A Virtual Time CSMA Protocol for Hard Real Time Communication [1]

Wei Zhao
Krithi Ramamritham
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

We study a virtual time CSMA protocol for hard real time communication systems where messages have explicit deadlines. In this protocol, each node maintains two clocks: a real time clock and a virtual time clock. Whenever a node finds the channel to be idle, it resets its virtual clock to be equal to the real clock. The virtual clock then runs at a higher rate than the real clock. A node transmits a waiting message when the time on the virtual clock is equal to the latest time to send the message. This protocol implements the minimum-laxity-first transmission policy.

We compare the performance of our protocol with two baseline protocols both of which transmit messages according to the minimum-laxity-first policy. While both use perfect state information about the nodes and channel, the first is an idealized protocol which obtains this information without paying any cost and the second one pays a reasonable price for it. The simulation study shows that in most cases, our protocol performs close to the first one and better than the second one.

1 Introduction

This paper concerns communication protocols for distributed hard real-time systems. Tasks performed in such systems are time constrained. Consequently, the messages transmitted in the network often also have explicit time constraints such as deadlines. In this paper, we propose and evaluate a protocol which particularly takes messages' time constraints into account, and hence is suitable for the control of communications in distributed hard real-time systems.

[1]This material is based upon work supported in part by the National Science Foundation under grants DCR-8318776, and DCR-8500332, and by the Office of Naval Research under grant 048-716/3-22-85.

The most common communication network used in distributed hard real time systems is the multiple access network. In this type of network, stations transmit messages via a shared channel. Only one message can be successfully transmitted over the channel at any time. A *collision* occurs if at any time two or more messages are transmitted on the channel. No message can be received correctly in the event of a collision.

Based on how the collisions are handled, multiple access communication protocols can be broadly divided into three categories:

1. *Inference avoiding* protocols: This category includes ALOHA [1] and various CSMA protocols [9]. These protocols operate without taking past history of the channel into account.
2. *Inference seeking* protocols: Various tree, window, and stack protocols [2,4,5,7,12,27] and Urn protocols [10,28] belong to this category. These protocols make inference on the collision history, and usually solve collisions by partitioning some parameter space of messages.
3. *Deterministic* or *Collision-free* protocols: The Time Division Multiple-Access Protocols (TDMA), the Bit-Map Protocol [11], the Broadcast Recognition with Alternative Priorities Protocol [24,3,6], the Multi-level Multi-access Protocol [23], etc. are in this category. They work in such a way that collisions do not occur at all.

While we are currently developing protocols belonging to all three categories for hard real-time communication, in this paper we report on the performance of a protocol belonging to the first category.

In our model, each message has a deadline which specifies the absolute time by which the message must arrive at its destination; otherwise it is useless. The major performance metric in such a network is the *ratio of message loss*, i.e., the fraction of messages that do not arrive at their destinations by their deadlines. A good protocol for hard real-time communication should minimize this ratio.

The design objectives of multiple access protocols and scheduling algorithms are quite similar: Both are for allocation of a serially-used resource to a set of processes [13]. It is known from the theory of hard real-time scheduling that, in the static case, i.e., when all the task characteristics are known a priori, minimum-deadline-first and the minimum-laxity-first scheduling policies are optimal in the sense that they can schedule a set of tasks if there is some policy which can do so [15]. In the dynamic case, these policies also offer better performance than others [8]. Due to this fact, in our protocol, we adopt a network-wide transmission policy in which the message with the minimum laxity is transmitted first.

Kurose *et al.* suggest a window protocol for real time communication [12,13], implementing the minimum-laxity-first policy. They assume that the laxities of all the messages when they arrive are constant. Under this assumption, the minimum-laxity-first policy is identical to the first-come-first-served policy. Panwar *et al.* [20,21] have also studied the problem of optimal transmission policy. They assume that the length of the i-th message being transmitted on the channel is *independent of what the i-th message is.* In other words, the lengths of messages varies with the order in which they are transmitted. They prove that if the channel is not allowed to remain idle when there is a message waiting to be sent, the minimum-laxity-first policy is the best. In our protocol, all of above assumptions are removed: We allow messages to have arbitrary laxities, and also allow the message lengths to be determined at the time they arrive and hence to be invariant with the order in which they are transmitted.

In our protocol, each node maintains two clocks: a real time clock and a virtual time clock. Whenever a node finds the channel to be idle, it resets its virtual clock to be equal to the real clock. The virtual clock then runs at a higher rate than the real clock. A node transmits a waiting message when the time on the virtual clock is equal to the latest time to send the message which is equal to the message deadline minus the message length. This protocol implements the minimum-laxity-first transmission policy. Our protocol belongs to the newest class of CSMA protocols — virtual time CSMA [18,19,14]. We will have a brief discussion of the general virtual time CSMA in Section 3.

The remainder of this paper is organized as follows: Section 2 defines the model we adopt in this study. While Section 3 introduces our protocol, Section 4 presents the results of simulation studies comparing the performance of our protocol with that of two baseline protocols. Section 5 concludes the paper and suggests future work.

2 The Model

In a multiple access network, a set of nodes are connected to one communication channel. At any given time, only one message can be successfully transmitted over the channel. The maximum end-to-end delay for a bit is τ. We assume that the time axis is *slotted.* The length of a slot is defined to be one time unit. Given that the maximum end-to-end delay is τ, we let the length of a slot be equal to τ. A node can start transmitting a message only at the beginning of a slot. The length of a message is a multiple of the length of a slot. *The normalized end-to-end delay* is defined as

$$\alpha = \tau / M$$

where M is the mean message length. Previous work shows that CSMA protocols are applicable to environments where α has a small value such as 0.01.

Each message, M, can be characterized as follows:

- Length L_M which is the total number of time units needed to transmit message M;
- Deadline D_M. This is the time by which message M must be received by its destination;
- Latest time to Send the message, LS_M, is equal to $D_M - L_M$;
- Laxity at time t, $LA_M(t)$ is the maximum amount of time the transmission of message M can be delayed at time t. Therefore,

  $$LA_M(t) = D_M - L_M - t = LS_M - t.$$

  When it is clear from the context, we may omit argument t as well as subscript M in the above expressions.

From these definitions, it is clear that if we transmit tasks according to their latest start times, it is equivalent to the minimum-laxity-first transmission policy.

3 A Virtual Time CSMA Protocol: VTCSMA-L

Various Carrier Sense Multiple Access (CSMA) protocols have been proposed and evaluated [9]. Most of them may not be suitable for hard real-time communication because the message transmission delay cannot be bounded. Molle and Kleinrock [18,19] suggested a new class of CSMA protocols called *Virtual Time CSMA*. In this protocol, each node has two clocks. One clock gives the real time, and another gives the virtual time. The virtual clock stops running when the channel is busy, and runs when the channel is idle. When it runs, the virtual clock runs at a higher rate than the real one if it is behind the real clock. A message is sent only when its arrival time is equal to the time on the virtual clock. The major advantages of the VTCSMA protocol are its fairness in transmitting the waiting messages (since it is based on the FCFS policy), lower collision rate, and better delay-throughput behavior [14].

Below, we propose a virtual time CSMA protocol for hard real-time communication. We let the virtual clock run along the axis of messages' LS such that a network-wide minimum-laxity-first transmission policy is achieved. Because of this, we call this protocol *VTCSMA-L*. The following is the outline of our protocol:

1. Each node has two clocks, one clock maintaining the real time, and another maintaining the virtual time set in a manner discussed below.
2. Each message M is associated with a parameter *virtual latest time to start transmission, VLS$_M$*. When a message arrives, VLS$_M$ of message M is set to be LS$_M$. If the channel is idle and

   $VLS_{\text{New Message}} \leq$ the reading of the virtual clock,

   the new message is sent immediately; otherwise it waits. VLS$_M$ may be modified when the transmission of this message causes a collision. We will discuss the details of this modification in Step 5.
3. Each node senses the channel. Suppose that after either a successful message transmission or a collision, at real time t every node finds that the channel is idle:

   a. Every node resets its virtual clock to be equal to the real clock. The virtual clock starts to run at rate $\eta > 1$. η is a parameter of this protocol. We will discuss selection of η later. The virtual clock will stop when the channel is busy.

   b. Every node drops any waiting message if the LS of the message is less than the current time t, i.e., such a message is lost.
4. A node sends message M when VLS_M equals the reading of the virtual clock. Note that because the virtual clock runs faster than the real clock, the message is sent at (real) time

   $$t + (VLS_M - t)/\eta$$

   where $\eta > 1$. This time is earlier than the latest time to send message M, LS_M.
5. When a collision occurs during the transmission of a message, say at (real) time t', the sender node

   a. re-transmits this message *immediately* with probability P_i; or

   b. draws a number R randomly from the interval (t', LS_M), and modifies VLS_M as

   $$VLS_M = R,$$

   c. then, puts this message back in the queue of messages waiting to be transmitted.

The protocol above ensures that the message with the minimum VLS is transmitted first. By the definitions of VLS, LS and LA, we notice that this policy is equivalent to that of the minimum-laxity-first.

The following example helps to understand how the time on the virtual clock relates to the time on the real clock (see Figure 1).

EXAMPLE: Assume that at real time t_1 the channel is found idle, every node resets its virtual clock to be $t'_1 = t_1$. Then the virtual clock runs at a rate of $\eta > 1$. Say, two messages M_1 and M_2 are waiting to be sent with VLS$_{M_1}$ = t'_2 and VLS$_{M_2}$ = t'_4 respectively. When the virtual clock equals to t'_2, M_1 is sent out. This message is transmitted over the channel from the real time t_2 to t_3. During the transmission of M_1, the virtual clock does not run.

At real time t_3, because the channel is idle again, the virtual clock is set to be $t'_3 = t_3$. Then it starts running at rate η. When the virtual clock reaches t'_4, message M_2 is sent out. The channel is busy, and the virtual clock stops running. Say, at real time t_5, the transmission of message M_2 finishes. Then the virtual clock is set to the value of the real clock (so $t'_5 = t_5$), and starts running again.

From this example, it is clear that the virtual clock does not run *continuously*. At real time t_3, the virtual

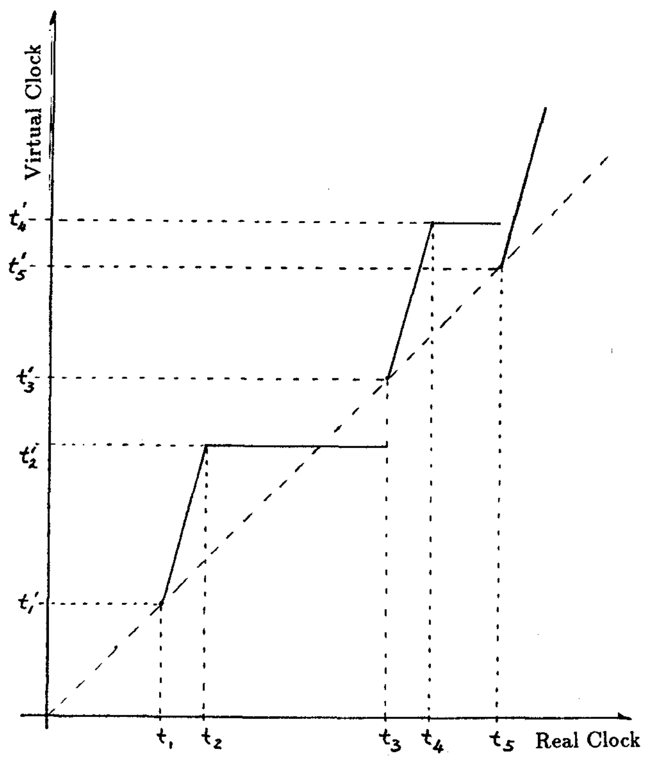


Figure 1: Virtual Clock vs. Real Clock

clock is set to equal the real clock. It is not necessary for the virtual clock to cover the interval (t_2', t_3') because if there is any message with its VLS in (t_2', t_3') this message has been lost by real time t_3. On the other hand, (t_5', t_4') is *twice* covered by the virtual clock. This is necessary because when message M_2 is being transmitted, some new messages may arrive with its VLS in that region and at real time t_5, there is still a chance to transmit it.

4 Performance Evaluation by Simulation

4.1 Base-line Protocols

We will compare the performance of VTCSMA-L with two baseline protocols. The first baseline protocol is called *Centralized Minimum Laxity message transmitted First*, abbreviated as CMLF. In this protocol, all the transmissions of messages are assumed to be scheduled by a centralized controller. This controller obtains perfect knowledge about the nodes and the channel without any communication overheads. It schedules the transmissions of messages such that the message with the minimum laxity is transmitted first. This algorithm is hence an ideal one, not realizable in practice. We use it purely to give us an upper bound on performance.

The second baseline protocol is based on the binary countdown protocol proposed by Mok and Ward [16]. We modify it as follows for use in hard real-time communication.

1. Let ML_i be the minimum laxity (> 0) of the messages waiting to be sent on node i. When each node senses the channel to be idle, if it has any message to be sent, it sends out the *complement* of the binary code of its ML_i one bit per slot in a *binary countdown* manner [16,26]. Briefly, the binary countdown method functions as follows: as soon as a node sees that a high-order bit position that is 0 in its ML_i has been overwritten with a 1 (by some other node(s)), this node gives up and does not send any more bits.
2. After a complete (complement code of) ML_i has been transmitted along the channel, each node knows what the minimum laxity among all the messages waiting to be sent is. Those nodes, which have messages with this minimum laxity, then send out their identification number in the same binary countdown manner.
3. The node, which has the message with the minimum laxity and has the minimum identification number, starts sending out the message.
4. Each node repeats the above procedure when the channel is idle again.

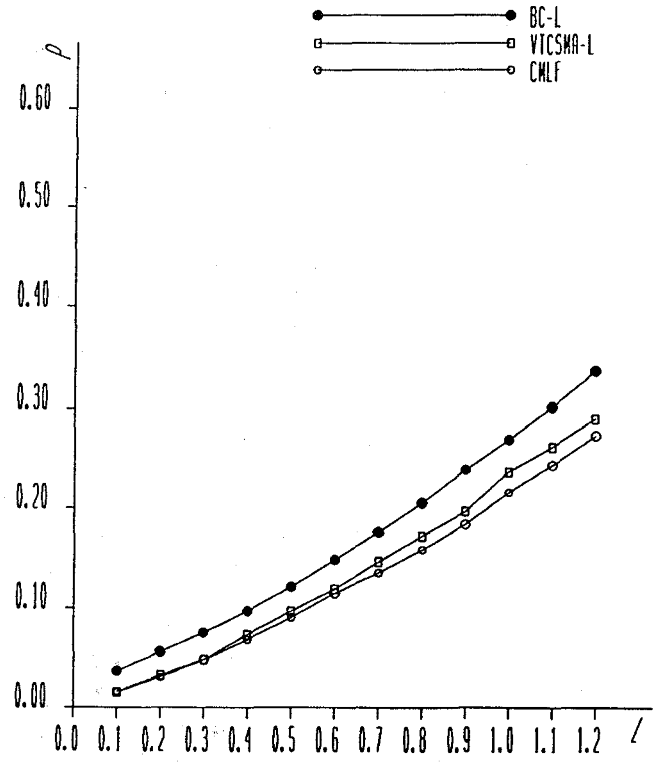
Because the "countdown" is done on laxities of messages, we call this baseline protocol *Binary Countdown on Laxity*, abbreviated as BC-L. Note that BC-L uses

$$\lceil \log_2(\text{Maximum Laxity}) \rceil + \lceil \log_2(\text{Maximum Node Identification Number}) \rceil$$

time slots to decide which message is to be transmitted during each cycle of transmission.

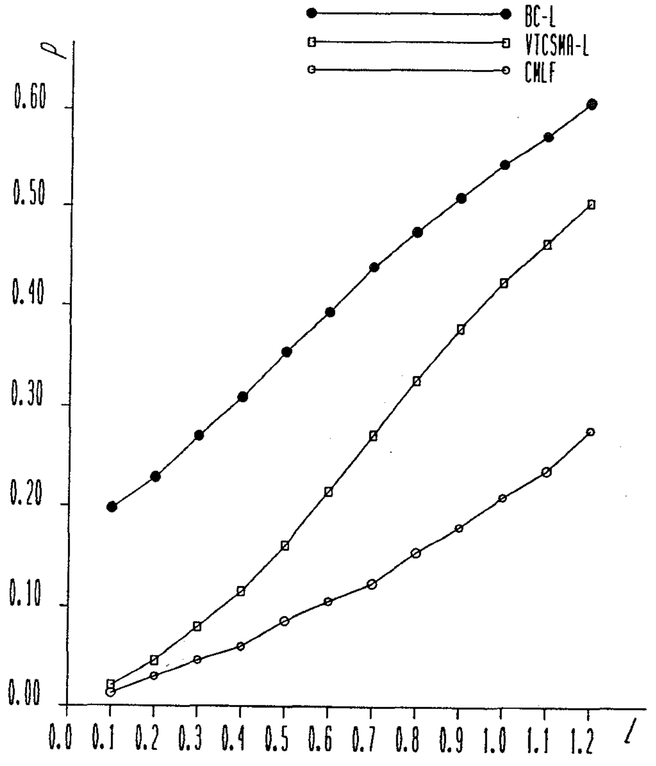
4.2 Simulation Results

We simulate 3 cases for a network of 10 nodes. The first two cases are for stochastic messages and the last one is for periodic messages. The simulation program is written in Simscript, and runs in an ULTRIX environment on MicroVAX-II. For simplicity, in all the simulations, the probability of immediate re-transmission, P_i, takes a value of 0.5. We will discuss extension to this in the last section.



Case 1: $\alpha = 0.01, \eta = 27.0$

Figure 2: Message Loss vs. System Load



Case 2: $\alpha = 0.1, \eta = 4.0$

Figure 3: Message Loss vs. System Load

4.2.1 Stochastic Messages

In Cases 1 and 2, messages arrive at each node as a Poisson process. The arrival rate at each node is the same. Message lengths are exponentially distributed with a mean of 100 in Case 1 and 10 in Case 2. The laxities of messages are randomly chosen from [0, 600] in Case 1 and [0, 60] in Case 2. Recall that the end-to-end delay τ is the unit of time. Thus, in Case 1, $\alpha = 0.01$, and in Case 2, $\alpha = 0.1$. [2] In each case, we simulate the network with system load from 0.1 to 1.2 where the system load (L) is defined as

$$L = AR * M * N$$

where AR is the mean of the message arrival rates at each node, M is the mean of the message length, and N is the number of nodes in the network. For the simulation, N = 10, M = 100 in Case 1, and = 10 in Case 2. AR is adjusted to get different values for L.

Figures 2 and 3 show the ratios of message loss (R) vs. the system loads (L) for Cases 1 and 2 respectively. From these figures, we can make the following observations:

1. Message loss in CMLF and BC-L tends to grow almost linearly with increase in system load.

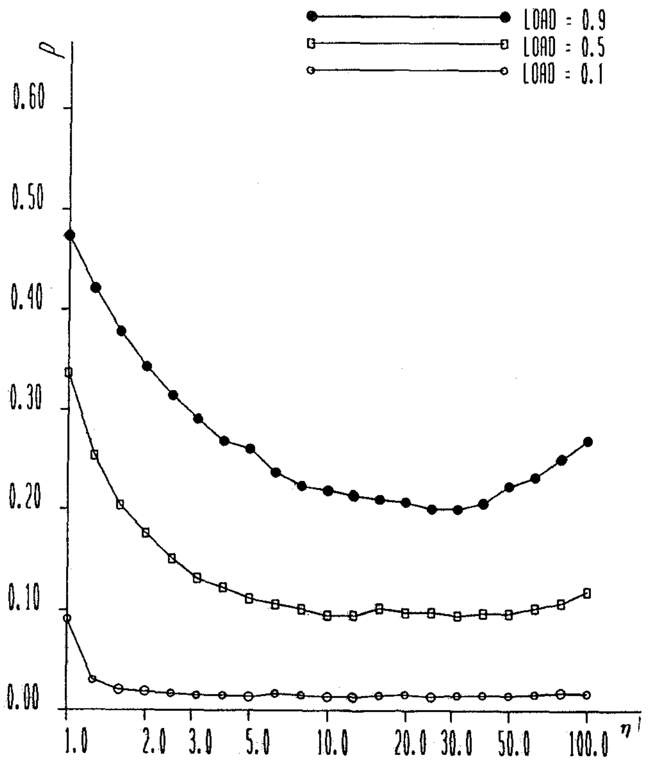
[2]We note that $\alpha = 0.01$ is considered to be the typical application environment for CSMA [19].

2. When α is small, e.g., equal to 0.01, the performance of VTCSMA-L is almost identical, or very close to that of CMLF.

3. When α is large, e.g., equal to 0.1, the performance of VTCSMA-L is between that of CMLF and BC-L. When the load is not too high, say, lower than 0.7, the performance of VTCSMA-L is closer to that of CMLF than to BC-L.

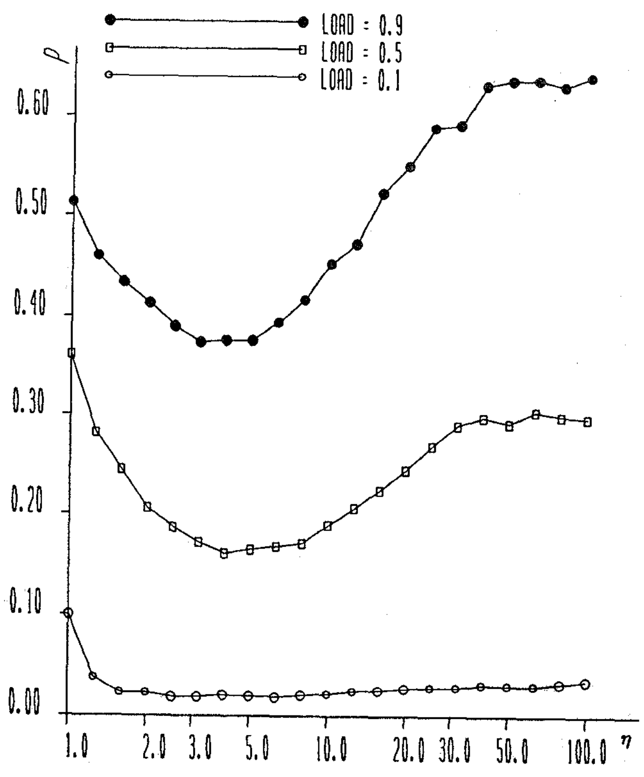
4. In any case, VTCSMA-L is definitely better than BC-L.

Selection of η is an important issue for the application of VTCSMA-L. For each value of α, we test the system performance over different values of η with three different loads. Figures 4 and 5 show the simulation results. We see that when the system load is light (= 0.1), or when α is small (= 0.01). For a relatively wide range of η values, message loss is very close to the minimum. This indicates that the system performance is not sensitive to the values chosen for η under these situations. This observation conforms with that made by Molle for the original VTCSMA [18,19], though performance metrics of his VTCSMA and our VTCSMA-L are different.



Case 1: $\alpha = 0.01$

Figure 4: Message Loss vs. η



Case 2: $\alpha = 0.1$

Figure 5: Message Loss vs. η

4.2.2 Periodic Messages

Case 3 is for periodic messages. In hard real time systems, messages are often *periodic*, i.e., they arrive and request to be transmitted periodically. These messages can be input/output data to/from application tasks as well as control information, such as the node surplus information, passed among nodes [17,22,29].

If a system has only periodic messages and all message characteristics are known a priori, then a pre-determined schedule may be used to control the transmissions of messages. However, if both periodic and stochastic messages exist, or if characteristics of periodic messages may change at run-time, then a dynamic scheme, such as BC-L or VTCSMA-L has to be used.

For simplicity, we simulate a system with periodic messages only. The simulation model is set as follows: The system has 10 nodes. Time is divided into periods. The length of a period is 1150 time units. In each period, one message arrives at each node. The message length is 100 time units. A message has deadline equal to the end point of the period within which it arrives. The reason for this choice of parameters is that if the messages at all the nodes arrive at the beginning of a period, then CMLF and BC-L will be able to transmit all the messages.[3] Note that all the messages in a period have the same laxity. This is a worst-case situation for VTCSMA-L because a collision definitely occurs when there are two or more newly arrived messages. The objective of this simulation is to verify the viability of VTCSMA-L under this situation.

Protocols	b = 0	b = 575	b = 1035
BC-L	0.00	0.11	0.18
VTCSMA-L	< 0.01	0.06	0.18
CMLF	0.00	0.01	0.11

Table 1: Ratios of Message Loss for Periodic Messages

For a given period, P, we assume that the messages belonging to this period may arrive in the interval $[a, a+b]$ where a marks the begin point of P and b is a simulation parameter that is changed for different simulations. Table 1 lists the values of b and the corresponding ratios of message loss for the three protocols.

[3]Recall that there are 10 messages in a period (one message per node). Total transmission time for 10 messages is 1000 time units. To transmit a message, BC-L takes 15 time units for the "binary countdown". Thus, the minimum length of a period is 1150.

From Table 1, we notice that when $b = 0$, i.e., every message belonging to a period arrives at the beginning of the period, VTCSMA-L cannot guarantee that all the messages will be transmitted. However, BC-L and CMLF can. This is due to the probabilistic nature of VTCSMA-L. However, the amount of messages lost is very small (< 0.01). Hence, in a system where this amount of loss is tolerable, VTCSMA-L can be used. When $b = 575$, i.e., all the messages in a period arrive in the first half of the period, the performance of VTCSMA-L is better than BC-L. When $b = 1035$, i.e., messages can arrive at any *feasible time* within a period, [4] VTCSMA-L and BC-L have very similar performance. Thus, in general, for the transmission of periodic messages, we see that VTCSMA-L performs at least as well as BC-L.

5 Conclusions

We have proposed a multiple access protocol for hard real time communication. We have compared the performance of our VTCSMA-L protocol with CMLF and BC-L. The reader may note that these three protocols are all intended to ensure network-wide minimum-laxity-first policy. But the costs are different. CMLF is the idealized protocol with no costs. BC-L takes

$$\lceil \log_2(\text{Maximum Laxity}) \rceil + \lceil \log_2(\text{Maximum Node Identification Number}) \rceil$$

time slots for each message transmission. VTCSMA-L achieves the minimum-laxity-first policy in a different manner. When the channel is found to be idle, whereas both CMLF and BC-L transmit a waiting message immediately, VTCSMA-L would let the channel idle for a while (until the virtual clock matches the minimum laxity of messages waiting to be sent). This has a two-fold effect: Recall that when the channel is found to be idle, all the nodes reset their virtual clocks.

1. While the channel is idle, a message with a laxity less than that of waiting messages may arrive and has a chance to be transmitted. If the channel were used, this newly arrived message might be lost.

2. On the other hand, some of waiting messages (not the one with the minimum laxity) may be lost due to the fact that the channel is left idle, i.e., is not being fully utilized.

[4]Note that the length of a period is 1150 and that BC-L takes 115 time units to transmit a message. Therefore if a message in a period arrives at time after $a + 1035$, BC-L definitely has no chance to transmit it.

The simulation results shown in the last section indicate that in the domain we tested, for the case of stochastic messages, the performance of VTCSMA-L is always better than BC-L, and is most likely closer to the performance of CMLF. For the case of periodic messages, i.e., in an extremely unfavorable case, VTCSMA-L performs at least as well as BC-L most of the time.

The implementation of the protocol requires the synchronization of clocks of all nodes. In a distributed system, clock synchronization is an interesting and challenging problem. However, our protocol is robust in the sense that the network will continue functioning even if clocks are not perfectly synchronized. Of course, in this situation, the performance of the network may degrade because the minimum-laxity-first transmission policy cannot be guaranteed.

The work reported in this paper is preliminary. Many extensions are possible. VTCSMA-L should easily extend to the un-slotted channel. If the exact number of nodes which are involved in a collision can be known (say, as N_c) [27], P_i, the probability of immediate re-transmission after a collision may be set dynamically to improve the performance. When N_c is larger, P_i should be smaller so that the overhead of collision resolution may be reduced. On the other hand, if N_c is small, P_i may take a larger value to increase the chance that a node finally gets the right to transmit a message. More generally, we are currently studying other protocols for use in distributed dynamic hard real time computer systems.

References

[1] N. Abramson, "The ALOHA System — Another Alternative for Computer Communications", *AFIPS Proceedings of Fall Joint Compu.*, Vol. 37, 1970.

[2] J. I. Capetanakis, "Tree Algorithm for Packet Broadcast Channels", *IEEE Transactions on Communications*, Vol. Com-27, No. 10, October 1979.

[3] I. Chlamtac, W. R. Franta, and D. Levin, "BRAM: The Broadcast Recognizing Access Method", *IEEE Transactions on Communications*, Vol. COM-27, No. 8, August 1979.

[4] G. Fayolle, P. Flajolet, M. Hofri, and P. Jacquet, "Analysis of a Stack Algorithm for Random Multiple-Access Communication", *IEEE Transaction on Information Theory*, Vol. IT-31, No. 2, March 1985.

[5] A. Grami, K. Sochraby, and J. Hayes, "Further Results on Probing", *Proceedings of the IEEE International Communications Conference*, 1982.

[6] L. W. Hansen and M. Schwardtz, "An Assigned-Slot Listen-Before-Transmission Protocol for a Multiaccess Data Channel", *IEEE Transactions on Communications*, Vol. COM-27, No. 6, June 1979.

[7] J. F. Hayes, "An Adaptive Technique for Local Distribution", *IEEE Transactions on Communications*, Vol. Com-26, No. 8, August 1978.

[8] E. D. Jensen, C. D. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of IEEE Real-Time Systems Symposium*, December 1985.

[9] L. Kleinrock, "Packet Switching in Radio Channels: Part 1 — Carrier Sense Multiple-Access Modes and their Throughput-Delay Characteristics", *IEEE Transactions on Communications*, Vol. COM-23, No. 12, December 1975.

[10] L. Kleinrock and Y. Yemini, "An Optimal Adaptive Scheme for Multiple Access Broadcast Communication", *Proc. ICC*, 1978.

[11] L. Kleinrock and M. O. Scholl, "Packet Switching in Radio Channels: New Conflict-free Multiple Access Schemes", *IEEE Transactions on Communications*, Vol. Com-28, No. 7, July 1980.

[12] James Francis Kurose, Mischa Schwartz, and Techiam Yemini, "Controlling Window Protocols for Time-Constrained Communication in a Multiple Access Environment", *Proceeding of the 8th IEEE International Data Communication Symposium*, 1983.

[13] J.F. Kurose, "Time-Constrained Communication in Multiple Accesses Networks", *Ph.D. Dissertation, Columbia University*, 1984.

[14] J.S. Meditch and D. H. Yin, "Performance Analysis of Virtual Time CSMA", *Proceedings of IEEE Infocom'86*, April 1986.

[15] A.K. Mok, and M.L. Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment",*Proc. of the Seventh Texas Conference on Computing Systems*, November 1978.

[16] A.K. Mok and S. A. Ward, "Distributed Broadcast Channel Access", *Comput. Network*, Vol. 3, November 1979.

[17] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment",*Ph.D. Dissertation*, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983.

[18] M.L. Molle, "Unifications and Extensions of the Multiple Access Communications Problem" *Ph.D. Dissertation*, University of California at Los Angeles, July 1981.

[19] M.L. Molle and Lenonard Kleinrock, "Virtual Time CSMA: Why Two Clocks are Better than One", *IEEE transactions on Communications*, Vol. COM-33, No. 9, September 1985.

[20] S. Panwar, D. Towsley, and J. Wolf, "Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service" *submitted for publication.*

[21] S. Panwar "Time Constrained and Multiaccess Communications" *Ph.D. Dissertation*, University of Massachusetts at Amherst, February 1986.

[22] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Hard Real-Time Tasks under Resource Constraints in the Spring System", submitted to *IEEE Transactions on Computers*, November 1985.

[23] E. H. Rothauser and D. Wild, "MLMA — A Collision-Free Multi-Access Method", *Proc. IFIP Congr. 77*, 1977

[24] M. Scholl, "Multiplexing Techniques for Data Transmission over Packet Switched Radio Systems", *Ph. D. Dissertation*, University of California at Los Angeles, 1976.

[25] J.A. Stankovic, "A Perspective on Distributed Computer Systems", *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984.

[26] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., 1981.

[27] D. Towsley and G. Venkatesh, "Window Random Access Protocols for Local Computer Networks", *IEEE Transactions on Computers*, Vol. C-31, No. 8, August 1982.

[28] Y. Yemini, "On the Channel Sharing in Discrete-Time Packet Switched, Multiaccess Broadcast Communication", *Ph.D. Dissertation*, University of California at Los Angeles, 1978.

[29] W. Zhao, "A Heuristic Approach to Scheduling Hard Real-Time Tasks with Resource Requirements in Distributed Systems", *Ph.D. Dissertation*, The University of Massachusetts at Amherst, 1986.