The Best-Effort Virtual-Time CSMA/CD Protocols for Real-Time Systems

Sam K. Oh
Glenn H. MacEwen

November 2, 1992
External Technical Report
ISSN-0836-0227-
92-343

Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6

Document prepared November 5, 1992

Abstract

Two *clairvoyant* virtual-time CSMA/CD protocols for dealing with time-constrained messages are described. A clairvoyant protocol takes into account not only the messages currently available for transmission but also the messages expected to be produced in the future. In the absence of clairvoyancy, the urgent but currently unavailable messages may not be successfully transmitted when they are actually produced. Simulation results show that these two clairvoyant protocols yield better performance over two other virtual-time protocols with no clairvoyancy within a reasonable range of system loads. The percentage of message timing failures is used as the primary performance measure.

Keywords: Real-Time. Communication Protocols. Clairvoyancy. Timing failures.

Contents

1 Introduction

Real-time computer systems are required by their environments to produce correct results not only in their values but also in the times at which the results are produced. To satisfy such *timeliness* requirements, the execution times of program units called *real-time tasks* are constrained. Each task *timing constraint* can be either a *delay*, a *deadline*, or both. A delay constraint on a task specifies a point in time after which the task must complete, while a deadline constraint specifies a point in time before which the task must complete.

For increased performance and reliability, distributed architectures having multiple nodes interconnected by a communications network are increasingly used for real-time systems. In such real-time distributed systems, since tasks running on different nodes interact by sending and receiving messages via the network, the timely delivery of messages is essential for meeting the timing constraints of the tasks. In other words, the delivery times of messages must also be constrained. Message delivery times can be constrained in a way similar to task execution times, by either a delay, a deadline, or both. A delay constraint on a message specifies a point in time after which the message must be delivered, and a deadline specifies a point in time before which the message must be delivered.

A class of networks frequently used for real-time applications is the *multiple access network* in which nodes, and hence tasks, share a single communication channel. In this class of network, only a single message can be successfully transmitted over the channel at any one time. If two or more messages are simultaneously transmitted on the channel, none will be correctly delivered to their destination due to a *collision.* Therefore, such systems must have synchronization methods called *protocols* for controlling the transmission of messages [PZ78].

Based on how collision is handled, protocols for multiple access networks can be categorized into *controlled-access* and *contention-based* protocols [KSY84]. The controlled-access protocols are characterized by *collision-free* access to the channel; they allow each node to access the channel without collision. Time division multiple access (TDMA) and token passing protocols are examples. On the other hand, contention-based protocols allow nodes to transmit messages simultaneously over the channel; hence, collisions can occur. Various carrier-sense-multiple-access (CSMA) protocols with/without collision detection (CD) belong to this category. The multiple access protocols in all of these classes serve different application areas, and hence are valuable in different parts of the requirement space. However, many of the protocols in these classes are not intended for real-time systems.

We call protocols devised to deal with time-constrained messages *real-time communication protocols* (or simply real-time protocols). Figure 1 shows a typical communication between two nodes in a real-time system. At the sending node, messages produced by any one of the set A of application tasks are placed into queue Q; the network access protocol P at the sending node decides which message should be transmitted first. Messages whose delivery times cannot meet timing constraints are discarded. When a message is received, the protocol P at the receiving node discards the message if it is corrupted or its delivery time cannot meet a certain specified time limit.

In real-time systems, a message expected to be produced in the future can be

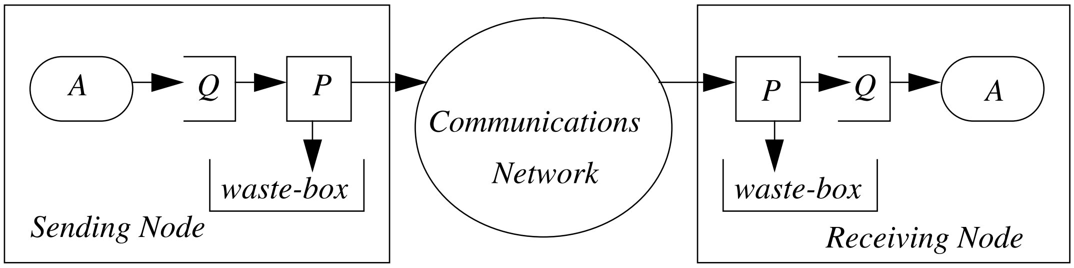

Figure 1: Real-Time Communication in A Multiple Access Network

more urgent than those already available for transmission. Such an urgent *future message* may not be delivered by its deadline if less urgent but available messages are transmitted first. In addition, a message transmitted too early may result in a collision with another more urgent message from another node. If a communication protocol can recognize and consider urgent future messages when choosing a message for transmission, these undesirable and even harmful situations may be avoided or minimized. A protocol with such an ability is referred to as a *clairvoyant* protocol. Recently, several real-time CSMA/CD protocols for messages with deadline constraints have been introduced in the literature [KSY83, PTW88, ZR87, ZSR90]. However, all of these protocols have no such clairvoyancy. Two clairvoyant real-time CSMA/CD protocols are introduced in this paper; these protocols make use of the concept of the virtual clock [MK85, ZR87].

1.1 Task and Message Characteristics

Task scheduling algorithms provide a basis for real-time protocols. Tasks can be categorized into either *periodic tasks* and *aperiodic tasks*. A periodic task is *activated* repetitively at fixed intervals of time, while an aperiodic task is activated in response to repeated asynchronous events. Task execution times are, in general, stochastic. For tasks to meet deadlines, it is usual to consider estimated worst-case execution times of the tasks.

It is known from the theory of task scheduling that the *earliest deadline first* (EDF) and the *least laxity first* (LLF) scheduling algorithms are *optimal* [1] in a uniprocessor system in which the configuration is *static*, i.e. all the task characteristics are known *a priori*. In practice, there are many *dynamic* applications in which sufficient knowledge regarding the task characteristics cannot be obtained *a priori*. Even in such dynamic environments, the EDF and LLF policies have been shown to yield better performance than others [JLT85].

There is an analogy between multiple access real-time protocols and task scheduling algorithms for uniprocessors because, in both cases, a single resource is being allocated. Based on this analogy, several real-time protocols adopt a policy that approximates the EDF or the LLF policy [KSY83, PTW88, ZR87, ZSR90]. We call a message that must be transmitted first the *most urgent message*. A message with

[1] An optimal scheduler is one which may fail to meet a deadline only if no other scheduler is guaranteed to succeed [DM89].

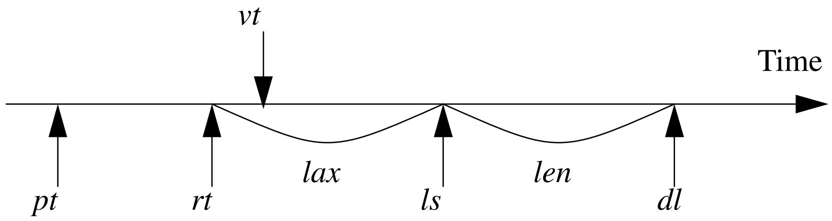

Figure 2: Various Times and Parameters Associated with A Message

the least laxity becomes the most urgent message in a protocol using the LLF policy, and a message with the earliest deadline becomes the most urgent in a protocol using the EDF policy. We use the following terminology:

- The *real time* (sometimes called current time), *rt*, is a value read from a local physical clock.
- The *production time* (sometimes called arrival time) of message m, *pt(m)*, is the clock time at which m is available for transmission.
- The *virtual-time*, *vt*, is a value read from a virtual clock. The usage of *vt* is described later in this section.
- The deadline of message m, *dl(m)*, is the point in time by which the message must have been delivered to its destination.
- The *length* of message m, *len(m)* is the total number of message bits in m, i.e. the total number of time units needed to transmit the message without collision. It is also referred to as *message transmission time.*
- The *latest sending time* of message m, *ls(m)*, is the latest time for guaranteeing timely delivery; that is, $dl(m) - len(m)$.
- The *laxity* of message m at time *rt*, *lax(m)*, is defined to be $(dl(m) - len(m)) - rt = ls(m) - rt$. A negative laxity implies failure to meet the deadline. A message is said to be *tardy* when its laxity is negative.

When clear, we may omit argument '*rt*' as well as '*m*' in the above expressions. Figure 2 illustrates the relationships among the various times and parameters associated with a message.

1.2 Real-Time Protocols Review

Kurose *et al* [KSY83] introduce a protocol in which each message is required to be sent within a fixed time interval, K time units from its production time. Since messages produced earlier will always be more urgent than those produced later, a message that comes first will be transmitted first. Panwar *et al* in their protocol [PTW88] assume that a message is constrained by a time interval that is a random variable with general distribution. Also, message lengths are taken to be a sequence of independent and identically distributed random variables. With these assumptions, the lengths of messages vary with the order in which they are transmitted. However, in real-time systems messages produced later may be more urgent than those produced earlier, and message lengths, being generally fixed at design-time, are invariant with the order in which they are transmitted. For these reasons, we exclude these two protocols from our discussion.

In conventional window-based CSMA/CD protocols, a *window* defines an interval on the axis of some message parameter (e.g. production time, deadline, latest sending time). When a collision occurs, the window is split into halves, and each node treats messages in the left half first and in the right half next. When a new message is produced in the left half while the right half is being treated, the message is not considered for transmission. Zhao *et al* [ZSR90] introduce a new window protocol *WL* in which a window is formed on the latest sending time axis, and messages are queued at each node according to their latest sending times. The deadline of a message in this protocol, instead of being relative to its production time, is specified as a point in time before which it must be delivered. Also, messages have fixed lengths that are invariant with the order in which they are transmitted, and are allowed to have arbitrary laxities. One major difference between this protocol and conventional window-based protocols is that a newly produced message can be considered for transmission by letting the window lower-bound be the current time.

Molle and Kleinrock introduce a virtual-time CSMA/CD protocol (VTCSMA) [MK85] in which a virtual clock at each node runs along the arrival time axis. Zhao and Ramamritham [ZR87] generalize this protocol by allowing four different message parameters, i.e. transmission time, arrival time, latest sending time, and deadline, to be associated with the axis along which a virtual clock runs. We name these protocols *VTT*, *VTA*, *VTL*, and *VTD* respectively, using the obvious acronyms. In these protocols, each node maintains two clocks: a real clock and a virtual clock. The real clock runs at unit speed, while the virtual clock is allowed to run at a higher rate than that of the real clock. Among these four protocols, the *VTL* and *VTD* protocols have been shown to be suitable and hence recommended for real-time systems [ZR87]. These two protocols assume the same message characteristics as those in *WL*. Messages are queued according to the increasing order of their latest sending times in *VTL* and of their deadlines in *VTD*. An outline of these two protocols follows:

- When a message is produced, a virtual time parameter *VS* is initialized to *ls* in *VTL* , and to *dl* in *VTD*.

- When the channel has been continuously idle, a node finds the most urgent message, for which $VS \leq vt$, and transmits it.

- When the channel is sensed busy, the virtual clock does not run.

- The channel will become idle either after a successful transmission or a collision. At this point, all tardy messages are discarded from the queue. The virtual clock is reset to the value of the real clock and starts to run again. A node finds a most urgent message, for which $VS = vt$, and transmits it.

- When there is a collision, the sender node retransmits the message with probability P, or it modifies VS for the message by drawing from the interval (rt, ls) in VTL and from the interval (rt, dl) in VTD. Then, the message is put back into the message queue.

1.3 Real-Time Protocols: Clairvoyancy

In a real-time distributed system with a dynamic environment, being clairvoyant may be very difficult for a task scheduler, but may be possible for a communication protocol. This is mainly because the tasks producing messages are scheduled within the system. Consider a task that produces a message. Assume that deadline and length of the message are known when the task is activated. Even though the message is not yet produced (i.e. it is future message), we can still obtain the message's latest sending time which can be used as the worst-case message production time. The message may be produced earlier than the latest sending time (hence than the deadline), because the actual execution time of the task may be significantly less than its estimated worst-case execution time.

The protocols reviewed above do not have clairvoyancy with respect to future messages. In this paper, we introduce two virtual-time based protocols: one based on the messages' latest sending times and the other based on the messages' deadlines. We denote the former by $BVTL$ and the latter by $BVTD$. In the sense that these two protocols attempt to approximate a basic policy as closely as possible and to transmit as many messages as possible successfully, we call them *best-effort* protocols. In these two best-effort protocols, the deadline and length of a message are assumed to be known at the instant at which the task producing the message is activated. With this assumption, they can be clairvoyant. They attempt to reserve the communication channel for a future message if (1) it is the most urgent one and (2) it cannot be successfully transmitted when a less urgent but currently available message is transmitted first. Tardy messages are simply discarded.

The remainer of this paper is organized as follows: Section 2 defines the system model. Section 3 introduces our best-effort protocols. Section 4 presents our simulation results. Section 5 concludes the paper.

2 The Model

CSMA/CD protocols can operate in either an *asynchronous* (unslotted) or a *synchronous* (slotted) mode. In asynchronous mode, each node runs a copy of the protocol independently using local observations of the channel. In synchronous mode, all nodes run their local copies of the protocol in lock-step; the time axis is divided into a sequence of time units, called *slots*, and each node can transmit messages only at the beginning of each slot.

The best-effort protocols are assumed to operate in synchronous mode. We also assume that each node has a well-synchronized local clock or there is a system-wide time reference that ticks every slot. The maximum end-to-end propagation delay for a bit is τ. A slot length is defined to be equal to τ. A message length is a multiple of slot length. Each message is characterized by: deadline, length, latest sending time, and laxity. We assume that a message's deadline and length are known at the instant its producer task is activated. The latest sending time and the laxity of a message are calculated from the deadline and the length of the message.

In the best-effort protocols, each node maintains two queues:

- A *future queue* Q_F that contains future messages.
- A *current queue* Q_C that contains messages that have been produced and are not tardy.

Messages are queued according to the increasing order of their latest sending times in $BVTL$ and of their deadlines in $BVTD$. We say that a future message is *recognized* when the task that will produce the message [2] is activated. When a future message is recognized, corresponding message characteristics are queued into Q_F. When the future message is actually produced and not tardy, it is queued into Q_C. Tardy messages are simply discarded from Q_C. We assume that the activation time of any task cannot be equal to or later than the latest sending time of any message it produces.

3 The Best-Effort Virtual Time CSMA/CD Protocols

3.1 Problems Addressed

Several problems in the VTL and VTD protocols can be identified: These problems may occur only rarely, depending upon the characteristics of the application tasks. However, the impact may be significant for particular applications.

- A collided message may become tardy. Since message tardiness is not checked at the time of collision, a collided and tardy message can be retransmitted.
- At the time of a collision, there may be a message that is newly produced and the most urgent. Such a message is not considered for transmission.
- Both the VTL protocol and the VTD protocol may waste one time-unit when the channel becomes idle after the successful transmission or the collision of a message. As a result, a message that could be successfully transmitted, can become tardy. For example, consider a message m with $ls = 300$ and $dl = 330$. Assume that the virtual clock runs at 35 times the rate of of the real clock (i.e. $\eta = 35$). It collides at $rt = 299$ and is queued back with modified VS (say, $VS = 299$ in VTL and $VS = 325$ in VTD). Assume an idle state at $rt = 300$ (after the collision). In this state, each protocol discards tardy messages and sets vt to equal rt; if a message whose $VS = vt$ is found, it is transmitted. Since m has $VS < vt$ ($VS = 299$) in VTL and $VS > vt$ ($VS = 325$) in VTD, it cannot be transmitted and becomes tardy in the next idle state (at $rt = 301$). By letting the virtual clock advance η (one tick of the real clock) and modifying "$VS = vt$" into "$VS \leq vt$", m may be successfully transmitted. Also, message tardiness needs to be checked at the continuous channel idle state to prevent transmission of tardy messages.

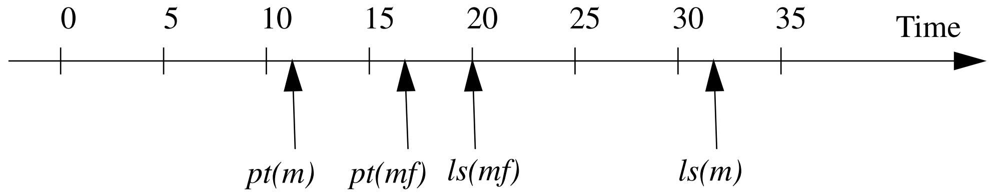

Figure 3: Message Production and Latest Sending Times

In addition to these problems, the virtual clock may greatly exceed the real clock when the system is lightly loaded. As a consequence, a message may be transmitted unnecessarily early, which may in turn aggravate system performance.

Consider the example shown in Figure 3. The channel is initially idle at $rt = vt = 0$. The virtual clock starts to run at three times the rate of the real clock ($\eta = 3$). Assume that message m with $ls = 32$ and $len = 10$ is produced at $rt = 12$ ($vt = 36$). Since $VS(m) < vt$, the VTL protocol starts transmission of m; the virtual clock stops. The transmission of m completes at $rt = 22$. Assume that another message *mf* with $ls = 20$ and $len = 5$ is produced during the transmission of m, say at $rt = 17$. Since $ls(mf) = 20$, *mf* becomes tardy and is discarded. If the protocol has clairvoyancy about *mf*, it could transmit both m and *mf* successfully by delaying m's transmission until $rt = 20$ (because $ls(mf) = 20$). A similar situation can occur in the VTD protocol.

3.2 Protocol Descriptions

In the best-effort protocols, each node maintains two clocks: a virtual clock and a real clock. The virtual clock runs at a rate η times faster than that of the real clock. While the channel is busy, the virtual clock stops. When a message is newly produced, the virtual parameter *VS* is initialized to *ls* in $BVTL$ and to *dl* in $BVTD$. As shown in Table 1, there are five possible protocol states. The actions taken in each of these states are as follows:

- State II:
  The virtual clock runs continuously. Tardy messages are discarded. If there exists a most urgent message m in Q_C whose $VS \leq vt$, there are three cases:

  C1: There exists no most urgent future message. m is transmitted.

  C2: There exists a most urgent future message *mf*, and *mf* can be delivered by its deadline even after m's transmission (i.e. $lax(mf) > len(m)$). m is transmitted.

  C3: There exists a most urgent future message *mf* which cannot be successfully transmitted. Check whether or not m can be successfully transmitted after waiting for and transmitting *mf*. There are two subcases:

[2] Multiple messages can be produced by a task. In such a case, the messages are queued according to their latest sending times or deadlines.

Protocol State	Channel State at Current Slot	Channel State at Previous Slot
II	idle	idle
BI	idle	busy
CI	idle	collision
BSY	busy	
COL	collision	

Table 1: Protocol States

C3a: m can be transmitted (i.e. $lax(m) > lax(mf) + len(mf)$). Wait until *mf* is produced. (We discuss two waiting strategies later in this section.)

C3b: Otherwise, transmit m.

- States BI and CI:
  The virtual clock is set to equal $rt + \eta$. Tardy messages are discarded. If there is a most urgent message m whose $VS \leq vt$, there are four cases corresponding to cases C1, C2, C3a, or C3b.

- State BSY:
  A node that is transmitting a message continues its transmission. Other nodes wait for the completion of this transmission.

- State COL:
  If a collided message becomes tardy, it is discarded. Otherwise, each node checks whether there exists a newly produced most urgent message. If so, the collided message is put back into the message queue Q_C. The associated *VS* parameter is modified as in the *VTL* and *VTD* protocols.

  With the most urgent message m (it could be a collided message or a newly produced message), check whether or not there exists a future message that will be the most urgent when produced. There are four cases corresponding to cases C1, C2, C3a, or C3b. In cases C1, C2 and C3b, m is either transmitted with retransmission probability P, or put back into the message queue Q_C. In case C3a, m is put back into the queue, and a suitable waiting action is taken (See below.).

Two waiting strategies can be used for the most urgent future message *mf*:

- a *local waiting strategy.*

- a *global waiting strategy.*

With the former, a node waits for *mf* locally, while all the nodes wait for *mf* with the latter. With local waiting, a node simply waits for *mf* without sending any message.

With global waiting, a node transmits a *synchronization message* with the shortest length (i.e. one-bit message). When the channel is sensed idle after the successful transmission or collision of the synchronization message, each node resets its virtual clock to equal the value of the real clock+η. As a result, the most urgent future message has a chance for transmission. The *ls* value of a synchronization message is taken to be equal to rt. Consequently, when it has a collision, it becomes tardy and is discarded. The best-effort protocols use local waiting only when *lax(mf)* is one. Otherwise, the global waiting strategy is used.

4 Performance Analysis

4.1 Simulation Model

A simulation model was used to evaluate the performance of the best-effort protocols, parameterized by the number of nodes and the distributions of message production times, lengths and laxities. Messages are produced at each node as a Poisson process. That is, the inter-production times of messages are exponentially distributed. Each node is assumed to have the same message production rate. For simplicity, we assume that the ith message is recognized when the $(i-1)$th message is produced. Note that the mean inter-production time of messages is the reciprocal of the mean message production rate Λ. Message lengths are exponentially distributed with the mean message length *mean.len*. Message laxities are uniformly distributed in the interval $(0, laxvar * mean.len)$ where *laxvar* is a positive integer greater than one.

The system load LD is defined to be:

$$LD = \Lambda * mean.len * N$$

where N is the number of nodes in the system. Given LD, *mean.len* and N, the mean message production rate Λ can be obtained. The simulation time was taken to be 5000 times the mean inter-production time of messages. The fraction of channel time used by successfully transmitted messages is known as the *effective channel utilization*. The maximum value of the effective utilization over all possible loads is known as the *capacity* of the protocol. The capacity is perhaps most greatly affected by the value of the *normalized end-to-end propagation delay* α, defined as $\alpha = \tau / mean.len$ [KSY84]. The performance of our best-effort protocols is measured with $\alpha = 0.1$ and 0.01 (i.e. mean message lengths are 10 and 100) and with *laxvar*= 3 and 9. The retransmission probability P of a collided message is set to be 0.5. To make comparison possible, we use the same period of simulation and the same distributions of message characteristics (production times, laxities and lengths) as those used for simulating the VTL and VTD protocols.

Real-time distributed systems usually include both periodic and aperiodic tasks. Periodic tasks tend to exhibit regular behavior in terms of their activations. However, the times at which they produce messages are not regular because they in general have stochastic execution times. As the execution time variance of each of these tasks increases, more future urgent messages may occur. Moreover, the activations of aperiodic tasks increase the irregularity of message production times. It is this irregularity that real-time protocols attempt to exploit to increase the chance of successful message transmission while decreasing the chance of message collisions.

To investigate the variation in performance as the number of urgent future messages increases we introduce a parameter *SEQ*, called the *sequencer*, into our simulation model. As the sequencer value increases, the number of messages in each *run* tends to increase, where a run is a sequence of consecutive messages produced by the same node. As a consequence, each node tends to have less chance of message collision but to have an increasing chance of having urgent future messages. Each node has an equal chance of producing n consecutive messages, where the number n is a random number drawn from the interval $(0, SEQ)$.

4.2 Performance Measures

As the primary performance measure, we use the message loss ratio *ML* defined as:

$$ML = \frac{TNDM}{TNSM + TNDM}$$

where *TNDM* is the total number of discarded messages and *TNSM* is the total number of successfully transmitted messages.

It can be useful to know how much performance improvement is obtained by one protocol over another in terms of message loss. We use the *performance improvement PI* for such purposes. It is defined as:

$$PI = TNSM_x - TNSM_y$$

where $TNSM_x$ and $TNSM_y$ respectively represent the total number of successfully transmitted messages by the protocols x and y.

The effective channel utilization *ECU* shows how effectively the communication channel is used. The *ECU* is defined as:

$$ECU = \frac{TSM}{TSIM}$$

where *TSM* is the total time units used for the transmission of successful messages and *TSIM* is the total time units simulated.

4.3 Comparative Analysis

We use three different classes of baseline protocols to analyze and show performance of our best-effort protocols. They are as follows:

- The *VTL* and *VTD* protocols.
- The *centralized* LLF (*CLLF*) and EDF (*CEDF*) protocols: These protocols have perfect knowledge about the nodes and the channel, and can transmit messages with no collision. The *CLLF* protocol shows behavior identical to that of the non-preemptive LLF task scheduler, and the *CEDF* protocol shows behavior identical to that of the non-preemptive EDF task scheduler.
- The *clairvoyant* CLLF (*CCLLF*) and CEDF (*CCEDF*) protocols: These two protocols not only have perfect knowledge about the nodes and the channel but also have clairvoyancy about future messages.

While each protocol in the second class shows the best performance achievable with its own transmission policy but with no clairvoyancy, each protocol in the third class does so with clairvoyancy incorporated. Of course, the protocols in both of these classes are not realizable in practice.

The performance of the virtual-time CDMA/CD protocols is shown to be not sensitive to the number of nodes but sensitive to the virtual clock rate η [ZR87]. First, we have tested the *VTL* and *VTD* protocols with an increasing number of nodes; their performance is shown to be quite insensitive to the number of nodes. Then, to find the η value that gives the best *ML* and *ECU*, we tested the *VTL* and *VTD* protocols under various loads from 0.1 to 1.2 and when α=0.1 and 0.01. The best η values for both protocols are in the range of (22,26) when α=0.1 and of (33,36) when α=0.01. Based on these results, we analyze and compare the best-effort protocols with these two protocols in a system of 10 nodes. We use η=25 when α=0.1 and $\eta = 35$ when α=0.01.

Each graph in Figure 4 shows the message loss ratio with increasing system load when α=0.1. The value of the sequencer is set to be 10. The top two graphs (a) and (b) show the message loss ratios of the LLF-based protocols with *laxvar* 3 and 9 respectively, the bottom two graphs (b) and (d) show those of the EDF-based protocols. Corresponding to each of these graphs, the graphs in Figure 5 show the performance improvement of the best-effort protocols and the centralized protocols over the *VTL* and *VTD* protocols. The curves of the CCLLF and CCEDF protocols show the maximum (ideal) performance improvement achievable with clairvoyancy at each given load.

According to the results shown in these graphs, the *BVTL* and *BVTD* protocols always yield message loss ratios lower than those of the *VTL* and *VTD* protocols respectively. Furthermore, when the load is low or medium (up to 0.4 with $laxvar =$ 3 and up to 0.5 with $laxvar = 9$), they even maintain message loss ratios that are lower than those of the *CLLF* and *CEDF* protocols. Only when the system has a very high load (1.0 or more), the *BVTL* and *BVTD* protocols show similar or slightly worse performance over the *VTL* and *VTD* protocols. (see *BVTL* when $LD = 1.0$ and $laxvar = 9$.) The overhead introduced by local and global waiting is the main cause of this phenomenon.

Similarly, each graph in Figure 6 and Figure 7 shows the message loss and the corresponding performance improvement in the case of $\alpha = 0.01$. With small laxities ($laxvar = 3$), the *BVTL* and *BVTD* protocols always show low message loss ratios not only over the *VTL* and *VTD* protocols but also over the *CLLF* and *CEDF* protocols. Even with large laxities ($laxvar = 9$), the *BVTL* and *BVTD* protocols show lower message loss ratios until the system is highly loaded (0.9 or more). Slight performance degradation of *BVTL* over *VTL* is observed when $LD = 1.0$ and $laxvar = 9$.

As mentioned earlier, as the sequencer value is increased, each node tends to have fewer message collisions but to have more urgent future messages. Figure 8 shows the changes in the message loss as the sequencer value is increased. While little performance change is observed in the *VTL* and *VTD* protocols, continuous performance improvements are observed in the *BVTL* and *BVTD* protocols. Note that the performance of each *BVTL* and *BVTD* protocol approaches that of the *CCLLF* and *CCEDF* protocols.

Although not shown in this paper, we have also compared the effective channel utilizations of the best-effort protocols with those of the VTL and VTD protocols. When α is very small (0.01), the $BVTD$ protocol (over VTD) shows better utilization up to $LD = 0.6$. When the load is in the range of (0.7,1.0), depending on the message laxity and the sequencer value, utilization is sometimes better and sometimes poorer. On the other hand, the $BVTL$ protocol (over VTL) always shows better utilization up to LD=1.0, and sometimes better when LD= 1.1 and 1.2. With large α (0.1), the effective utilization of the best-effort protocols is better up to $LD = 0.5$ and sometimes better within $LD = (0.6, 0.9)$.

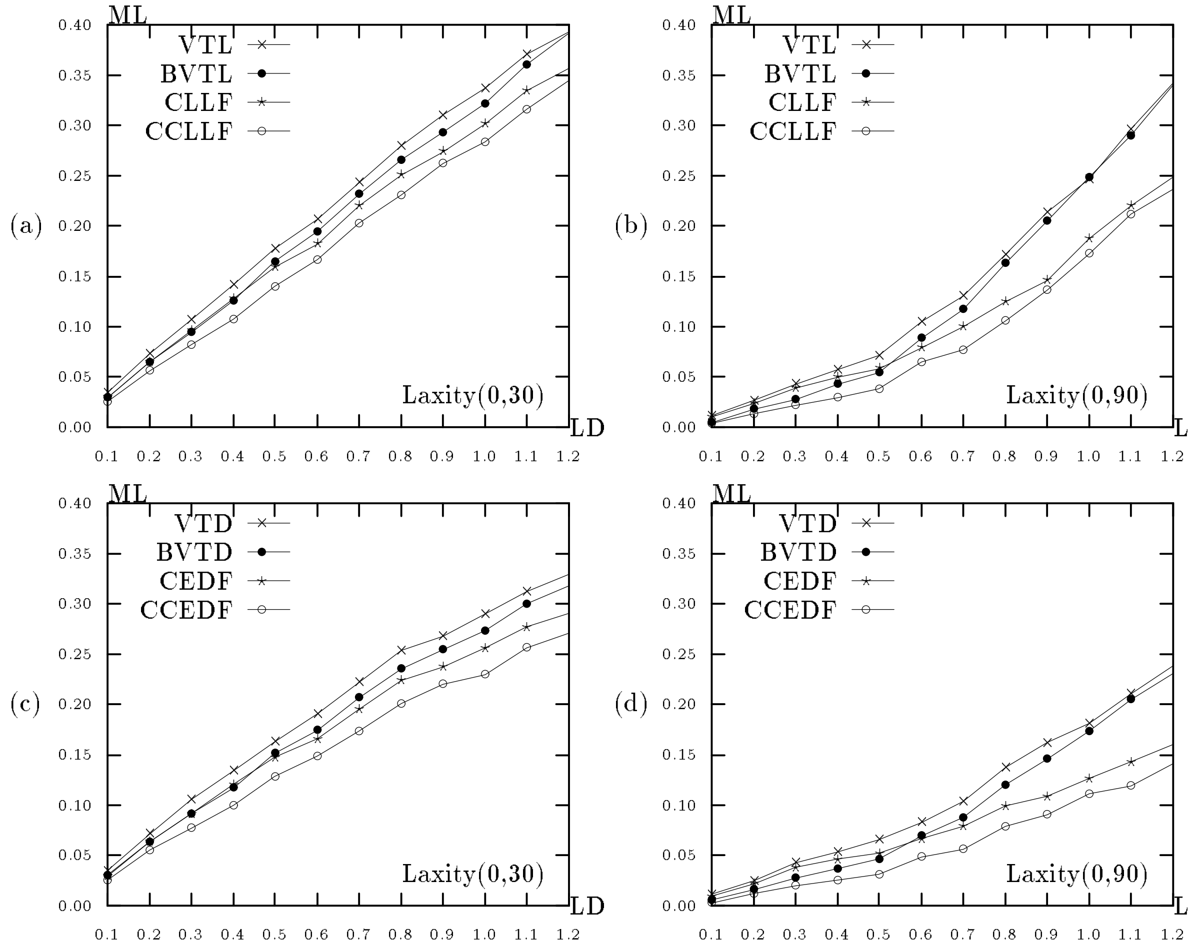

Figure 4: Message Loss ($\alpha = 0.1$)

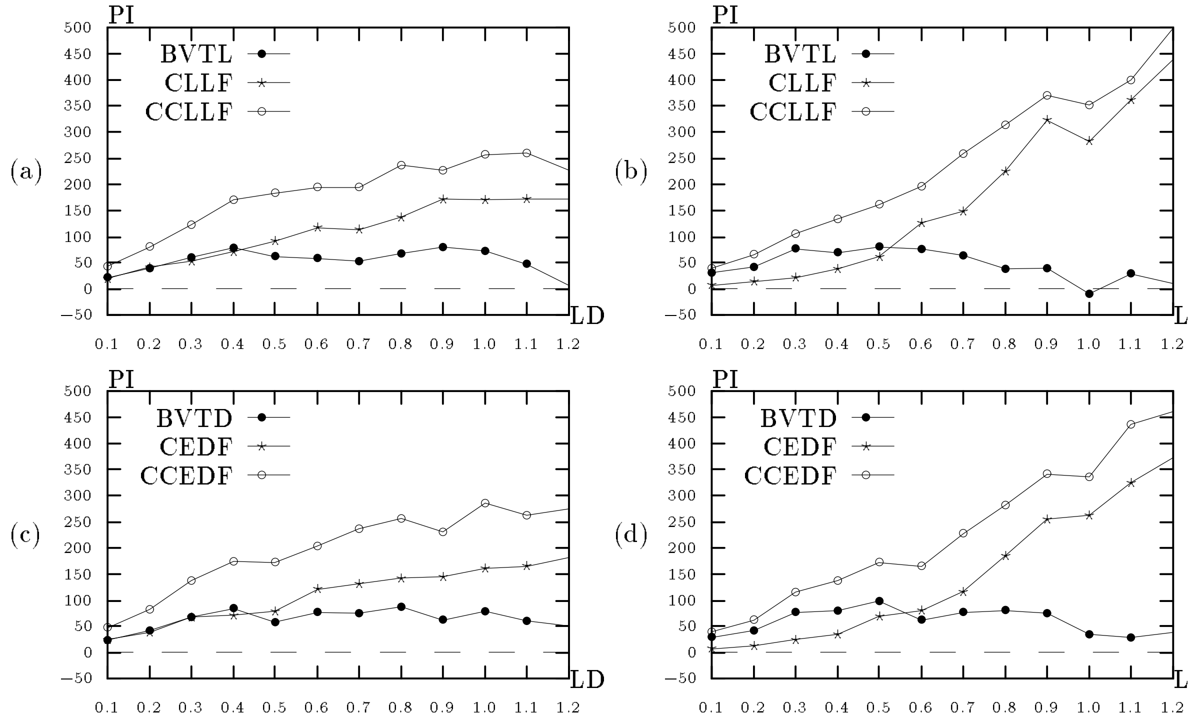

Figure 5: Performance Improvement ($\alpha = 0.1$)

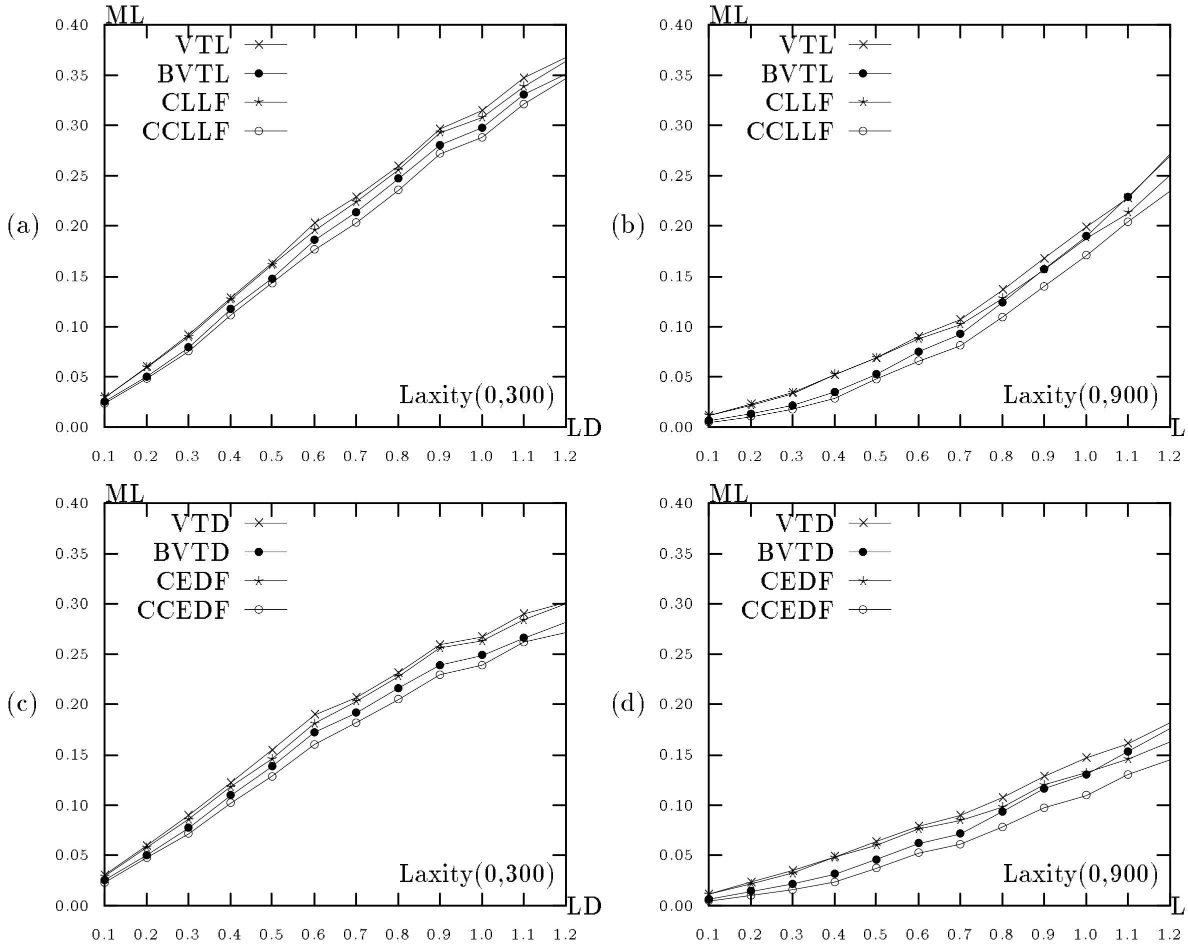

Figure 6: Message Loss ($\alpha = 0.01$)

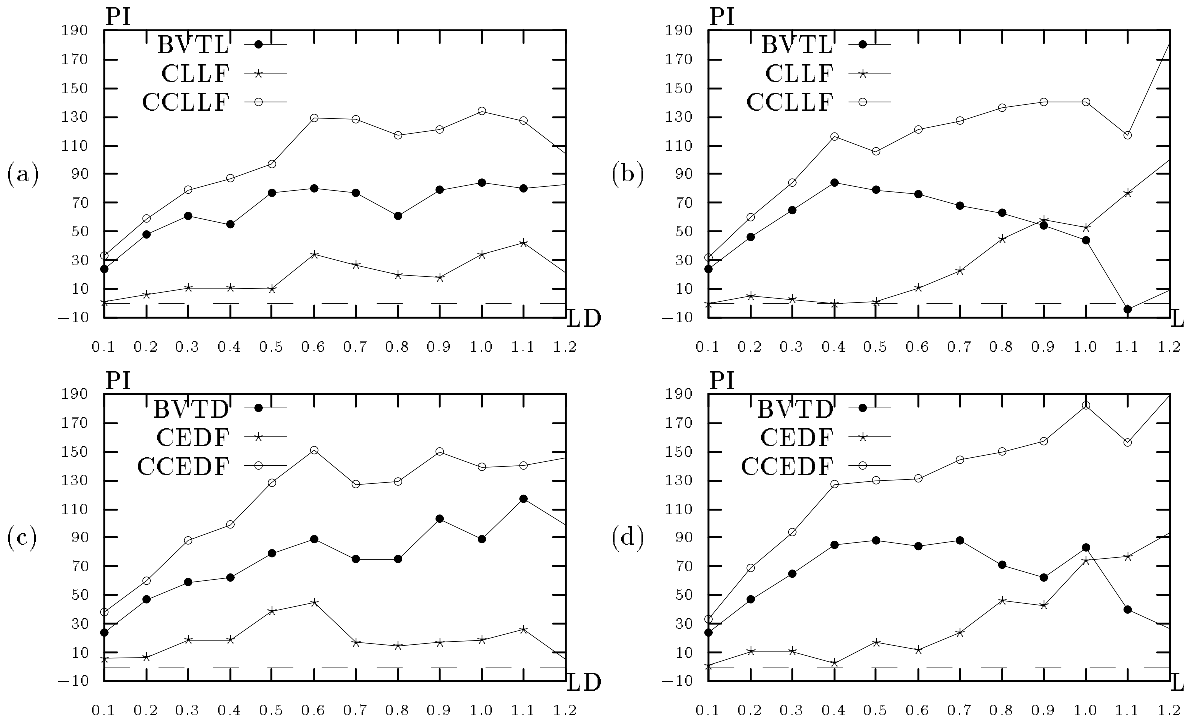

Figure 7: Performance Improvement ($\alpha = 0.01$)

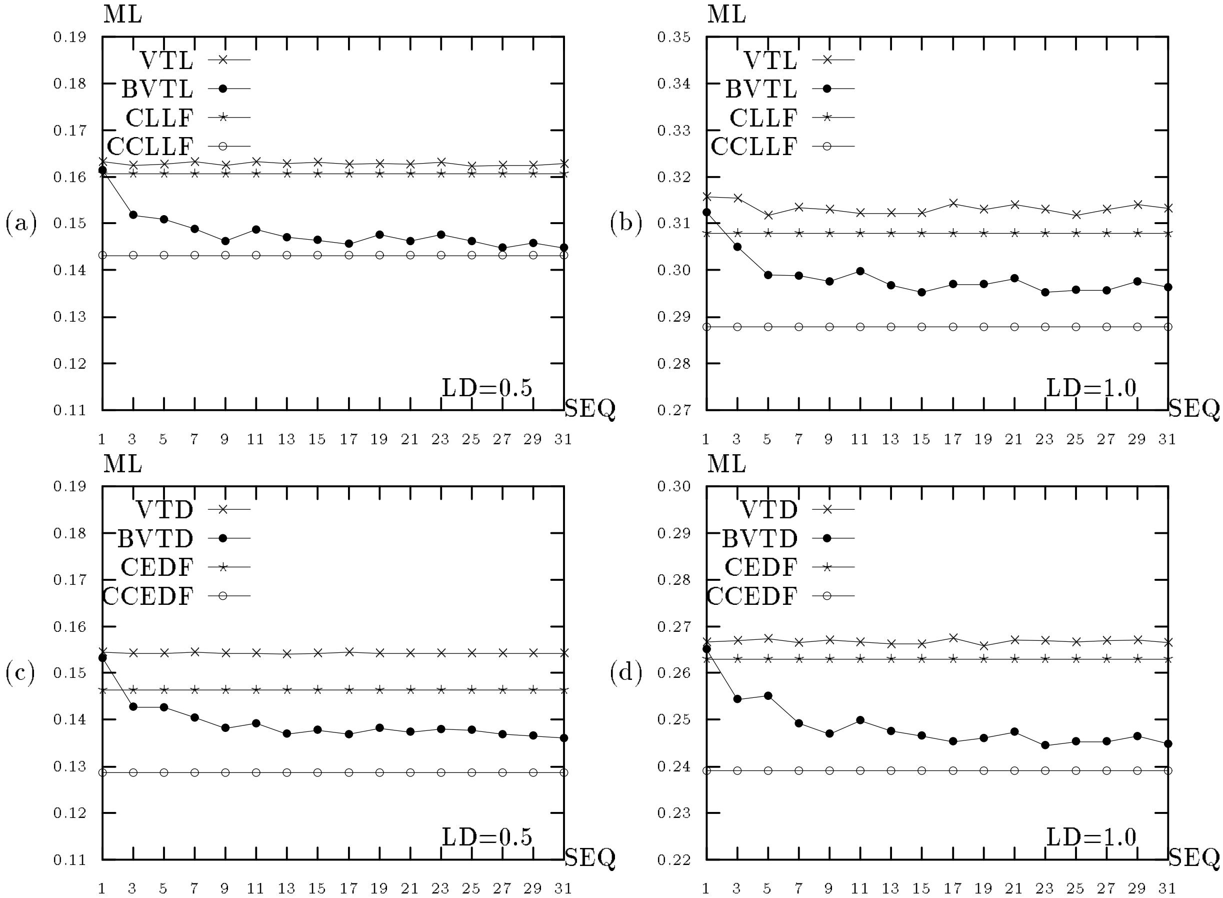

Figure 8: Message Loss with Varying Sequencer ($\alpha = 0.01$, $Laxity = (0, 300)$)

5 Conclusion

Two clairvoyant virtual-time protocols, called *best-effort* protocols, are introduced in this paper. By making use of design-time specification information about the messages and corresponding tasks, these protocols attempt to wait for and deliver urgent future messages that might otherwise be discarded. According to our simulations, the best-effort protocols always have better performance over the VTL and VTD protocols under reasonable system loads (0.1 to 0.9). However, due to the time and channel overheads caused by global and local waiting, they show similar or slightly worse performance when the system load is quite high (1.0 or more). Finally, the best-effort protocols show improving performance as the number of urgent future messages increases.

References

[DM89] M.L. Dertouzos and A.K. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, December 1989. Also shown in Proceedings of the 7th Texas Conference on Computer Systems, November 1978, pages from 5-1 to 5-12.

[JLT85] E.D. Jensen, C.D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. *Proceedings of the Real-Time Systems Symposium*, pages 112–122, December 1985.

[KSY83] J.F. Kurose, M. Schwartz, and Y. Yemini. Controlling window protocols for time-constrained communication in a multiple access environment. *Proceedings of the 8th Data Communications Symposium*, 13(4):75–84, October 1983.

[KSY84] J.F. Kurose, M. Schwartz, and Y. Yemini. Multiple-access protocols and time-constrained communication. *ACM Computing Surveys*, 16(1):43–70, March 1984.

[MK85] M.L. Molle and L. Kleinrock. Virtual Time CSMA: Why two clocks are better than one. *IEEE Transactions on Communications*, 33(9):919–933, September 1985.

[PTW88] S.S. Panwar, D. Towley, and J.K. Wolf. Optimal scheduling policies for a class of queues with customer deadlines to the beginning of service. *Journal of the ACM*, 35(4):832–844, October 1988.

[PZ78] L. Pouzin and H. Zimmermann. A tutorial on protocols. *Proceedings of the IEEE*, 66(11):1346–1370, November 1978.

[ZR87] W. Zhao and K. Ramamritham. Virtual time CSMA protocols for hard real-time communication. *IEEE Transactions on Software Engineering*, 13(8):938–952, 1987.

[ZSR90] W. Zhao, J.A. Stankovic, and K. Ramamritham. A window protocol for transmission of time-constrained messages. *IEEE Transactions on Computers*, 39(9):1186–1203, September 1990.