SHIFT ARITHMETIC ON A TOKEN RING NETWORK

by

Miron Livny
and
Udi Manber

Computer Sciences Technical Report #581

February 1985

SHIFT ARITHMETIC ON A TOKEN RING NETWORK‡

(Preliminary Version)

Miron Livny and Udi Manber

Department of Computer Science
University of Wisconsin
Madison, WI 53706

February 1985

ABSTRACT

We propose in this paper an extension to the token ring protocol that allows special type of arithmetic, called *shift arithmetic*, to be performed directly on the nodes' interfaces. The new protocol is based on the approach that the communication channel together with the interfaces can form an environment in which simple commands can be executed. Each command operates on operands located at the interfaces and places the result at the interface that initiated the command. The commands utilize the mandatory 1-bit delay of the token ring protocol to implement arithmetic and logical operations on the operands. We present a low level command structure, and high level very efficient algorithms that use the simple commands to compute commonly used functions. The goal of this protocol is to enhance the performance of distributed algorithms on ring networks by performing many simple tasks in the lowest possible level. This reduces the processing overhead generated by the need for synchronization and data movement through the different protocol layers. Potential application areas include parallel algorithms, distributed simulation, load balancing, distributed operating systems, distributed databases, and real-time computations.

1. INTRODUCTION

Local area networks are designed to provide low cost fast exchange of information among computers in a limited area. The complexities involved in designing protocols that support the network operations lead to a design that includes several hierarchical layers. The protocol of the lowest layer (the physical layer) is responsible for the raw transmission of bits, while the highest layer deals with the actual applications. In between, there are layers that are concerned with establishing connections, dividing messages to packets and then to data frames, receiving acknowledgements, providing fault tolerance, etc. Some of the main delays involved in local area network communication are caused by the need to move through all the protocol layers.

In this paper we study the possibility of extending the function of a local area network from "passive" exchange of messages to "active" computation involving information distributed among the communicating nodes. An example for such a computation is evaluating the average machine load in a distributed system, which is very useful for load balancing. The data required for this computation in each node (in the example above one number) is very specific and need not go through some of the protocol layers that deal with general type of data. In addition, the physical layer in

‡ This research was supported in part by the National Science Foundation under grant MCS-8303134.

some cases already contains significant processing power and can handle the computation (addition in the example above) with little overhead. We regard the communication channel together with the interfaces as an integral environment in which simple commands can be executed. We call this environment, together with the command structure, ATON. Each ATON command operates on operands located at the interfaces and places the result at the interface that initiated the command. The protocol we propose is an extension of the token ring network protocol. It supports several simple commands very efficiently. These commands can in turn be used to implement algorithms for a variety of applications.

In a token ring network a special bit pattern, called the *token*, is circulated around the network. Each station (node) that wants to transmit listens and tries to identify from the sequence of bits it receives the bit pattern corresponding to the token, say 11111111. When the token is identified the station modifies it (e.g., by inverting the last bit to 0) and then transmits its message. This station is said to be holding the token. Since there is now no token, no other station can transmit. The token must be appended to the end of the message. In order to decide whether a sequence of bits is indeed the token, each node, which receives one bit at a time, has to "run" it through a pattern matching circuit. If a bit is found to be the last bit of a token then it is inverted; otherwise the same bit is sent. As a result, there is a 1-bit delay caused by the decision process [Ta81].

Consider a loop network of n node machines such that each machine i maintains a number x_i. Assume that we want to compute the sum of all the x_i's and have the result in one machine. In a regular network architecture each machine will have to send a message containing its value. Whether all messages go to one machine or a distributed algorithm is used (e.g., summing all the numbers along a logical tree), it is clear that $n-1$ messages are required. A loop network allows only one message at a time, hence a significant time is required if n is even moderately large. On the other hand, if we add to the pattern matching mechanism in the token ring protocol a simple adder we can perform the addition of all the numbers with only one "active" message. Assume that one node initiates the computation and sends its own number to the network with the least significant bit first. Let each machine maintains a register with its number, and a bit for the carry; when the bit from the network comes in it is added to the appropriate local bit and the carry, the output is sent to the neighboring node, and the carry is updated. This is a simple example of the kind of operations that can be efficiently performed directly on the nodes' interfaces. The reason for the efficiency is that addition can be performed sequentially bit by bit. We call such computations *shift arithmetic* computations.

In this paper we present a low level command structure, and high level very efficient algorithms that use the simple commands to compute commonly used functions. The paper is organized as follows. In section 2 we present a command structure that supports several functions which can be computed by shift arithmetic. In section 3 we discuss how to implement the command structure on top of the token ring protocol. We selected functions which on the one hand offer a wide variety of applications, and on the other hand are easy to implement. These functions are by no means the only functions that can be efficiently implemented. They serve as examples for the validity of the idea rather than a final design. In section 4 we give examples of algorithms that utilize active messages and demonstrate that shift arithmetic is powerful. In section 5 we describe several applications, in section 6 we suggest extensions to the basic implementation, and we present conclusions in section 7.

We are only at the beginning phases of the ATON project and the emphasis in this paper is on presenting the main ideas and their feasibility rather than describing a complete system. We do not advocate using this extension for every network. In many cases there are few, if any, arithmetic operations of values distributed among the nodes that need to be performed. In these cases the network serves only as a logical point-to-point communication network with little cooperation or sharing. We believe though that as more distributed operating systems are built and more distributed applications are developed there will be a need for very efficient computation in the lowest level. Several such applications are discussed in section 5.

2. COMMAND STRUCTURE

The operands of an ATON command are integers that are located at the network interfaces. Each ATON interface has an *operand register* in which the operand is stored. The command operates on this set of physically distributed operands and produces an integer result that is placed at the interface that initiated the command. The command is executed in a pipeline fashion concurrently on the n interfaces. Each stage of the pipeline is an operation on two or more bit operands. The first bit operand is taken from the stream of bits coming from the network. The second bit operand is taken from the local operand register. There may be other bit operands, for example a carry for addition or a condition code to compare operations. The operation produces a one bit result that is sent out to the network, and it may also modify the local operands.

Each interface may trigger the execution of a command by sending an active packet. The access to the ring is controlled according to a regular token ring protocol and thus an interface has to wait for the token to reach it before a command can be triggered. An interface that wishs to trigger a command, the *T-interface* of the command, must first build an active packet in a local buffer. The type of packet, passive or active, is determined according to the value of a flag which is part of the header of the packet. Both types of packets have the same structure. Figure 1 illustrates the format of a packet. A packet consists of three main parts, *header*, *data*, and *trailer*. The header contains four fields. The first field marks the beginning of the packet, the second field contains a number of control flags, the third field includes the destination address, and the fourth field contains the address of the interface that sent the packet. The set of operands on which an active packet operates is determined by the value of its destination address. This set of operands constitutes the *execution region* of the command. We currently allow only two types of execution regions - the entire network or a single interface. However, it is possible to extend the addressing mechanism so that execution regions of various sizes are supported.

The data part of an active packet can be viewed as a carriage on which intermediate results are transferred from one interface to its neighbor. ATON is taking advantage of the fact that the architecture of a token ring enables every interface to change the content of a packet as it passes by. The interface uses the information stored in the data part of an active packet as an input operand and replaces this information with the result of its local computation. The data part consists of three fields, *Counter*, *Variable-1*, and *Variable-2*. Counter records the number of participating interfaces. Each active interface increments the value of Counter. Counter is used by the T-interface to determine whether the expected number of interfaces have taken an active part in the execution of the command. Variable-1 and Variable-2 contain intermediate results whose type depends on the command. They serve as the operands to the pipeline. More details on these fields will be given later.

The trailer of the packet consists of three fields. The first field contains a set of standard control flags, the second field contains the Cyclic Redundancy Check (CRC) of the packet, and the third field is a delimiter that marks the end of the packet. The CRC field of an active packet will be modified by every interface that participates in the execution of the command. An interface uses the incoming CRC to verify the content of the data it has received. It then sends a new CRC that reflects the changes in the active packet. In case of an error, the interface that detects the error places a CRC that does not match the new content of the packet. As a result, the error will propagate to the T-interface, which will then disregard the result.

Eight instructions constitute the basic command set of ATON. The logic of all basic commands does not require more than 1-bit delay per interface. Each basic ATON command can be executed as a sequence of one bit operations and thus no intermediate buffering is required at the interface. We assume that the token ring uses a *differential Manchester* code [AS82] to convert binary data into analog signals so that there is no need for a *bit stuffing* mechanism [Pr82].

In the comming paragraphs we will present the commands and define their meaning. We denote the operand of the T-interface by x_1 and the rest of the operands by $x_2, \cdots, x_n$ in the order they appear in the ring. A simple hardware extension to the token ring interface that implements the ATON command set will be discussed in section 3. The ATON command set consists of the

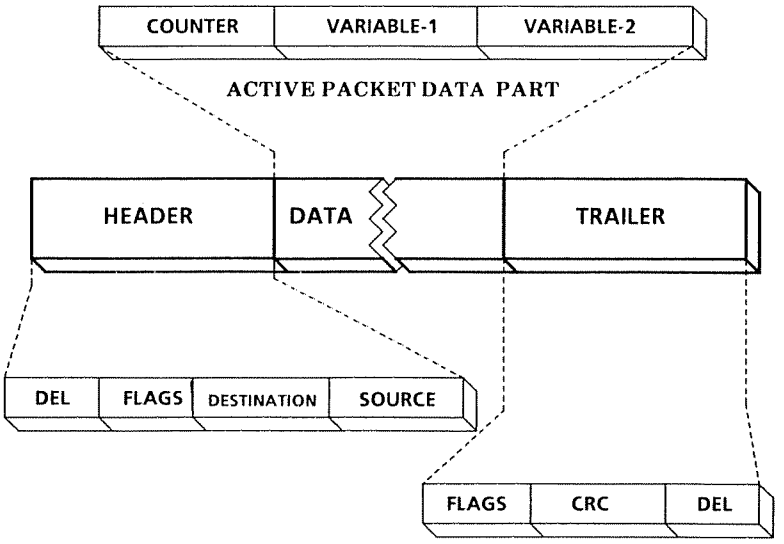

Figure 1: ATON Packet Format

following three groups of commands:

- **arithmetic**: The first group of commands includes three commands, *ADD*, *AND*, and *OR*. The first command computes the sum of $x_1,\ldots,x_n$ whereas the two latter commands perform the AND and the OR operations on these n operands respectively. Each interface places the intermediate result of the computation in Variable-1.
- **select**: In many cases a group of numbers is searched in order to select one of its members that meets a given condition. A search for the largest number or the smallest number of a group are two examples of such an operation. So far we have included only two commands in this group - the MAX and the MIN command. The MAX command computes the maximum of $x_1,\ldots,x_n$ whereas the MIN command finds the minimum of these n operands. However, we believe that more commands will be included in this group as more algorithms that are based on the ATON protocol will be developed. Variable-1 plays the same role in the execution of a select command as it plays in the previous group of commands. However, unlike arithmetic commands, a select command utilizes Variable-2 as well. The address of the interface whose operand register has been selected by the command is stored in Variable-2. (If several operands satisfy the condition then the last address will appear in Variable-2.)
- **counting**: Unlike the previous command groups, the value of x_1 plays a special role in the execution of commands that are part of the count command group. Each count command counts the number of times x_1 *REL* x_i is true, where REL is either $=$, $<$, or $\leq$. The value of x_1 is stored in Variable-1 by the T-interface. It remains unchanged throughout the execution of the command. Variable-2 is used as the counter. Note that by using the value of the counter field

from the header part of the packet the T-interface can find the number of operands that do not meet the given relation. Therefore all basic relations are covered by these three commands.

3. IMPLEMENTATION

In this section we present a schematic description of the structure and functionalities of the ATON interface. We will address the key issues related to the structure and operation of the interface and discuss how the different types of commands are executed by the interface.

The structure of an ATON interface is similar to the structure of a token ring interface like the ProNET interface [Pr82]. The interface consists of a *Modem*, *Input machine*, *Output machine*, and a *buffer*. Figure 2 presents the schematic structure of an ATON interface. The buffer unit of an ATON interface includes, besides storage facilities for incoming and outgoing packets, a set of registers. The current set of commands utilizes ten different registers. One of these registers is the operand register. The results of a command are stored in three registers. All registers can be set and tested directly by the host CPU. Accessing the registers should be a simple operation with minimal overhead. The input machine is responsible for address recognition and OPC identification. The machine maintains bit and field counters, and directs both incoming data and the results of the different commands to the buffer unit. Transient passive packets, as well as active packets triggered by other interfaces, are forwarded by the input machine to the output machine.

The novelty of the ATON protocol is encapsulated in the output machine of the interface. In figure 3 we show the structure of the output machine. The main logic associated with the execution

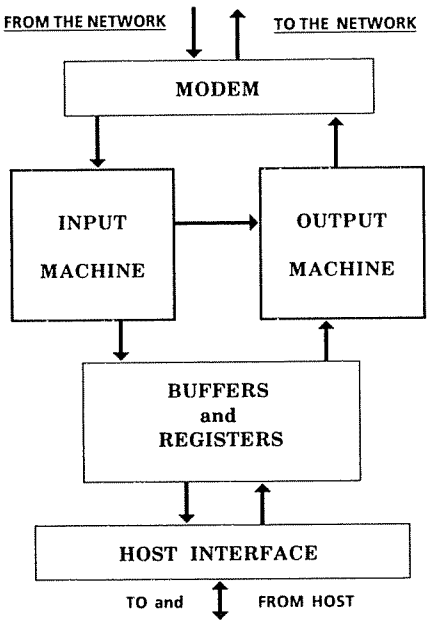

Figure 2: ATON Interface

of an ATON command is located at this machine, which is composed of two components - an *arithmetic unit* and a *compare/select* unit. The arithmetic unit consists of a one bit adder, an AND gate and an OR gate. The three arithmetic commands as well as all the counting operations associated with the execution of other commands are performed by this unit. The operands of an add operation, both the one stored at the interface and the one stored in the active message, enter the add operation with their least significant bit first. The operand is stored in the packet with the least significant bit closest to the header and the operand register is fed into the arithmetic unit via a shift operation that moves from the most significant bit towards the least significant bit.

The second unit, the compare/select (CoSel) unit, is composed of a *selector*, a set of flags, and a *Test and Set* (TeSet) component. The CoSel unit performs the selection commands and checks the relation of count commands. The flags, set by the TeSet logic, control the operation of the selector. The selector determines whether the bit coming from the network or the bit coming from the operand register will be sent to the network at the next clock tick. The replacement of the Variable-1 field of a MAX or MIN command is performed by this component of the CoSel unit. The TeSet component identifies the first bit place in which the operand register differs from Variable-1 and determines whether Variable-1 is greater than or less than the operand register. A flag that reflects the relation between the two operands is set accordingly. In the case of a MAX command setting the *Greater Then* flag leads the selector to send bits from the operand register to the network, whereas the same setting will have an opposite effect in the case of a MIN command. The flags are also used when the Variable-2 field of selection and counting commands is processed. The values of the flags determine

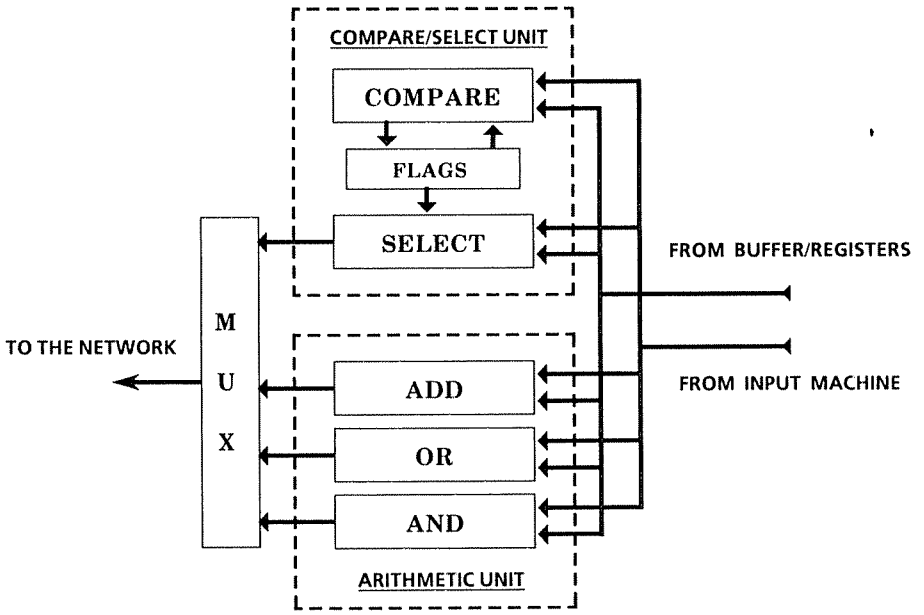

Figure 3: ATON Output Machine

whether the Variable will be incremented (in case of a count command) or loaded with the interface ID (in case of a selection command).

4. EXAMPLES OF EFFICIENT ALGORITHMS

In this section we give examples of efficient algorithms to demonstrate the power of shift arithmetic. We have already shown that certain simple algorithms such as finding the minimum or maximum can be performed with one message. More complicated algorithms, which require some modifications to the basic command structure, are described here. We believe that these algorithms are typical of the use of ATON. The necessary modifications to the command structure will become obvious with the description of the algorithms.

1. **Sorting**

   Sorting can be performed with one (long) message in the following way. Initially the T-interface sends its operand. At step i the message consists of $i-1$ operands in increasing order following each other. Node i compares its operand with each of the $i-1$ operands until it finds the correct place to insert it. It then uses a shift register to buffer the rest of the message (one operand at a time) while it inserts its operand to the message. (This is similar to the way insertion rings work; see [Li78]). We now show that finding the right place to insert can be done with only 1-bit delay. Let the incoming message be denoted by $a_1 a_2 \cdots a_{i-1}$ (the a_k's are the sorted order of x_1 to x_{i-1}, and they are represented with the most significant bit first). The operand x_i at the node is compared bit by bit to the a_k's. At the same time, the current a_k is put in a shift register. This is done in a very similar way to the MIN or MAX commands. Assume that $x_i > a_{k-1}$ and consider the comparison of x_i and a_k. Let x_i and a_k differ first at bit j. This bit determines which one of the two is larger. If x_i is larger then the node continues to output a_k; otherwise it continues to output the rest of x_i while the rest of a_k is put in the shift register. In this case a_{k+1} will be put in the shift register while a_k is output, and so on. At the end the message will contain all the n operands sorted. The amount of delay equals the total number of bits, and this is optimal in the worst case since all the bits may have to move. No additional time, besides the message delay, is spent for the actual sorting.

2. **Finding a majority**

   The problem is the following: Given n nodes such that each node i contains an integer x_i with k bits, determine whether there exists a *majority*, namely a number that appears more than $\frac{1}{2}n$ times. If a majority exists then find it; otherwise report that no majority exists. This problem is important for recovery in distributed databases [Th79]. An optimal sequential solution is given in [FS82]. The solution we propose consists of two active messages. The first message is divided into k parts each of size $\lceil \log_2 n \rceil$. (This message is longer than regular messages.) Each part corresponds to one bit. Initially each part contains, as the leading bit, the corresponding bit of the node that initiated the message. Each node adds its bits to the corresponding parts. The outcome of this message is thus for every $0 \le i \le k-1$ the sum of the n i'th bits. The originator of the message now computes a number x such that bit i in x is 1 if and only if the corresponding i'th sum is greater than $\frac{1}{2}n$. If there exists a majority M then in particular each bit in M appears more than $\frac{1}{2}n$ times, hence it is equal to x. We only need to verify that a majority exists. This can be done by sending another active message with the basic command of counting the number of times x appears.

3. **Emulating tree computation**

   Consider a computation which is performed on a (logical) tree such that the operands are in the leaves and each internal node represents an arithmetic or logical operation. For example, the tree can be a parse tree of an arbitrary expression. Assume, for simplicity, that the tree is balanced, that it contains n leaves (n is a power of 2) each of them resides at a different node in the network, and that the tree is fixed. Assume, furthermore, that all the operations can individually be performed by shift arithmetic with 1-bit delay. The computation can be performed

with only one active message in a similar way to the sorting algorithm. Conceptually, all even nodes perform the operations at height 1, all the nodes with indices divided by 4 perform the operations at height 2 and so on. The last node perform $\log_2 n$ operations ending with the root. In reality, all the operations above are pipelined, and most can be done concurrently. We omit the details of the algorithm.

5. APPLICATIONS

Consider first a multicomputer environment with a distributed operating system. In order to fully utilize such a multi-resource environment we would like to distribute work among the processors so that idle processors can help others with full load. One of the major problems in designing an effective load balancing mechanism is communication costs. We have identified this problem as the *transmission dilemma* of a load balancing strategy [LM82]. One aspect of this dilemma is how to provide all the processors with an updated picture of the instantaneous load distribution of the system. A number of studies have addressed the load balancing issue and several different answers to this dilemma have been developed [Li83, BS84, KF84]. Some of the proposed algorithms are based on frequent computation of the average load of the system, whereas others are based on a polling approach. Both approaches limit the rate at which information is being exchanged in order to maintain the level of processing and communication overhead due to load balancing activity at a 'reasonable' level. ATON provides an environment where both polling and average load computation can be performed in a very simple and efficient manner. Selecting the node with the maximum or minimum load as well as obtaining the average instantaneous load of the system requires only one ATON command. Therefore, the exchange of information regarding the load distribution of the system can be performed frequently and an up to date picture of the system load can be maintained by all processors.

A more specific problem of load balancing was encountered in the design of DIB - a Distributed Implementation of Backtracking [FM85]. DIB is a general-purpose package that allows applications that use backtrack or branch-and-bound to be implemented on a multicomputer. It is based on a distributed algorithm, transparent to the user, that divides the problem into subproblems and dynamically allocates them to the available machines. The package runs on the Crystal multicomputer at the University of Wisconsin-Madison. Any number of machines may be devoted to the application. The distribution of work in DIB is dynamic. When a machine M_1 finishes the work it was given it sends a ''request for work'' to another machine M_2. If M_2 is currently working it divides its work and sends part of it to M_1. The performance of the algorithm depends very much on the way the selection of who to ask for work is done. On the one hand we would like to assign M_1 higher priority work, but on the other hand we cannot afford to poll all machines every time work is sought. The following algorithm can be used with the ATON network.

Each piece of work is assigned (by the designer of the application) a priority. Each node maintains a queue of work to be done (organized as a heap). The highest priority of work on the queue is always put on an ATON register. This register is updated by the node whenever the highest priority work is changed. When a node looks for work it first sends an ATON message seeking the node with available work of highest priority. It then sends a regular ''request for work'' message to that node.

Another area of distributed computation that can benefit from the efficiency of the shift arithmetic approach is distributed simulation. In order to reduce the amount of storage utilized by a distributed simulation that runs on a *Time Warp machine*, the value of the *global virtual time* (GVT) has to be maintained by the system [DH85]. The GVT is used by each processor to release memory space occupied by useless information. Any information about events or messages with a time stamp smaller than GVT is useless and thus can be removed. If every processor maintains a copy of its timer in its operand register of the ATON interface, a processor can derive the current value of GVT by triggering the MIN command. The low overhead of such a command enables a frequent update of GVT and thus reduces the memory requirements of the simulation. Such a reduction

increases the applicability of the time warp approach.

Other applications of ATON include distributed operating systems (finding available resources), distributed databases (optimizations of distributed queries, recovery algorithms, etc.), and real-time applications (data collection).

6. EXTENSIONS

So far we assumed the existence of only one operand register. As a result only one application using ATON can be performed on the network. In general, it is possible to add a cache to the interface, such that every command includes the address of the appropriate operand in the cache. Fetching a word from a small cache can still be done within the 1-bit delay. This will enable several different applications to run at the same time, although not concurrently, in the network.

Some algorithms need to be executed frequently (e.g., finding the average load, number of active nodes). One way to do it is to have the node machines update the cache with the corresponding variables periodically. Then let the interfaces run these operations (by initiating the appropriate active messages) only when the network is not used for a while - for example, whenever a node sees the token twice in a row.

Another possibility to extend ATON is by allowing more than 1-bit delay for the shift arithmetic operations. As long as the delays are not cumulative their effect is not too significant. We found algorithms that can be improved with additional delays, for example, the majority algorithm.

7. CONCLUSIONS

We studied in this paper the feasibility of designing network protocols that allow efficient implementations of active messages. Active messages are simple commands that are performed on operands which are located at the network interfaces of the node machines. The new protocols make the communication channel together with the interfaces an environment in which simple computation can be carried out very efficiently. We have shown that these protocols can be implemented without a significant overhead and with little additional hardware, and that they enhance the performance of distributed algorithms on ring networks.

REFERENCES

[AS82]
D. W. Andrews and G. D. Schultz, "A Token Ring Architecture for Local Area Networks - An Update," *Proceedings of COMPCON Fall 1982*, Washington DC, September 1982, pp. 615-624.

[BS84]
A. Barak and A Shiloh, "A Distributed Load Balancing Policy for a Multicomputer," Department of Computer Science, The Hebrew University of Jerusalem, 1984.

[FM85]
R. A. Finkel and U. Manber, "DIB - A Distributed Implementation of Backtracking," To be presented at *the fifth International Conference on Distributed Computing Systems*, Denver, May 1985.

[FS82]
M. J. Fischer and S. L. Salzberg, "Finding Majority Among n Votes," Research Report

#252, Yale University, Department of Computer Science, October 1982

[JS85]
D. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism," *Distributed Simulation 1985*, Simulation Series, Vol 15, Number 2, January 1985.

[KF84]
P. Krueger and R. A. Finkel, "An Adaptive Load Balancing Algorithm for a Multicomputer," Technical Report #539, Department of Computer Science, University of Wisconsin, Madison, April 1984.

[Li78]
M. T. Liu "Distributed Loop Computer Networks," in *Advances in Computers*, M. C. Yovits (ed.), Academic Press, pp. 163-221, 1978.

[Li83]
M. Livny, "The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems," Ph.D. Dissertation, Weizmann Institute of Science, Rehovot, Israel, August 1983. Also available as Technical Report #570, Department of Computer Science, University of Wisconsin, Madison, December 1984.

[LM82]
M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Proc. ACM Computer Network Performance Symposium*, April 1982, pp. 47-55.

[Pr82]
Proteon Associates, Inc., "Operation and Maintenance Manual of PRONET UNIBUS HSB" August 1982.

[Ta81]
A. S. Tanenbaum "Computer Networks," Prentice-Hall, 1981.

[Th79]
Thomas R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," *ACM Transactions on Database Systems*, Vol 4, June 1979, pp. 180-209.