computer Networks,* Prentice-Hall, Inc., Englewood Cliffs, N. J. (1981).

[17] J. L. Hammond and P. J. P. O'Reilly, *Performance Analysis of Local Computer Networks,* Addison-Wesley, Reading, Mass. (1986).

[18] L. Kleinrock, *Queueing Systems, Volume 1: Theory,* Wiley-Interscience, New York (1975).



(a) Mapping of arrival times to transmission times in virtual time CSMA without priority classes



(a) Mapping of arrival times to transmission times in PVT-CSMA two priority classes

FIGURE 1: REORDERING OF PACKET TRANSMISSIONS BY PRIORITY CLASS IN PVT-CSMA

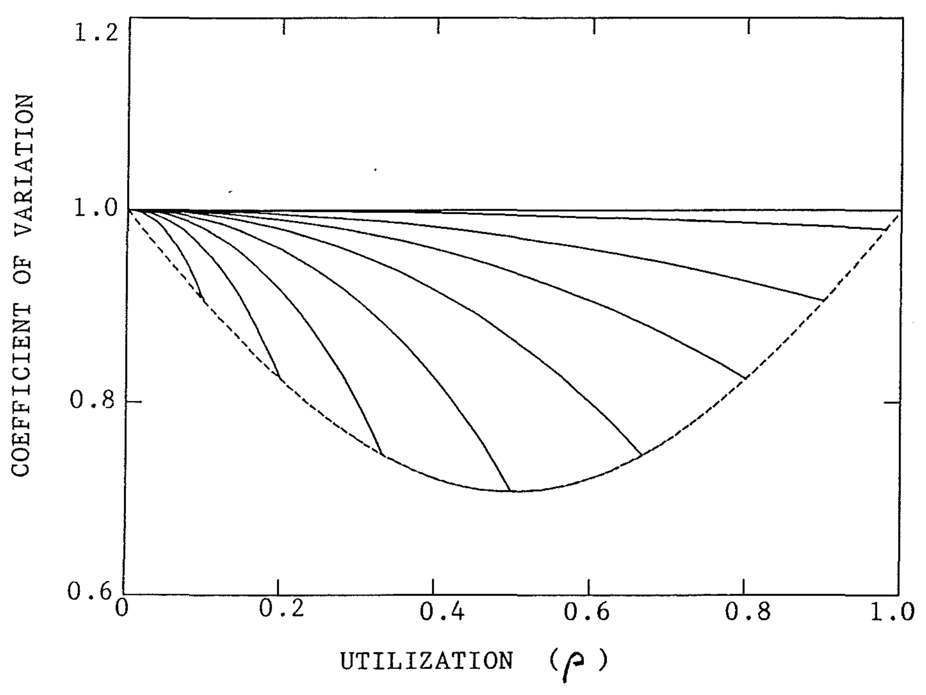
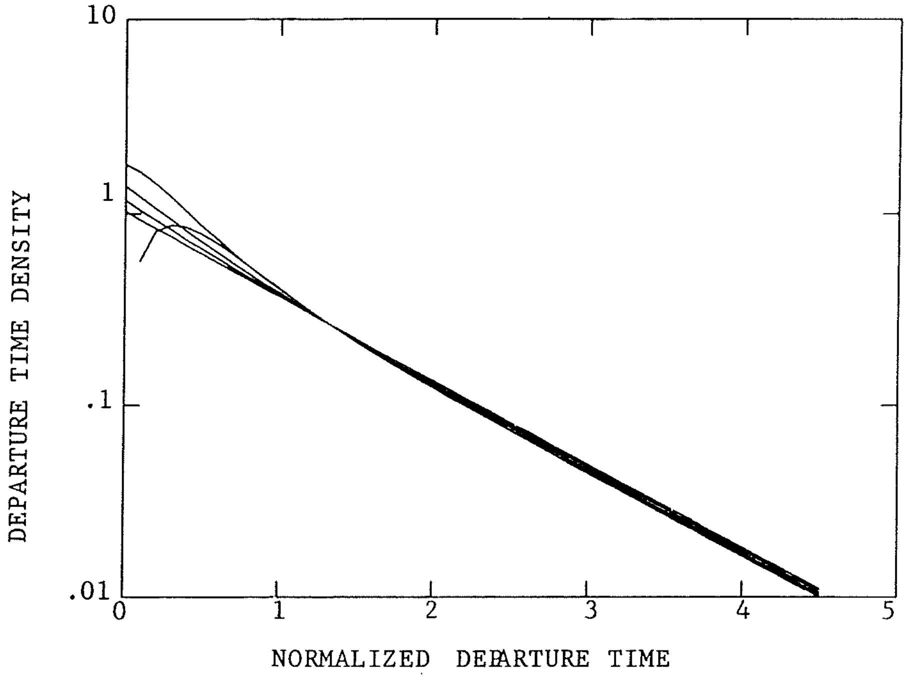
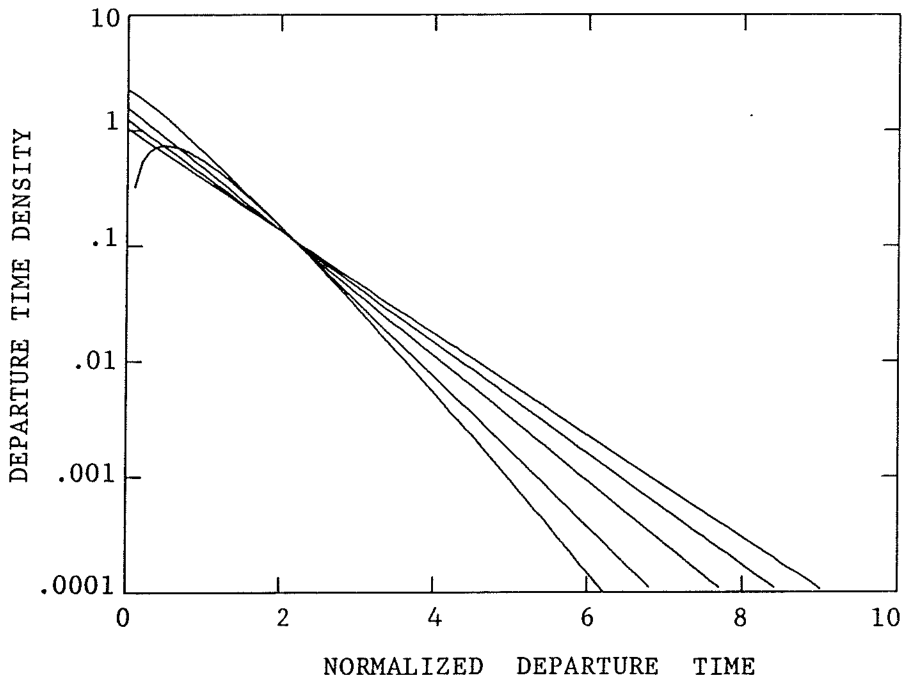


FIGURE 2: Effect of on the Ceofficient of Variation for the Interdeparture Time Distribution (n = 1.1111, 1.25, 1.5, 2, 3, 5, 10, 50)



(a) NORMALIZED UTILIZATION $\beta\rho$= 0.5



(b) NORMALIZED UTILIZATION $\beta\rho$= 1.0

FIGURE 3: Interdeparture time Distribution at two values for the Normalized Utilization (η = 2, 3, 5, 10, 50)