Design and Simulation of a Reliable Token-Ring Protocol for Realtime Communications

J. Cao
Dept of Computer Science
City Univ. of Hong Kong
Kowloon, Hong Kong

R. Steele
Dept of Computer Science
Flinders University
Adelaide, S.A. Australia

L.J. Yao
Network Analysis Section
Telecom Research Lab
Clayton, Vic. Australia

W. Jia
Dept of Computer Science
City Univ. of Hong Kong
Kowloon, Hong Kong

Abstract

The key to the successful use of computer networks to support realtime communications is an adequate set of network communication protocols that ensure not only timely but also reliable message transmission. Traditional fault tolerance techniques may become invalid in a realtime environment as the latency introduced by those error recovery mechanisms may become unacceptably high. This paper presents the design and simulation of a reliable, realtime token ring protocol. The data link layer design of the protocol provides efficient error detection and recovery, reducing the total message loss rate. The performance of the protocol is evaluated through analysis and simulation.

1 Introduction

Realtime communications are required to support distributed realtime applications such as air traffic control, defense systems, electric power distribution and monitoring, and factory automation. They differ from traditional data communications in two distinct ways. First, they impose an explicit timing constraint like *deadline* on each message transmission such that the message must be received by the destination before its deadline expires. Second, they require high reliability, because message timing requirements may not be met if the system has no provision of error handling and is suspended or delayed due to unanticipated component failures or transmission errors. Therefore, the key to the successful use of computer networks for realtime processing is to provide timely and reliable message delivery.

Token ring is a well-established protocol for regulating the access to a common transmission medium among a large number of stations [3]. In recent years, token ring networks have gained increasing popularity in realtime communications, mainly due to their performance characteristics, such as the bounded time delay, ability to provide guaranteed bandwidth, inherent fairness, and high utilization under overload. Much of the research on real-time token ring protocols has been focused on designing medium access control (MAC) protocols for transmitting time-constrained messages [11, 6, 1, 9]. These real-time MAC protocols implement policies which explicitly incorporate time constraints to control and coordinate access to the shared medium. However, a few of them addressed the issue of fault tolerance. Most of these protocols have assumed a perfect network and protocol operating environment and thus can not tolerate faults occurred during the protocol operation.

This paper presents an integrated design of a fault tolerant protocol for realtime communication on token ring networks. The data-link layer protocol, based on a priority based realtime MAC protocol [11, 5], supports realtime message transmission and provides efficient error recovery. Fault tolerance features are incorporated in two ways:

1. *Efficient handling of token and message loss due to temporary processing and/or communication errors.* Given that any real-time protocol will be overhead monotonic (with increasing overhead more messages will miss their deadlines), any decrease in protocol overhead will improve the protocol's performance. This forms the motivation for an attempt to make the detection of message and token loss more time efficient. The method proposed in this paper is based on the coordination of the sender station and the monitor, rather than solely relying on the monitor, to detect token and message loss. Comparing with the technique used in the IEEE 802.5 Standard, the time complexity of the token/message loss handling is greatly reduced. The proposed method also complements the work on designing robust token ring protocols that can recover from failures such as

deadlock and livelock due to corruption of the token field [7, 2].

2. *Minimizing message loss in the event of a station crash.*

   Since the storage at the Data-Link Layer is volatile, after a station has crashed, all the messages that had been received by the station but have not been delivered to the upper-level protocols for processing will be lost. Traditional fault tolerance techniques used at high-level protocols (eg. the transport layer) may become invalid in a realtime environment as the latency introduced by those error recovery mechanisms may become unacceptably high. Our solution to tolerating station crashes is a data link layer mechanism for message logging and retransmission. To recover messages lost due to a station crash, in our protocol, messages that have been sent to a station are logged by another station. Upon recovery, all these messages, if possible, will be retransmitted automatically by the Data Link Layer protocol. Retransmission at Data Link layer gives a more time efficient recovery method and, consequently, there is more chance for a message to still meet its deadline after retransmission. This data-link layer design of message recovery is justified by the fact that, unlike in a general-purpose communication system where not all applications want to sacrifice efficiency for high quality of services at the low levels, in real-time systems, all higher layer protocols and applications will need this time efficient error recovery.

Simulation of the proposed protocol has been performed. The purpose of the simulation is to compare the performance of the new protocol with the performance of a priority-based realtime protocol without provision for fault tolerance. The results of the simulation show that our protocol can be quite effective in reducing the total message loss rate.

The rest of the paper is organized as follows. Section 2 describe the system model and assumptions necessary for our discussion. Section 3 presents the design of our protocol. First, we briefly describe the underlying priority-based realtime message transmission protocol. Next, we describe how the protocol handles message and token loss caused by temporary processing and transmission errors. Then we describe the logging method used in the protocol for message recovery in the case of a station crash. Section 4 presents a simulation study of the proposed protocol and discusses the simulation results Section 5 concludes the paper with the discussion of our future work.

2 System model

The physical layer of the ring consists of a collection of ring interfaces connected by point-to-point links that form a circle (see Figure 1). Each station is connected to the ring via a ring interface. To enable the ring to continue operation when a link is broken or a station is down, a star-shape ring topology can be used [10]. The star-shape topology involves the use of a ring concentrator. Logically, the network is a ring, but physically each station is connected to the wire concentrator with a cable containing two twisted pairs, one for data coming to the station and one for data going from the station. The wire concentrator contains relays which can be triggered to release either by the lack of current going through a station or by software. The release of the relay will cause the corresponding station to be bypassed.

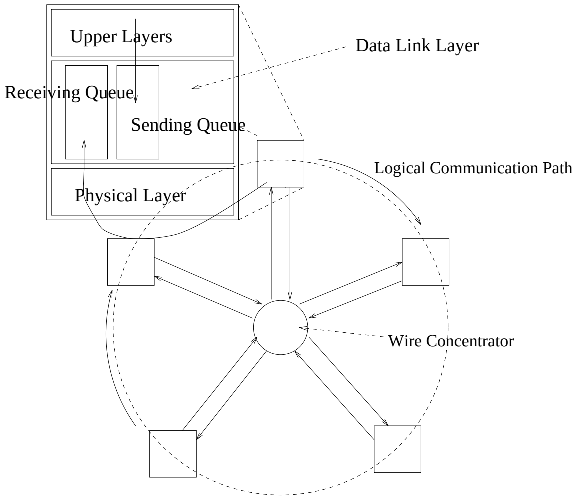

Figure 1: Ring model

The ring interface has two operating modes: listen and transmit. In the *listen* mode, the ring interface repeats each input bit to the output. In the *transmit* mode, the interface breaks the connection between the input and the output and transmit its own data onto the ring. When a bit arrives at an interface it is copied into a 1-bit buffer and then copied out onto the ring again. During the time the bit is in the buffer, it can be inspected and copied into a local message buffer, or modified before being sent out onto the ring again. The copying step introduces a 1-bit delay and, this is the same at a host copying a message into storage or one just passing it on [10]. Bits are sent and received in fixed-size blocks, called frames. A *frame* is a unit of transmission, which is passed from station to station around the ring.

The Data Link Layer (DLL) carries out the function of co-ordinating access to the shared medium and providing error control. The DLL is further divided into two sub-layers: a *medium access control* (MAC) sublayer and a *logic link control* (LLC) sublayer. The primary function of the MAC protocol is to schedule message transmission over the ring. It arbitrates access to the ring and determines which message is to be transmitted at any given time. In a realtime system, the MAC protocol is responsible for implementing suitable policy for transmitting time-constrained messages. The LLC protocol aims to achieve reliable and efficient communication between stations at the data link layer, by detecting errors and retransmitting messages.

Messages are entities introduced to the Data-link layer protocol from upper layers. A message arrived at the DLL is segmented into packets for transmission. Each packet is fitted into a frame. The MAC protocol is responsible for assembling and disassembling frames, and provide CRC validation. A message that has been carried all the way from the source node to the destination node is said to be *received*. At a station, a message received and delivered to the upper-level is said to be *processed*. In our model, each DLL entity defines an interface between the DLL protocol and the higher layer protocols. It includes storage for the messages to be sent to the ring or to be logged by the local station. It also maintains a buffer for the messages to be delivered to the higher layer protocols. The higher layer protocols are assumed to have the simplest functionality of generating messages and processing received messages.

The life of a message starts with its generation by a higher layer entity at the sender station, followed by being passed to the the Data Link Layer for transmission. Upon reception at the destination Data Link Layer, the message will be passed up to the upper layers at that station, where the message will be processed. Each message has a number of attributes, which determine the actions the protocol must take with it. Each message is uniquely identified by an identification number. Other basic attributes are *source address, destination address, length, arrival-time,* and *deadline*. The arrival time of a message is the time when the message arrives at the station's DLL interface. The message deadline denotes the absolute time by which a message must be received at destination station.

Failures in a token ring network can take many forms. In this paper, we are concerned with the following kinds of failures, which can be caused by either transmission errors or station crashes:

- *loss of messages*: messages are lost during transmission before being received by the destination.
- *loss of token:* the token is considered lost after a pre-defined period of time during which no token/messages have passed.

  The difference between a message being lost and one just corrupted is whether or not it can be recognized by the destination station. Messages can fail to arrive at their destinations if the header information is damaged or discarded. In a ring of typical parameters, rarely will a whole message be lost as only a small part of a message is ever on the ring at any time. However if a message cannot be recognized by its destination station, it would seem to be lost, even though only part of it has been altered.
- *loss of received messages*: messages are lost from the state of the destination DLL entity. This can arise through the station crashing. It is assumed that station crashes may destroy messages that have been received but not yet processed. Higher layers will pass to the Data Link layer as a parameter the last message number that has been processed.

We assume that the network can eventually recovery from hardware failures such as communication link breakage and station crashes. That is, no hardware fault is permanent. The task of designing a fault tolerant token ring protocol is, once a working ring is reconstructed, to "resynchronize" the stations to determine a correct ring configuration and to recover lost and missing messages.

3 Design of the reliable relatime token-ring protocol

Figure 2 shows the structure of the DLL entity used in our protocol. The DLL entity includes storage for the information about messages that are maintained by the local station. Each DLL entity has a *Sending Queue* (SQ) which contains those messages that are currently waiting to be sent on the ring. Messages in the SQ are ranked according to their deadlines with the most urgent message at the head. Messages whose deadlines expired will be dropped from the SQ. Messages received from the ring will be stored in the *Receiving Queue* in their arriving order before it is passed to the upper layers for processing. The *Logging Storage* (LS) contains the messages that will be logged by the local station. As we will see in Section 3.3, message logging is used in the new protocol to recovery messages lost due to station crashes.

3.1 Priority-based Realtime Message Transmission

In our protocol, realtime message transmission is based on a priority-based token-ring protocol [11, 5], which will be referred to as the PBRT protocol.

The PBRT protocol is an extension of the fault-free operation of the IEEE 802.5 Standard [3]. In the Standard, the basic operation of the token ring is based on the use of a special packet called the *token* to select the next message to be transmitted. The token format is shown in Figure 3. The Start delimiter and End delimiter fields of the token mark the beginning and ending of the frame. A station wishing to send data to another station waits for a free token, changes it to busy (by setting the **T** bit), and appends its own data packet behind the claimed token. When the transmitted packet returns, the source station will remove it from the ring and then releases the token to the station downstream.

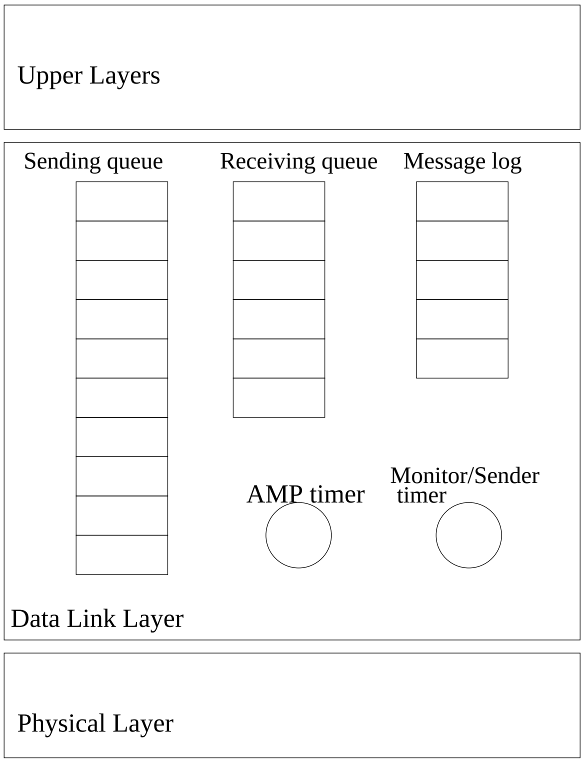

Figure 2: Structure of a node

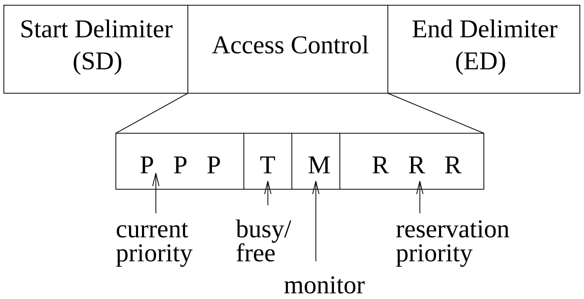

Figure 3: IEEE 802.5 token format

Messages arrive at a station with priorities. The ring supports prioritized operation by using the priority (PPP) and reservation (RRR) fields. When a busy token goes by, each station examines the reservation (RRR) field. If the reservation field is of lower priority than the priority of the station's pending packet, the contents of the reservation field are updated with the pending packet's priority. However, if a higher priority has already been claimed in the reservation field, the station is not allowed to modify it. The source station wait until the transmission is finished and then generates a free token. The priority of the newly generated free token is the returned RRR field, that is, it is the priority that has already been reserved, if any; otherwise, it is at the present priority level.

Although the IEEE 802.5 MAC protocol supports prioritized message transmission, it does not directly deal with the explicit time constraint of individual messages. The PBRT protocol extends the IEEE 802.5 protocol to support real-time message transmission by mapping the deadlines of messages into priorities, using an appropriate priority assignment function [11]. Messages on each station are queued in increasing order of their deadlines. When a free token or a message frame goes by, the priority of the first message in the queue is calculated. This priority is then compared with the priority in the token to determine whether this station should send the message at this time, as is done in the original IEEE 802.5 protocol.

The priority of a message is calculated using a priority assignment function, which is a function of the deadlines of messages. The priority assignment function used in the PBRT protocol assigns a lower numerical priority value to an earlier deadline. However, due to the physical length of the token on one hand and the potentially infinite number of message deadlines on the other hand, the mapping is many-to-one. This implies that two or more messages with different deadlines may be assigned the same priority and hence the one with the later deadline may be sent first. From the results of extensive simulation studies, it has been found that the average performance of the PBRT protocol is close to that of the optimal EDF policy if a proper priority assignment function and a sufficient number of priorities are used [11, 5].

The PBRT protocol does not have any provision for fault tolerance. It relies on the original IEEE 802.5 protocol for error handling. On the other hand, although the 802.5 Standard incorporates an array of fault tolerance features, the normal actions of the 802.5 protocol and its fault-handling actions do not have any provision for real-time processing. In the remainder of this section, we describe a new protocol which extends the PBRT protocol's capability to support for detection and and recovery of certain types of failures. The protocol also improves the fault-handling techniques used in the IEEE 802.5 Standard, by introducing an efficient method for detecting message and token loss and a real-time method for reducing message loss in the event of a station crash.

3.2 Handling of token and frame loss

Both the token and a message can be lost due to several reasons. The corruption or loss of the header or ED leads to a frame being lost as opposed to corrupted in the normal sense: if the SD is lost or the source address is lost the frame would not be received again at the sender station; If the ED is lost the sender would not know whether transmission of the frame has finished.

In the IEEE 802.5 Standard, each token ring has one node designated as the *monitor* node which ensures the correct operation of the ring. The monitor node periodically sends out a message called "I am alive" informing other nodes of its existence. Other nodes enter the process of bidding for the monitor node after a pre-defined period during which no such message has passed. The first node that sends out a bidding message will eventually become the new monitor node. The monitor is responsible for detecting the token loss. It does so by observing if a free token or any frame passes on the ring. Message loss is detected indirectly via the detection of token loss. The basic technique is to set a timer, taking the advantage of the ring's property that a message will be either on time or will not arrive at all. The monitor runs a timer that will expire if the token takes longer to return than possible. Under normal conditions with no fault occurring, the maximum time before the timer expires is equal to $N \times THT + T$, where N is the number of stations in the ring, THT is the Token Hold Time, and T is the roundtrip propagation delay. Given that a usual THT value is 10 milliseconds, this delay can be up to several seconds.

It seems that the most reasonable way to detect message loss is to let the source station perform the task, as it has the privileged position of always being able to observe when a message was set and when exactly to expect its return. However, as the detection of message loss often leads to token generation, the actions taken by the sender must be coordinated with the actions of the monitor. In the new protocol, the monitor has the sole responsibility of detecting a freely circulating token. However, when the token is being held and so some messages are being transmitted on the ring, the monitor and the sender station share the responsibility of detecting message/token loss.

As described above, the IEEE 802.5 Standard uses two timers at each station. One is run to detect the absence of Active Monitor Present (AMP) frame, and the other is run by the monitor to detect the loss of the token. In our protocol, each DLL entity also maintains two timers (see Figure 2). One is the AMP timer. Another timer can be used in two different ways: (a) for detection of token loss if the local station is the monitor, (b) for detection of message loss if the local station is the current frame transmitter.

First, whenever an End Delimiter (ED) goes by, the monitor starts a timer with timeout T, which is the roundtrip propagation delay. This timeout value is determined based on the following timing analysis:

At time t: ED leaves the sender;

At time $t + fT$: ED passes the monitor, where f is fraction of the ring to the monitor;

At time $t + T$: ED returns to the sender;

At time $t+T+fT$: SD of new frame or a free token arrives at the monitor.

The analysis indicates that if an ED passes the monitor, any loss of the token in the next revolution would be detected by the monitor within time T.

However, a SD can be lost after past the monitor without being detected by the monitor. To solve this problem, when a SD comes past, the monitor will reinitialize the timer. The timer will be stopped if an ED arrives before the timer expires. In the case where the monitor detects a token or a message loss, it simply regenerates the token with the minimum reservation field. If the new token reaches to the sender, the sender will return to the token-capturing state. The sender will try to retransmit the lost frame the next time it captures the token. The timeout value can be determined as follows.

- If the SD is that of a free token, the ED will immediately follow and the timeout value should be set to the transmission time of one byte.
- If the SD is of a message frame, however, a timeout value of $THT - T$ is required, because the SD could be that of a data frame which could have a transmission time of almost the Token Holding Time. The following timing analysis indicates this fact:

  At time t: sender transmits SD;

  At time $t + fT$: monitor observes the SD;

  At time $t + THT - T$: the latest time when sender must transmit ED;

  At time $t + THT - T + fT$: monitor observes ED.

Therefore, when a frame is currently being transmitted on the ring, if a SD was lost just after passing the monitor or it was lost before reaching the monitor and the newly generated token was lost again before reaching the sender, then the token would disappear from the system for almost the period of THT. This is based on the assumption that only the monitor can regenerate the token. One way to improve the performance involves the cooperation with the sender. The sender stores the Access Control (AC) bits in the previous frame or token and regenerates the token upon detection of the loss of the SD. In this way, the token could disappear for a period of T, only one roundtrip propagation delay. This incurs no storage overhead, at least not in a protocol that uses the 802.5 token reservation mechanism, which requires that the latest version of the AC field be stored anyway. The actions of the sender and the monitor can be described as follows.

- The sender makes the standard calculation that it would after it regenerates the token. It will go ahead and transmit the most urgent frame of which the deadline has not passed, if the Token Holding Time allows it. This transmission will usually be the resending of the lost frame, but not always, as in the cases where the deadline has expired or a more urgent message has newly been generated at the station.
- The monitor is awaiting either an ED, if the previous token was lost after passing the monitor, or

a SD, if the previous token was lost before reaching the monitor and the new token generated by the monitor was lost again before reaching the sender. In both cases, when the monitor sees the SD of a new frame or of the token regenerated by the sender, it simply resets its timer to wait for an ED again. However, the values of the timeout would be different for a free token and a retransmitted frame.

In summary, in the new protocol, detection of the loss of a free token takes a maximum time of T, one roundtrip propagation delay. Loss of a message frame is detected also within T. Comparing with the time of $N \times THT + T$ required by the IEEE 802.5 protocol, our protocol is much more efficient. Also, in our scheme, the stations need not know the maximum number N of stations in the ring.

3.3 Realtime messages recovery

Messages can be lost due to transient transmission failures as described in the previous subsection. Messages can also be lost due to station crashes. In this section, we describe the technique used in the new protocol for recoverying messages lost due to station crash. Complementary to the PBRT protocol, the goal of the new protocol is to minimize the number of messages lost during abnormal operation of the token ring.

To achieve messages recovery upon station crash at the data link layer, two things must be done: (1) keep a log of the messages destinated at the failed station somewhere in the system; (2) resend the messages to the failed station when it recovers. The copying of a message at an extra station will not further delay the propagating message compared with just repeating the message on the ring. Thus no extra overhead will be resulted due to logging. However, there will be extra cost in retransmitting the logged messages upon recovery. Two design decisions must be made: first, one muse decide which stations log the messages; second, one must decide how to retransmit the logged message upon recovery.

One simple solution is to let each DLL entity log all the messages that have been transmitted by the local station and, upon recovery of a station from its crash, retransmit the messages to the recoveryed station. Retransmitted messages will be treated as normal messages and retransmission is taken place in the same way as in normal token ring operation. This method, however, is rejected mainly because it takes too long to retransmit the messages and during retransmission deadlines of most of these messages would be expired. A more efficient recovery method is needed.

For simplicity of description, the message recovery mechanism described below is based on the assumption that at any time only a single station can crash and the MTTF (mean time to fail) of the ring is sufficiently large such that no other station can fail before the currently failed station recovers and rebuilds its message log (e.g. MTTF is greater than most of the message deadlines). That is, the logger station is always alive at the recovery time. To relax the failure assumption, the proposed solution can be extended, e.g. by having more than one logger for each destination station. The reader is referred to [8] for more details.

The first issue to be considered is how to choose the logger. In the new protocol, the set of messages that have been successfully delivered to their destinations are partitioned into subsets according to their destinations. Each subset is logged by a separate station; in our scheme, it is the upstream neighbor of the destination station (see Figure 4). This choice has the following advantages. First, the messages to be retransmitted upon recovery from a station crash can be found at a single station. Second, any message that arrives at the destination must have been successfully copied by the logger station. Third, upon retransmission, the logger is the closest to the recovered station and therefore a retransmitted message will only traverse one link, minimizing the likehood of losing or corrupting the message. Note that, in terms of speedup of retransmission, it can be shown that choosing the upstream neighbor as the logger for a station does not provide an advantage over retransmission from other stations, given that retransmission is always triggered by a request from the recovered station [8].

The second problem involved is how to choose a retransmission policy. Upon recovery of the crashed station, messages that have been delivered to the station but have not been processed by the upper layer protocols at that station need be retransmitted, if their deadlines have not yet expired. The recovered station obtains necessary recovery information from the upper layer protocols and sends a control frame, called the *Unprocesses Message Retransmit* (UMR) frame. Among others, the UMR frame contains the recovered station's address and the id number of the message last processes by the upper layer before the crash.

The logger could retransmit the messages in several ways: it could either (1) place them on its Sending Queue (SQ), mixed with normal messages, or (2) send them immediately before the normal messages, or (3) send them altogether as a conglomerate frame. Different policies have their own advantages and disadvantages. In our protocol, we choose to use the third one because of the following reasons: (a) Messages that have been delivered should have been delivered, so recovered messages should be retransmitted before normal messages that have not been sent yet; (b) By sending the recovered messages as a single conglomerate frame, the equivalent of sending one frame after another without waiting for the previous to return is achieved. When the End Delimiter of the UMR frame returns to the recovered station it will have received all the frames available for retransmission. This will achieve the fastest possible retransmission.

Discussion about alternative approaches to achieving message logging and retransmission can be found in [8].

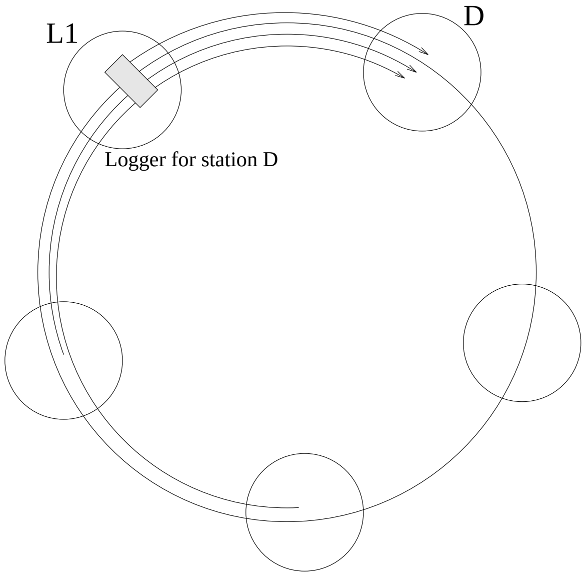

Figure 4: The logging scheme

4 Simulation results

The proposed protocol has been evaluated through simulation. Results of the simulation are used to compare the performance of the new protocol with message logging with the performance of the PBRT protocol which has no provision for recovery of messages upon station crash. This section describes the simulation design and the observations made from the preliminary results.

The simulation were developed using the discrete event simulation language *Simscript II.5* [4]. Simscript provides a rich set of features that support the development of simulations, including statements for time advance, event processing, generation of statistical variates, and accumulation and analysis of simulation-generated data. A simulation written in Simscript consists of a main program which initializes the simulation, a set of time-ordered events and a collection of activity routines that will be invoked upon occurrence of the corresponding events.

The simulation parameters used include the mean deadline of the messages, the load of each station, the number of station crashes, and the mean time to repair a crashed station (MTTR). In our simulation, we assume that the load of all the stations is always the same, the number of crashes is fixed at 5, and the MTTR is given a constant (0.003 seconds). Two groups of simulations have been run for both protocols:

1. Group One: with a constant mean message deadline and varying loads;
2. Group Two: with a constant load and varying mean message deadlines.

The generation of messages at the sender is modeled as a set of exponentially distributed events. Each run of the simulation continues execution until 5000 messages have been generated (but it is not necessarily true that all the messages have been sent). The length of all the messages is set at 150 bits and the Token Holding Time is set to allow for one message transmission. Finally, the number of nodes in the ring is set to 10 and the speed of the ring is set to 1-Mbps.

The following performance metrics are used to evaluate the protocols:

- *Number of messages lost due to deadline expiration during normal operation.* The normal message loss ratio is defined as

$$r_{normal} = \frac{N_{lost}}{(N_{lost} + N_{sent})}$$

  where, N_{lost} is the number of messages lost due to deadline expiration in the normal execution, N_{sent} is the total number of messages that have been transmitted.

- *Number of messages lost due to station crashes.* The failure message loss ratio is defined as

$$r_{crash} = \frac{F_{lost}}{(F_{lost} + N_{sent})}$$

  where, F_{lost} is the number of messages lost due to deadline expiration when failure occurs, N_{sent} is the total number of messages that have been transmitted.

- *Storage overhead of message logging.* The logging overhead is expressed as the maximum amount of storage required for message logging at any time.

Figure 5 and 6 show the normal message loss ratio for both protocols obtained from the results of Group One and Group Two simulations, respectively. It can be seen from Figure 5 that, for both protocols, the normal message loss ratio increases in an approximately linear fashion with increasing load. This could be expected because, as load increases, messages arrive with a smaller mean inter-arrival time but will still only be sent on the ring at the same rate allowed by the ring. The second observation is that the new protocol performs slightly worse then the PBRT protocol, in terms of the normal loss ratio. This is because of the processing overhead incurred in the message retransmission upon recovery of crashed stations. The logger also has to wait for the arrival of the whole UMR frame before retransmission.

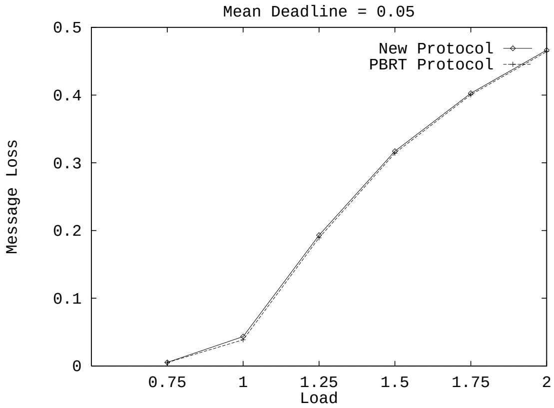

Figure 5: Normal message loss for varying load

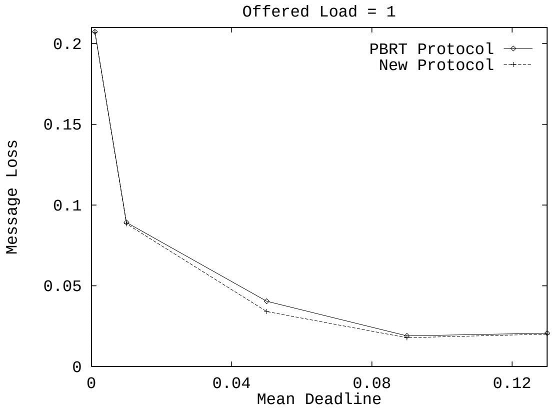

Figure 6: Normal message loss for varying deadline

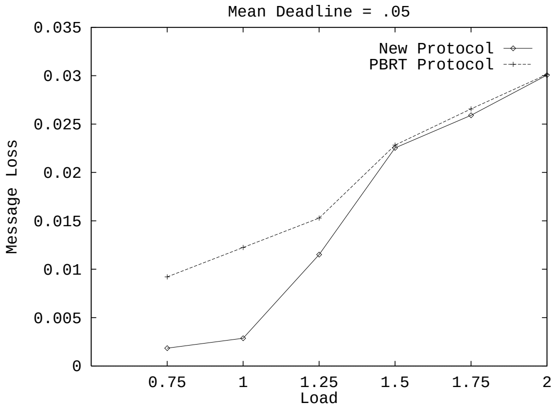

Figure 7: Failure message loss for varying load

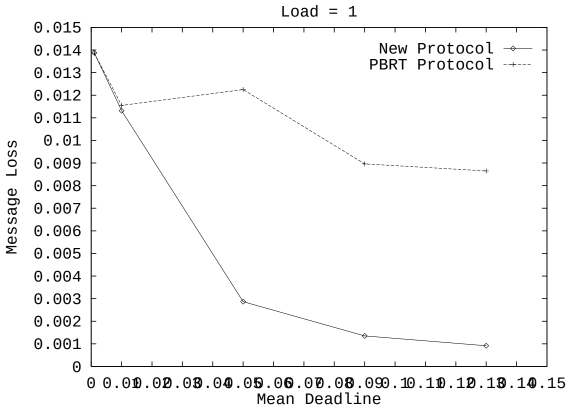

Figure 8: Failure message loss for varying deadline

Figure 6 indicates that, for both protocol, increasing mean deadline leads to significant drop in the normal loss ration. Again the ration for the new protocol is slightly higher than that of the PBRT protocol. The maximum disparity occurs at deadlines of a middle value. At very small mean deadlines, very few retransmission can be done and the new protocol will not incur a high overhead. At very high deadlines, a large amount of retransmission can occur but the mean deadline is sufficiently large so that most of the normal messages being delayed by retransmission will not be out of date.

Figure 7 and 8 show the failure message loss ratio for both protocols obtained from the results of Group One and Group Two simulations, respectively. From Figure 7, we can see that as the load increases the failure loss ratio for both protocols increases. At low loads the new protocol performs significantly better than the PBRT protocol. At high loads, however, their performance becomes quite close. The reason for this is that at high loads messages are delayed longer than at low loads and most messages lost due to crashes will not get retransmitted due to deadline expiry.

The advantage of the new protocol over the PBRT protocol clearly occurs when the mean message deadline is high. This can be observed from Figure 8. For very small deadlines, the performance of the two protocols is almost identical, because with very small deadlines no retransmission can be done upon recovery of station crash. The loss that occurs with the new protocol is determined largely by the number of messages that are waiting in the receiving queue of the crashes station, which is dependent on the rate by which messages are delivered to the station. Since we used a fixed value for the load the result is not likely to exhibit any particular trend.

Figure 9 shows the logging storage overhead for varying loads with three different mean message deadlines. The general observation is that the overhead increases as the load decreases or the mean deadline increases. This could be expected as with a high load and a large mean deadline more messages will be delivered to the destination station and stay in the logging queue for longer.

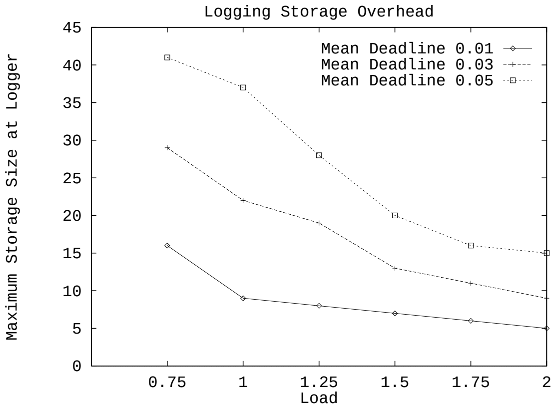

Figure 9: Logging overhead

5 Conclusions

We have proposed a fault tolerant protocol for realtime communication in token ring networks. The protocol introduces (1) techniques for efficient detection of token and message loss caused by transient transmission errors and (2) a scheme based on message logging for recovery of messages upon station crash. Simulation studies have been carried out to evaluate the performance of the proposed protocol and the preliminary simulation results indicate that the date link layer message logging scheme can be quite effective in reducing the total message loss rate. With low loads and large message deadline, the protocol was most effective and performs better than the PBRT protocol which has no fault tolerance.

Currently we are conducting more extensive simulation studies to study the behavior of the new protocol under various conditions and to compare the new protocol's performance with that of a protocol where message recovery is performed at higher levels. Our future work include a formal analysis of some aspects of the performance of the proposed protocol. We will also investigate the effect of extending the protocol to handle more failure modes.

References

[1] A. Bhargava, J.F. Kurose and D. Towsley, "A Hybrid Media Access Protocol for High-Speed Ring Networks", *IEEE Trans. Selected Areas in Communications*, 6(6):924-933, July 1988.

[2] R. Cohen and A. Segall, "An Efficient Reliable Ring Protocol", *Proc. 3rd Int'l Workshop on Distributed Algorithms*, LNCS 392, 1989, pp70-82.

[3] IEEE, "Token Ring Access Method, IEEE 802.5 Local Area Network Standard", *IEEE Computer Society*, Silver Spring, Maryland, 1985.

[4] P.J. Kiviat *et al*, "Simscript II.5 Programming Language", CACI Products Company, 1991.

[5] C.C. Lim, L. Yao, and W. Zhao, "A Comparative Study of Three Token Ring Protocols for Real-Time Communication", *Proc. IEEE Int'l Conf on Distributed Computing Systems*, 1991.

[6] N. Macolm and W. Zhao, "Hard Real-Time Communication in Multiple-Access Networks", *Technical Report*, Dept. of Computer Science, Texas A&M University, June 4, 1993.

[7] J. Pachl and L. Casey, "A Robust Ring Transmission Protocol", *Computer Networks and ISDN Systems*, 13, 1987, pp313-321.

[8] R.J. Steele, "Fault Tolerant Real-time Token Ring Protocols", *Honours Thesis*, Dept of Computer Science, University of Adelaide, Nov. 1994.

[9] J.K. Stronider, T. Marchok and J. Lehoczky, "Advanced Realtime Scheduling Using the IEEE 802.5 Token Ring", *Proc. 9th IEEE Realtime Systems Symposium*, Dec. 1988, pp42-52.

[10] A.S. Tanenbaum, "Computer Networks", 2nd edition, *Prentice-Hall International, Inc*, 1989.

[11] L. Yao and W. Zhao, "Performance of an Extended IEEE 802.5 Protocol in Hard Real-Time Systems", *Proc. IEEE 10th Conf. Computer Communications,, Infocom'91*, 1991, pp.5C.1.1-5C.1.10.

[12] W. Zhao, J.Stankovic and K. Ramamritham, "A Window Protocol for Transmission of Time Constrained Messages", *IEEE Trans. Computers*, C-39(9), Sep. 1990, pp1186-1203.