A MODIFIED ETHERNET PROTOCOL FOR LOAD BALANCING APPLICATIONS

By W. Zhang* and J.C. Majithia**

ABSTRACT

In this paper, we present a load balancing protocol in the context of a CSMA/CD local area network.

The main objective of this protocol is to achieve efficient load balancing among a group of homogeneous processing sites, interconnected by an Ethernet LAN. Usually when the bus load in such networks approaches bus capacity, the mean packet delay increases rapidly. Thus, some time-critical messages may be severely delayed in such an environment. If the Ethernet is used as the communication system in a load balancing application, then there can be severe performance degradation due to the network operation.

In the proposed network protocol, we introduce two elements viz. message priority and station privilege to achieve load balancing. The use of message priority tends to decrease the queuing delay and also ensure that different classes of messages can satisfy their individual performance requirements. The incorporation of station privilege allows for rapid access to a station with low work load i.e., dynamically attempt load balancing.

The protocol has been simulated using a LAN simulation facility. The results show that the desired objective of minimising delay and ensuring a load balance are both achievable. The modifications proposed can also be incorporated in a commercial Ethernet protocol.

1.0 INTRODUCTION

In many applications such as transaction processing, a distributed processing environment is not only desirable but may be essential. We envisage here a homogeneous environment where a set of identical computer systems or processors are interconnected by a communication network such as the Ethernet. Load balancing is usually incorporated as a strategy to achieve minimum job response time and also to achieve a balanced load distribution among the processors.

Load balancing strategies can be either static or dynamic. In general, these strategies aim at minimising the mean job response time, given a stable operation. Dynamic strategies may be more effective although they tend to be more expensive to implement. A distributed strategy is implemented at each site and can be viewed as existing at one of the protocol layers. This approach has several advantages, the principal one being the relatively self-contained implementation.

This paper deals with a modification of the Ethernet protocol to achieve load balancing in a distributed computing system. The proposed algorithm can be included in the highest layer of an Ethernet protocol. The algorithm attempts to achieve dynamically a load balance. It attempts to identify a processor with highest load and a processor with the lowest workload as quickly as possible. The paper is organised as follows.

In the next section, we describe briefly the load balancing problem and consider some of the existing techniques.

We then introduce the proposed balancing protocol in the context of a CSMA/CD Ethernet. Subsequently, we describe the various simulation experiments carried out to evaluate this protocol. The final section discusses these results and also suggests areas for further research.

2.0 LOAD BALANCING TECHNIQUES

Distributed systems support distributed execution of programmes in the sense that each program is divided into tasks, and each task is assigned to a processor. All the processors need not be identical. Hence, a distributed programme can exhibit parallel execution, and therefore is potentially faster to execute than the same programme executed on a single processor. Especially when communication bandwidths are very high and approach the bandwidth of local memory, it

* *Control Data Systems Limited, Ottawa, Ontario, Canada.*

** *Department of Computing and Information Science, University of Guelph, Ontario, Canada*

Pertinent discussion will be published in January 1996 West Indian Journal of Engineering if received by November, 1995.

becomes feasible not only to distribute tasks across processors, but also to reassign tasks dynamically to balance loads on the processors. We now describe some well known approaches to the load balancing problem.

2.1 Static Technique

There are three well-known static allocation techniques, and they are discussed below:

(i) Graphic Method

H.S. Stone[1] suggested this method for an optimal assignment of subtasks in a two processor system. The cost of interprocessor communication has been taken into account, in addition to the overall communication and other collective costs. The basic idea is to apply a maximal-flow algorithm to the graph model of a modular programme, i.e., one in which each module is represented by a node and weight on the edge connecting two nodes represents the cost of an intermodule reference when the two nodes (or modules) are assigned to different processors. Each cutset of this graph partitions the graph into two disjoint subsets, and each subset could be assigned to each processor, such that the weights of the edges comprising the cutset (which account for all costs) could be minimised. This, in turn, will minimise the total run time.

This method minimises the total interprocessor communication cost by performing a min-cut algorithm on the graph. The limitation on its generalisation is the high computing time complexity when more than two processors are involved. Furthermore, it is difficult to incorporate various constraints into such a model.

(ii) Mathematical Method

In this method, tasks are represented as x_1.

We assume:

C_{ij} is the communication cost between x_i and x_j.

r_j is the execution time of x_j.

The procedure in this method is as follows:

a) Define $d_{ij} = 0$ if x_i and x_j are assigned on the same host
$= 1$ otherwise

b) Communication time

$$T_C = \sum_{i=1}^{i=N_j=N} \sum_{j=1} d_{ij} C_{ij}.$$ N is processor number

c) Define $e_{ij} = 0$ if host i does not run x_j
$= 1$ otherwise

d) The task run time

$$T_r = \text{MAX} \left\{ \sum_{j=1}^{j=N} e_{ij} r_j \right\}, \text{ with } i = 1, 2, ..., N.$$

e) There are several constraints which must be taken into account when computing the task run time. The most obvious is the memory constraint.

Assume M_j is the memory needed for host j to execute tasks.

$$\text{MAX} \left\{ \sum_{j=1}^{j=N} e_{ij} M_j \right\} \leq M, \quad \text{with } i = 1, 2, ..., N.$$

M is the global memory limit in any host.

f) List all functions

$$\text{Cost} = \text{MAX}_{i=1}^{i=N} \left\{ \sum_{j=1}^{j=N} e_{ij} r_j \right\} + \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} (d_{ij} C_{ij})$$

$$\text{MAX} \left\{ \sum_{j=1}^{j=N} e_{ij} M_j \right\} \leq M, \text{ with } i = 1, 2, ..., N.$$

Then compute matrix e_{ij}, which will minimise the cost value.

This method can achieve optimal load balancing result, and can be used for a wide variety of systems. It allows the constraints to be easily incorporated into the allocation model to meet various application requirements. The disadvantages are the large amount of time and memory needed to obtain an optimal solution.

(iii) Heuristic Method

This approach uses a depth-first search to find a good but not necessarily optimal result. This method is also

known as the greedy heuristic[2]. For each node, the algorithm looks at the nodes in the immediate surrounding and obtains the optimal mapping in an attempt to eventually achieve a global optimisation. The algorithm makes use of two graph theoretic constructs called the Computation Flow Graph (CFG) and the Computation Resource Graph (CRG). It is a common practice to model the software for parallel machines in the form of directed graphs with directed edges representing the data dependencies, and other parameters being associated with each node and the connecting edges. In the same way, the processors and their resources could be modelled by graphs.

The CFG is a directed graph that shows the data dependency or the interrelationships with respect to data exchange between the various subtasks. Node is represented as a subtask and edge value is represented as the amount of data that will need to be transferred. The CFG is defined to be acyclic.

The CRG is an undirected graph: a node represents a processor and the edges denote the channel between the two adjacent processors. The mapping algorithm accepts these two graphs as input, and produces as output the mapping of the CFG on the CRG. Each node of the CFG can be associated with a level. A node has level 1 if it is a source node which does not need any input data from any other node. Nodes that need data from level 1 nodes are at level 2, and so on.

Allocation starts from level 1 nodes. For each level 1, a node in this level of the CFG is chosen using a criterion that selects nodes with the maximum computation and data communication cost. A match for this node is searched for in the CRG. A node of the CRG that would minimise the total communication and computation time is selected. Once all the nodes at level 1 have been matched, scheduling is continued for level 1+1, and so on, until all the nodes of the CFG have been mapped. The algorithm is restricted since it requires that the number of processors are at least equal to the number of nodes of the CFG.

Once the mapping is done, a simulation programme is run that will give an estimate of some important parameters that can be used to judge the effectiveness of the scheduling. If the results are not encouraging, another mapping may be tried.

This method combines the advantages of graph theory and mathematical approach. While the computation is not very complex, the problem is how to judge the mapping effectiveness.

2.2 Dynamic Techniques

Dynamic Techniques use the information about the current system state. A dynamic load balancing technique can be either centralised or distributed. In a Centralised Technique, a processor is specified to act as a contral processor which collects all the information about the state of each processor in the system. The central processor uses this information to send tasks to the lowest workload processor or processes them itself.

The major problem lies in the overhead of collecting status information and tasks. This overhead may be large, and scheduling decisions are frequently based on inaccurate and outdated system status information. A Distributed Technique generally uses a local monitor on each processor. Each monitor collects and updates the information about the state of the local processor. This information may be broadcast to all the remote processors. In a large system, this information can be written into a file which is accessed by all the processors. The information provided by the monitors can be used in several ways. Details of various approaches can be found in current literature. The load balancing protocol described in this paper belong to this category.

2.3 Desirable Results of Load Balancing

Indurkhya, et al[3] have considered the concept of an optimal load balancing policy. The discussion is based on the following assumptions:

P_a represents processor number; B represents the largest bandwidth of the bus;

N represents the total task number.
It attempts to allocate N among P with P being a constraint.

The analysis also assumes task is allocated to 2 or more processors, is therefore somewhat restricted.

The analysis develops models for the communication time and run time and attempts to optimise these under various task allocation policies.

The optimal task assignments are extremal in the sense that tasks are totally distributed as evenly as possible or not distributed at all. The maximum throughput occurs when processing time is distributed as equally as possible over the participating processors. The surprising result for the homogeneous case is that the optimal policy uses all processors or one processor depending on the ratio of running time

to communication time.

In general, almost all of the present algorithms tend to be application-dependent. A viable objective then would be to develop a load balancing protocol which is not application dependent. We now describe a modification which has this feature.

3.0 A MODIFIED ETHERNET AND A LOAD BALANCING PROTOCOL

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocols have been extensively studied and several commercial implementations have been available for some time. The Ethernet is an example of such an implementation. The load balancing algorithm assumes the use of a standard Ethernet and introduces two modifications in the basic transmission protocol[4]. In this modified Ethernet, the transmission works on a window basis is as follows:

(i) After acquiring the bus, the sender transmits W (window size) packets to a receiver. The receiver will acknowledge in the next slot after receiving W packets. If the receiver receives a packet with error, then it does not send out any acknowledgement. Thus, the sender will wait one slot duration and then retransmit all W packets. It should be observed that the acknowledge mechanism is quite similar to that proposed by Tokoro[5] Extensive simulations have been performed for such a modified Ethernet. Results[4] show that this negative acknowledgement scheme has a performance better than other schemes for a wide range of window sizes. For our study, the window size was fixed to 16.

(ii) A second modification concerns a priority scheme in the Ethernet protocol. Prioritising the handling of various data and control messages is often important for network performance. It serves several purposes. In the context of load balancing, it is an essential feature as will be seen later.

The load balancing protocol using the modified Ethernet is now described. With reference to **Fig. 1a**, the input queue is used to store messages from the local user. The output queue is used to store messages which are to be transferred to other stations. Messages are queued according to their priority or transmission privilege.

When a message comes into a station from a local user, it will be put into the input queue which operates on a FIFO basis. When it reaches to the head of the queue, it will be checked and directed either to the local processor or put into the output queue if it is for remote processing. Thus, the output queue contains two types of messages viz:

(i) Those requiring remote processing and

(ii) Result messages due to processing of an external message.

For simplicity, we assume that the messages can be accommodated in W packets where W=16 is the window size. The access to the Ethernet i.e., selection of the transmitting station is decided by a priority scheme. Each station has a priority assigned to it and all stations take part in a process called HPSIP (Highest Priority Station Identification Process). It is at the end of a HPSIP, that a station is selected to begin transmission. If at the end of HPSIP, the receiver needs to be selected, then the transmitting station prepares and transmits a broadcast packet. This initiates a process called LWSIP (Lowest Workload Station Identification Process). Thus, at the end of a LWSIP, the receiving station would also be decided. These two strategies executed at each node, collectively constitute load balancing scheme. Both involve two parameters - message privilege (or priority) and station privilege. The message privilege is dependent on message type and the relative urgency in delivery. Thus, acknowledgements and result return messages have the highest privilege. A one-packet message (message whose length is less than or equal to one packet) has the privilege value of 3. A two-packet message has the privilege value of 2. All other length messages have a privilege of 1. Thus, shorter messages are processed before longer messages.

For multiple packet messages, after one part of message has been transmitted in one transmission period, the privileges of remaining message will be increased by one, so as to ensure that an entire message is transferred as soon as possible.

We define the station priority as:

$P = k_1 *$ the station workload +
$+ k_2 *$ the first message privilege in the output queue +
$+ k_3 *$ the number of contentions to seize the bus.

The definition of workload L is also quite simplistic and is based on the data volume in the local

process queue and in the output queue. Parameters k_1,k_2 and k_3, can be varied to suit the need of a particular application. For example, values of 2,1,1 respectively implies that more significance is attached to the workload than either the message priority or the number of contentions.

The values of P and the station workload L can be constrained to lie in a range decided upon by particular application.

The station priority and station workload are translated into binary numbers, and used for HPSIP and LWSIP, as described in the next two sections.

3.1 Highest Priority Station Identify Process (HPSIP)

When a station is involved in HPSIP, it will operate as follows:

Station calculates the station priority and translates it into binary number.

(a) For all bits of the P (the binary number representing station priority) except the last bit P_0.

If the bit is 1, a bit packet is put on the bus and if successful, then the station stops HPSIP and becomes highest priority station. If not, then the next bit is tried in the next slot.

If the bit is 0, and the bus is "idle", then the next bit is tried in the next slot. If the bus is not "idle", then the station stops comparison and waits for next HPSIP.

(b) For the last bit P_0.

If the bit is 1, then station is the highest priority station.

If the bit is 0, and the bus is "idle", then station is the highest priority station. If the bus is not "idle", station stops comparison and waits for next HPSIP.

The highest priority stations can begin transmission, given that the destination processor is known. If they are involved in collisions, they will take part in a contending period to decide on one station to seize the bus. If the destination is not known, then the protocol goes into the process called LWSIP, where the processor with the lowest workload is determined and designated as the receiver. Here, it is assumed that a task can be processed by any processor on the network. Thus, all stations can participate in the LWSIP process. We now describe the activities involved at a node participating in LWSIP.

3.2 Lowest Workload Station Identification Process (LWSIP)

When a station is involved in LWSIP, it will work as follows:

Station calculates its workload, translates the value of workload into binary number and uses the complement of this binary number to take part in LWSIP.

From the beginning of LWSIP, each station does:

a) For all bits (the complement binary number representing station workload) except the last bit P_0.

If the bit is 1, a bit packet is put on the bus and is successful, then station stops LWSIP and it is the receiver. If not, then the next bit is tried in the next slot.

If the bit is 0 and the bus is idle, then the next bit is tried in the next slot. If the bus is not "idle", then station stops comparison.

b) For the last bit P_0.

If the bit is 1, then station may be the receiver (it may need to take part in contention period).

If the bit is 0, and the bus is "idle", then station may be the receiver (it may need to take part in contention period). If the bus is not "idle", station stops comparison.

In HPSIP, we use the binary number representing the station priority to take part in comparison. In LWSIP, we use the complement value of the binary number of the workload to take part in the comparison. So, in HPSIP, we get highest priority station and in LWSIP, we get lowest workload station. We can now describe the operations involved in using these two strategies in a load balancing environment.

4.0 THE LOAD BALANCING PROCESS

After a message is scheduled for a remote processor, it will be put into the output queue. The local station will prepare the packets and take part in HPSIP to identify highest priority stations. Stations remaining after an HPSIP will keep a record about this process, then try to seize the bus as sender. After one station becomes the message sender, it will transmit its packets.

If this packet is data packet, sender will send out

up to W (Window Size) packets to the receiver, and then wait for an acknowledgement.

If this packet represents a task requiring processing, then the sender will lead a LWSIP by sending a broadcast packet to select the lowest workload station as message receiver, send up to W (Window Size) packets to this receiver, and wait for an acknowledgement.

If message receiver cannot be decided in LWSIP(e.g., after a sender broadcasts a packet but no station replies in one slot), sender will put a call up packet on the bus to let all other stations know that it has finished.

After an acknowledgement or a call up packet disappears on the bus, sender will eliminate its record and wait for a duration of two slots to try again. The other stations which have kept a record of the HPSIP can begin to transmit. They may be involved in collision, and hence take part in contention period to resolve the conflict and allow one station to transmit packets. Thus, all these stations remaining after HPSIP will get a chance to transmit sequentially.

After all these stations have finished transmission, each will wait until at least two idle slots appear on the bus. This implies that the current transmission period has ended. A new transmission period will begin with HPSIP. A simple example is used to illustrate the process. Assume that initially, the Ethernet is idle. After one station successfully transmits packets, it will lead a new transmission period beginning with HPSIP. Suppose there are three stations ready to transmit. Station 1 has priority of 30 (11110), workload of 24 (11000), the complement of workload of (00111); station 2 has priority of 20 (10100), workload of 20 (10100), the complement of workload of (01011); Station 3 has priority of 30 (11110), workload of 20 (10100), the complement of workload of (01011) **(Fig. 1b)**.

Station 1, Station 2 and Station 3 put bit packets on the bus. In the second and fourth slot, Station 1 and Station 3 put bit packets on the bus. Station 2 loses. The next slot is idle, because no station puts a bit packet. After HPSIP, Station 1 and Station 3 begin to transmit and they are involved in collision. They will take part in contention period to allow one station to seize the bus. Suppose Station 1 wins (Station 3 will keep a record about this contention) and transmits a result return message to Station 3, Station 3 will acknowledge Station 1. After acknowledgement, packet disappears on the bus and Station 3 begins to transmit. Suppose it broadcasts a workload packet in next slot, no station replies to it because the Station 3 is lowest workload station. Next, Station 3 will put a call up packet on the bus to let all stations know it has finished transmission. After two idle slots on the bus, all stations know current transmission period has ended. A new transmission period will begin with HPSIP.

5.0 SIMULATION RESULTS

The load balancing protocol is adaptive and the refore not readily amenable to an accurate mathematical analysis. Hence, we have resorted to a simulation approach. The simulation is based on a simulation facility - LANSF (Local Area Network Simulation Facility) which has been developed by Gburzynski and Rudniki at the University of Alberta[6]. The simulation is written in C and run under BSD 4.3 UNIX on DEC 3100 workstations. The model simulated has ten stations on a CSMA/CD Ethernet operating at 10 Mbps. The basic time unit in the simulator is an ITU where 1 ITU = 10 sec. for a 10 Mbps LAN. The simulator required several other parameters to be specified. Details of these are given in [4]. Each run processes 600 data messages, or until such time when results are stable. While the simulation can provide a wide variety of results[4], we will focus on load balancing, throughput, mean message delay and overhead. The load distribution is uniform among all stations with a local/remote distribution being either 0.8 or 0.5. The various queue sizes are calculated at the end of the simulation by calculating the packet numbers in the local process queue and the output queue for each station, and choosing the maximum number and the minimum number. The difference of these two numbers is represented in **Fig 2.** The maximum difference is 96 packets. This is just about 6 data messages (each data message is 16 packets) and just 1% of 600 messages, so the load is fairly evenly distributed among the stations. We also observe that the load balancing improves considerably when the message interarrival rate is high. This is as expected: When the message interarrival rate is low, the number of queued message in each station changes very fast, so effective load balancing result is not possible.

The throughput is the CSMA/CD bus throughput, and is defined as the ratio of successful transmitted data packets and result return packet bits to the total simulation time. **Fig 3** shows the results for the two load distributions. The throughput for LR=0.5 is improved very evenly under different message

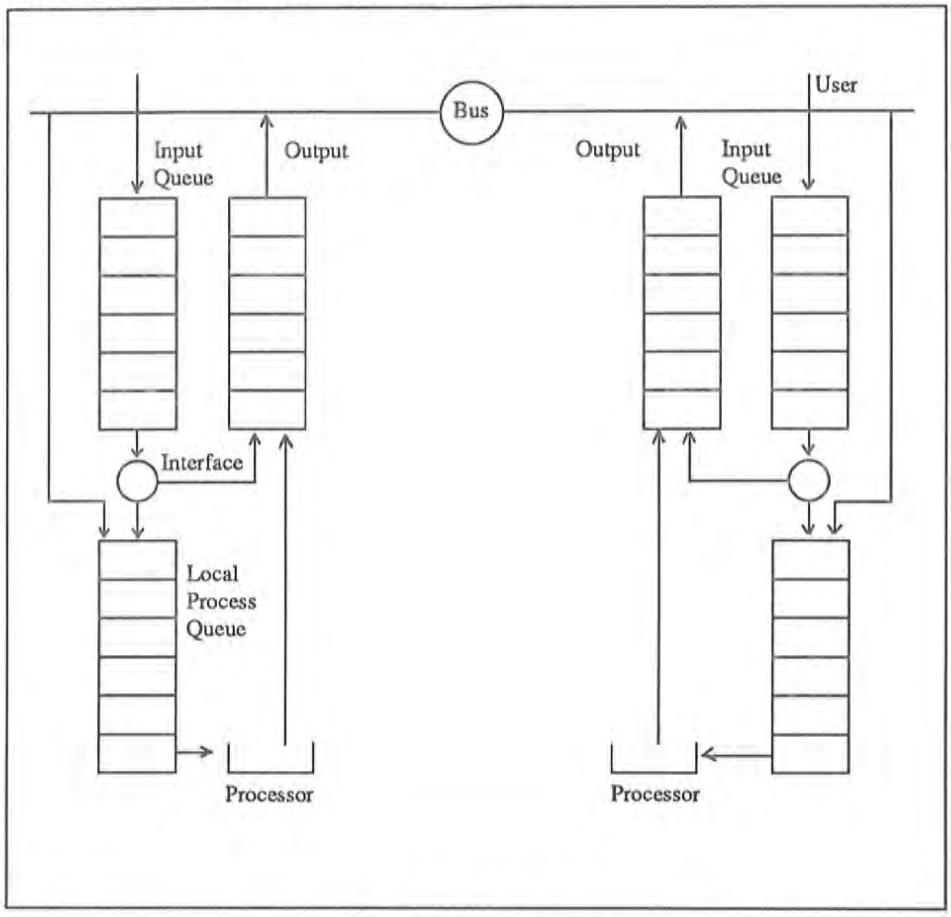

Figure 1a: **Network Model**

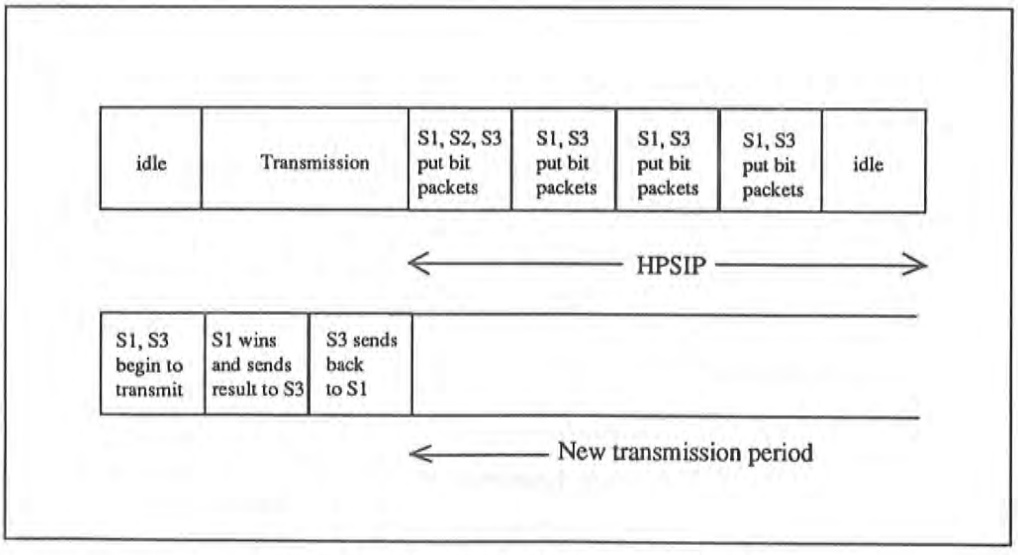

Figure 1b: **Example of a Transmission Phase**

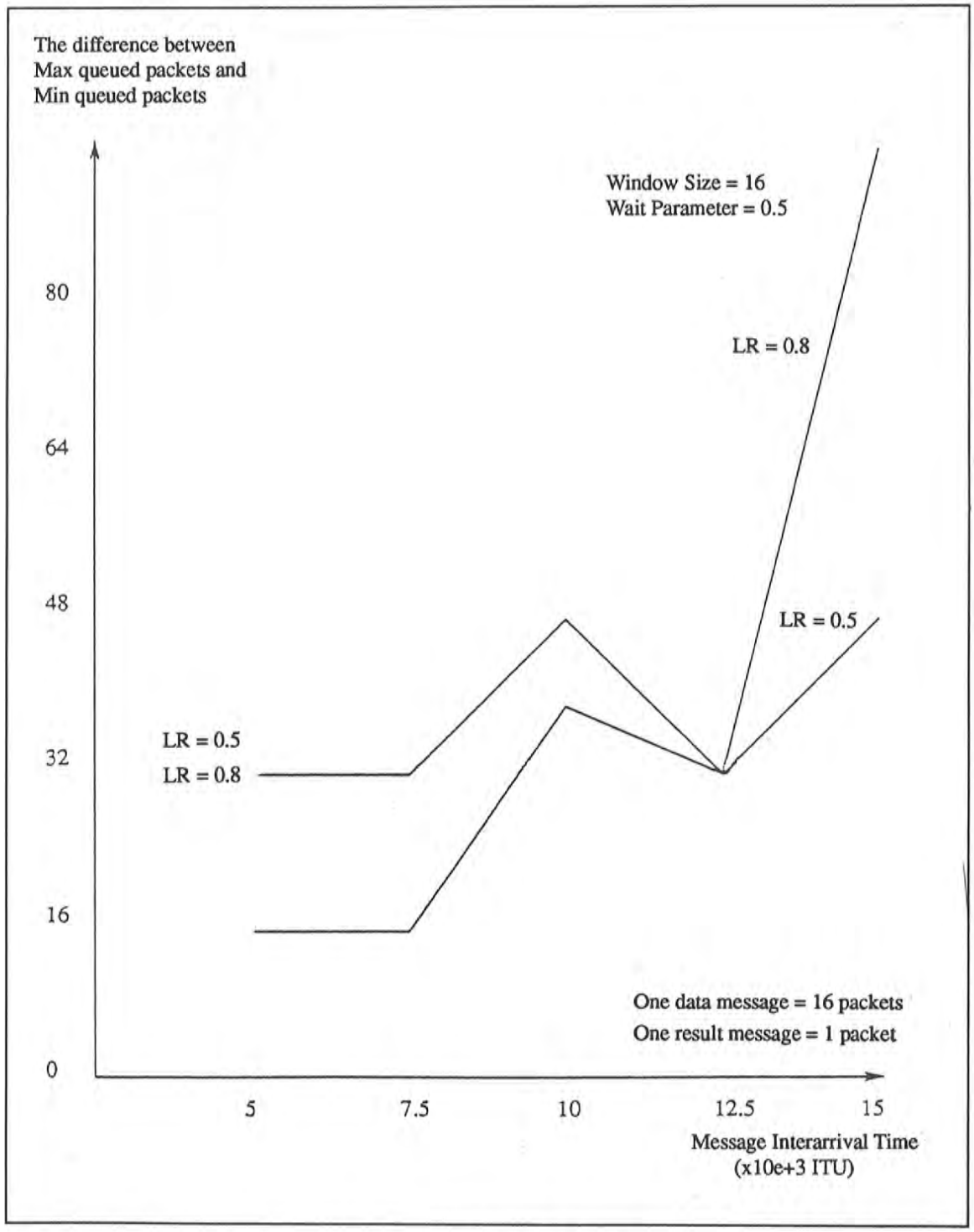

Figure 2: **Load Balancing Result**

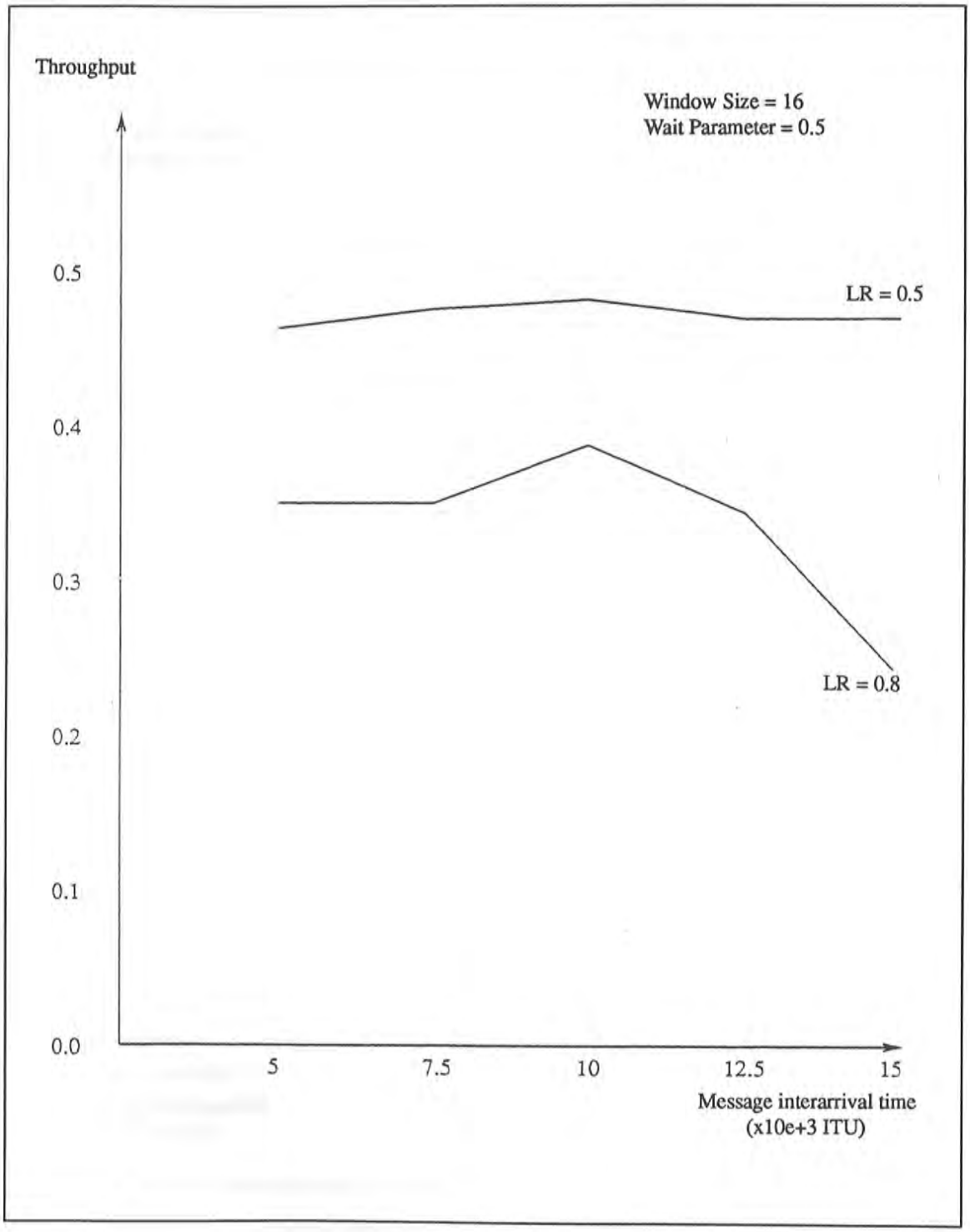

Figure 3: **Throughput Results**

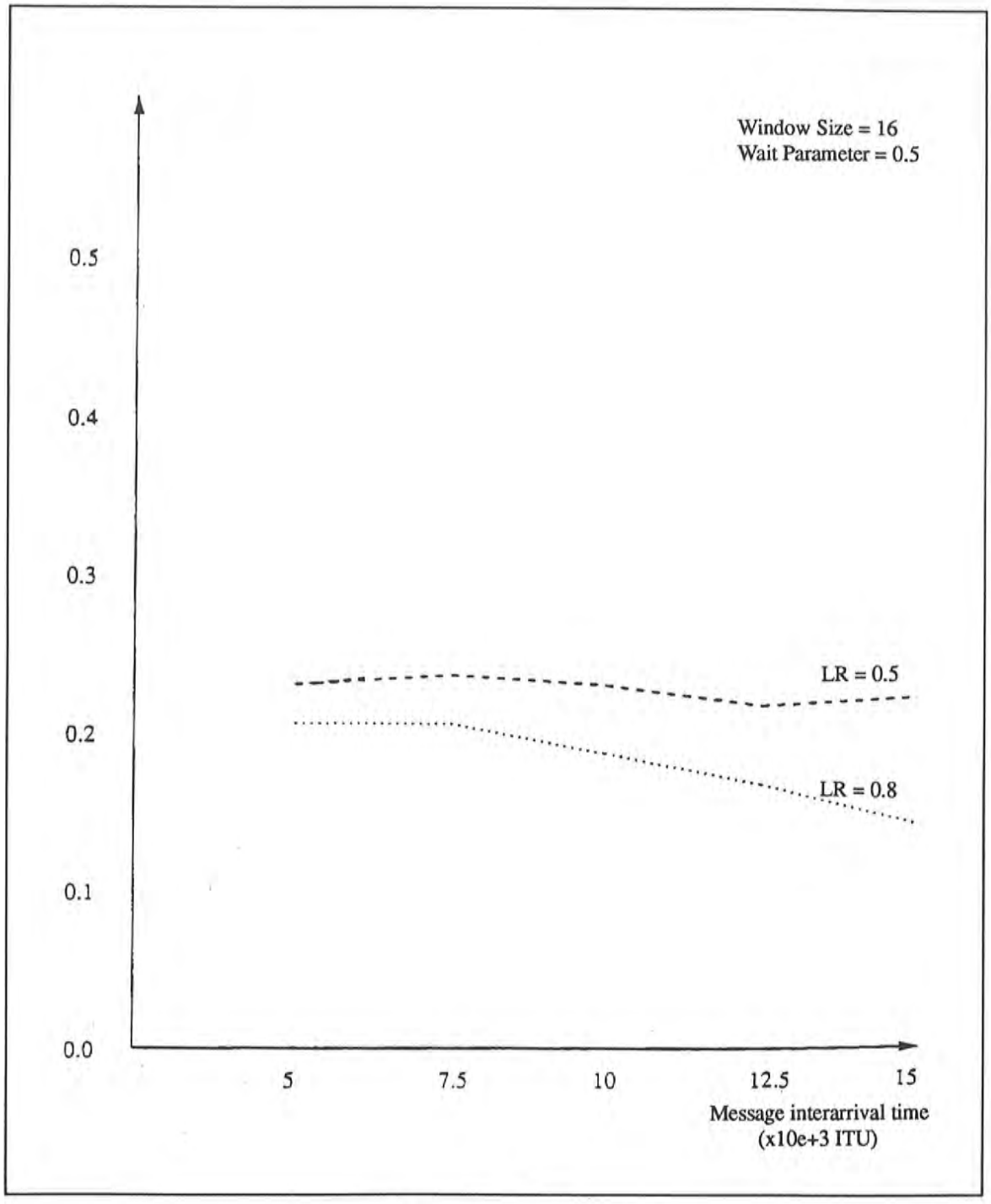

Figure 4: **Overheads**

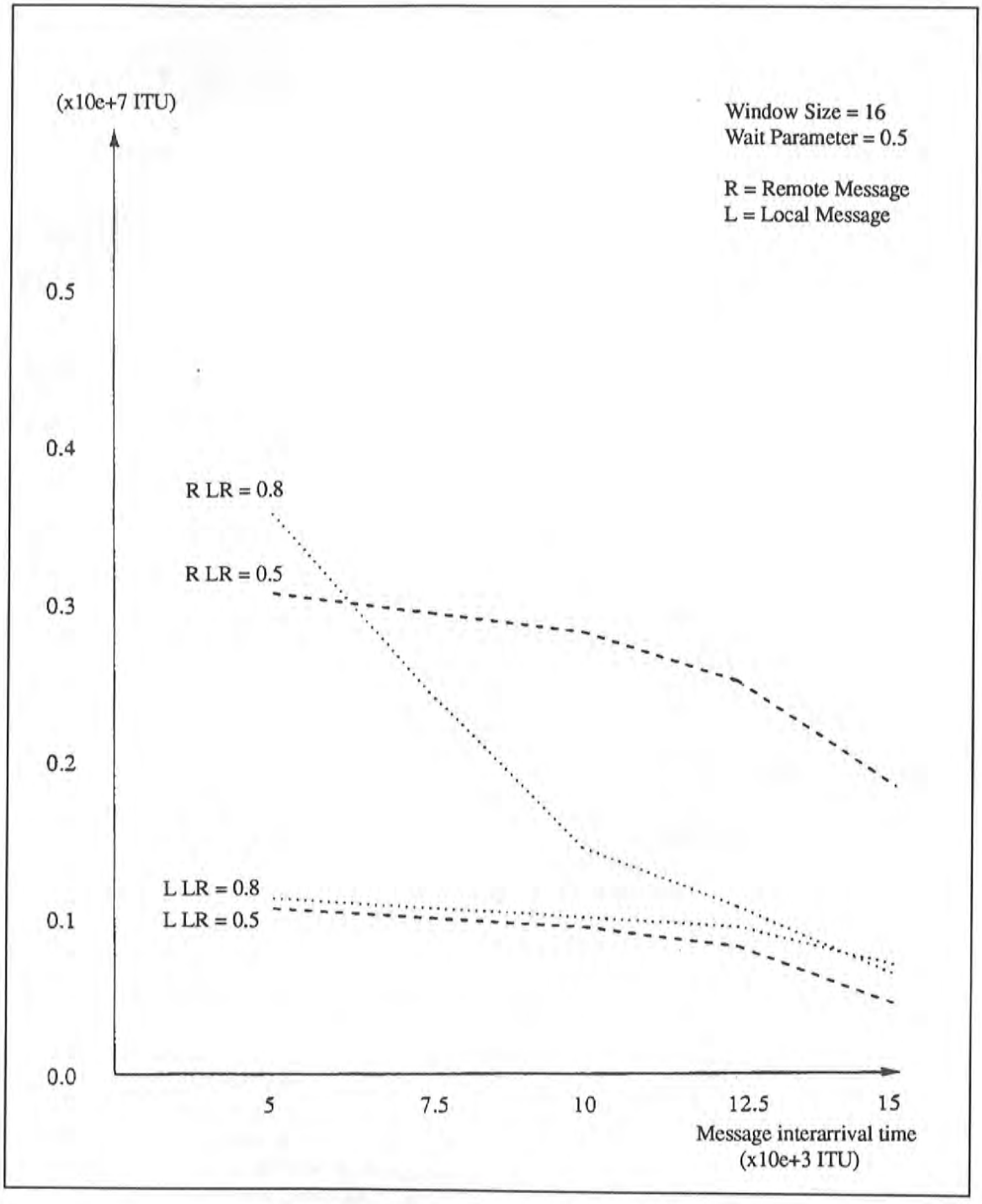

Figure 5: **Mean Remote and Local Message Delay**

interarrival rate. This is because of the reduced HPSIP times. For LR=0.8, when message interarrival rate is high, the throughput is improved considerably. The reason is the same as above. For LR=0.8, when message interarrival rate is low, the throughput is lower than that of the modified protocol. This is because almost all station workloads are near zero, the highest priority stations which do not get chance to transmit first in one transmission period leads LWSIP repeatedly but is not able to find a message receiver.

Fig. 4 shows the ratio of overhead to the total processed message bits in simulation. The overhead is arbitrarily defined as:

The LWSIP times * mean slot number in each LWSIP * bits in each slot+ HPSIP times * mean slot number in each HPSIP * bits in each slot.

This is an estimate of the overhead for load balancing protocol. **Fig. 5** shows the mean remote message delay and mean local message delay. The results are self-explanatory.

6.0 COMMENTS

The load balancing algorithm presented in this paper relies on two parameters. Both are calculated locally for each site. The balancing algorithm uses network and node load information and in a broad sense, a distributed algorithm.

This approach has a significant advantage in that by varying the remote load priority and access privilege, the algorithm can be adapted to a variety of applications. In the general application considered in our simulation work, we noticed that the maximum difference between the largest and smallest queues was about five messages. This difference was observed with a run involving 600 messages. Furthermore, the overhead is typically about five transmission slots. It is possible to reduce this even further. The proposed protocol is robust and inherently independent of other processes in the system.

The proposed protocol can be improved further primarily by redefining the number of priorities in HPSIP. It is also possible to make the entire protocol adaptive to changing load volume conditions.

7.0 ACKNOWLEDGEMENTS

This work has been supported by research grants from NSERC, Canada.

8.0 REFERENCES

1. Stone, H.S., *"Multiprocessor Scheduling with the Aid of Network Flow Algorithms"*, IEEE Tran. Software Engineering, Vol SF3, pp. 85-94 Jan. 1977.

2. Baxter, J., Patel, J.H., *"The Last Algorithm: A Heuristic based Static Task Allocation Algorithm"*, Proc. of 1989 Int. Conf. Parallel Processing, Aug, 1989 pp. 217-222.

3. Indurkhya, B., Stone, H. S., Lu X. C., *"Optimal Partitioning of Randomly Generated Distributed Programs"*, IEEE Trans. Software Engineering, Vol. SE 12 No. 3, March 1986, pp. 483-495.

4. Zhang, W., *"A Modified CSMA/CD Local Area Network for Load Balancing Application"*, M.Sc. Thesis, University of Guelph, 1991.

5. Tokoro M., Tamaru K., *" Acknowledging Ethernet"* COMPCON, Fall 1977, pp.320-325.

6. Gburzynski, P., Rudnicki, P., *"The LANSF Protocol Modelling Environment"*, Technical Report TR 89-19, University of Alberta, September 1989.