UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

A livelock-free protocol for slotted rings

Research Report

Jan Pachl

CS-87-20

March, 1987

UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

A livelock-free protocol for slotted rings

Jan Pachl

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

March 17, 1987

Abstract

The purpose of this paper is to show that the commonly used empty-slot protocol for slotted rings is livelock-prone, and to propose a livelock-free modification of the protocol.

Research supported by the Natural Sciences and Engineering Research Council of Canada.

1 The empty slot protocol

A *slotted ring* is a unidirectional ring communication network in which a number of fixed-length slots are passed from station to station around the ring. A station that has data to send to another station waits for an empty slot to arrive and uses the slot to send the data.

A well-known example of a slotted ring is the Cambridge ring [5], in which the slot has the following format:

The one-bit field E ("full-empty") has two values, *empty* and *full*. The one-bit field M ("monitor") has two values, *new* and *old*. The DEST field is the destination station address, the SRCE field is the source station address, and the RESP field ("response bits") transports an acknowledgement from the destination to the source.

There is exactly one monitor station in the ring. Each ordinary station (other than the monitor) executes the following protocol to send data: If there are data to transmit, the station repeats each slot as received until the received value of the E field is *empty*; then the station changes the value of the E field to *full*, and sends the value *new* in the M field, the destination address in the DEST field, its own address in the SRCE field and data in the DATA field; the value *destination absent* in sent in the RESP field.

The destination station recognizes its address in the DEST field, and, if it has a buffer available for reception, makes a copy of the DATA field and sends the value *packet accepted* in the RESP field. However, except for the RESP field, the destination station repeats the slot as received. The sending station waits for the slot to return around the ring, then changes the E field back to *empty*, and repeats the rest of the slot as received. (The number of slots in the ring is fixed, and the station is therefore able to determine when the slot has returned.) The sending station also reads the RESP field, and thus finds out whether the data have been received by the destination.

The monitor station manipulates the contents of the E and M fields, and repeats the rest of the slot as received. (On a lower protocol level, the monitor station is also responsible for maintaining the circulating slots.) When the monitor station receives the values (*full, new*) in the (E,M) pair, it sends (*full, old*) instead; when the monitor station receives (*full, old*), it sends *empty* in the E field.

In [5] it is argued that ring reliability decreases as delays in each node increase. As is pointed out on p. 171 in [3], the Cambridge ring protocol

may be implemented with minimum delay in non-monitor stations. Indeed, if a (non-monitor) station has data to send then the value of the E bit sent by the station in the next slot is *full*, regardless whether the slot arrives empty or full. Thus the outgoing value of the E bit depends on its incoming value only when the station has no data to send, and in that case the E bit value is simply repeated by the station. However, in the monitor station the delay must be longer, because the value of the E bit sent by the monitor station depends on the value of the M bit received in the same slot.

Transmission errors affect the E field in two ways: *empty* value may be changed to *full*, and *full* value may be changed to *empty*. The monitor station protocol ensures that the ring recovers from the empty-to-full errors. However, in the next section it is shown that the protocol described above may fail as a result of a single full-to-empty error.

2 A livelock example

Figure 1 shows one possible execution in a ring with five non-monitor stations. (The monitor station is not shown.) There is a single circulating slot. In the figure, the thick line means that the E field is set to *full*, and the thin line means that the E field is set to *empty*.

Station 1 sends data, and a transmission error marks the slot empty between stations 3 and 4. Station 4 immediately uses the slot to send its own data, and from then on the stations send their data in the order

$$4, 2, 5, 3, 1, 4, 2, 5, 3, 1, \ldots$$

As long as all the nodes always have data to send, the situation persists forever. (This would happen, for example, if a higher-level protocol required that unacknowledged data be retransmitted.) If the data from station 1 are destined for station 4, from station 2 for station 5, from station 3 for station 1, from station 4 for station 2, and from station 5 for station 3, then no data ever reach their destination.

3 The recovery procedure

It is not difficult to devise modifications of the protocol which avoid livelocks like the one in section 2. However, in this section it will be shown that there even exists a modification that is compatible with the original protocol, in the sense that the slot format and the protocol in the monitor station remain the same.

In the protocol described in section 1, the sending station reads the RESP field in the slot returning around the ring. Thus the station can also read the

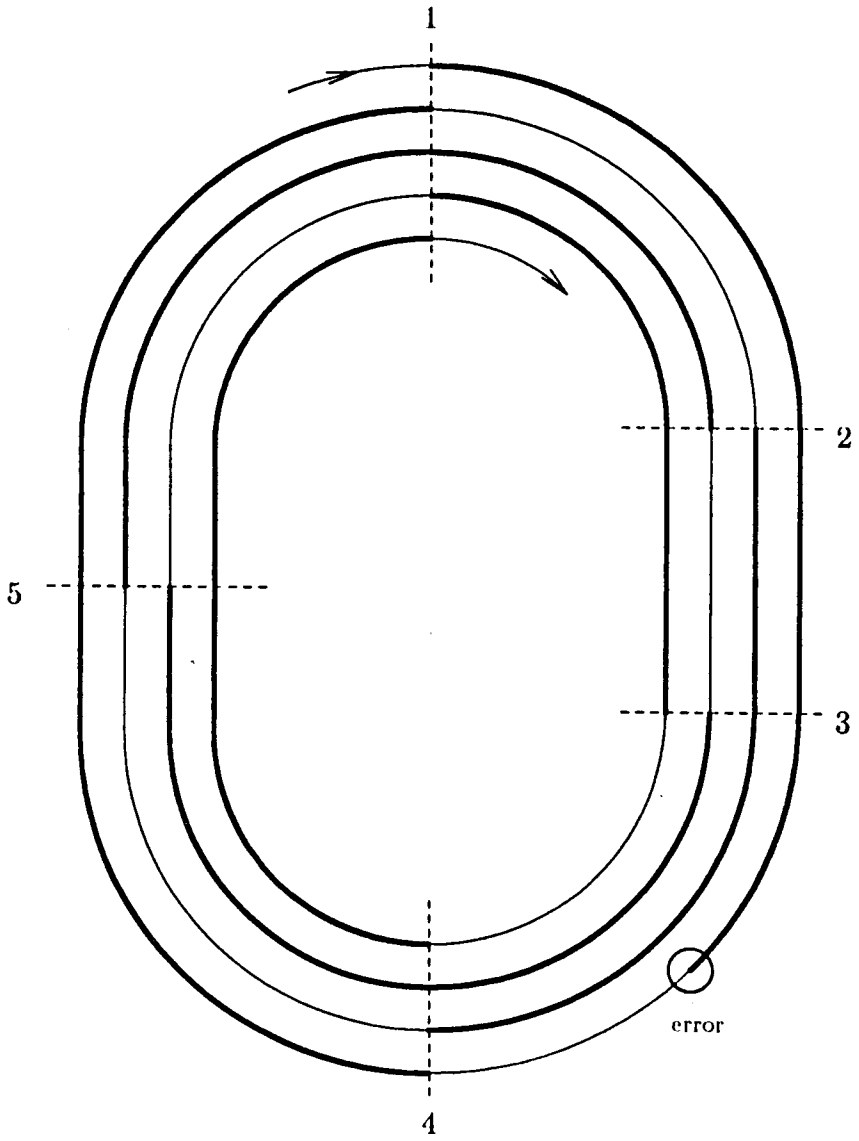

Figure 1. A livelock scenario.

thick line = slot marked full
thin line = slot marked empty

SRCE field and compare it with its own station address. If they differ, then the content of the slot may have been overwritten by another station, and it is possible that a livelock has developed. Therefore the station initiates a *recovery procedure.*

In the proposed recovery procedure, the station waits until the slot in which the problem has occurred is received with the E field set to *empty*, changes it to *full*, sends *new* in the M field, an invalid address in the DEST field, and its own address in the SRCE field. (Thus the station waits until it is allowed to send its own data in the suspect slot, and sends dummy data, not addressed to any valid destination.) However, the station does *not* change the E field back to *empty* when the slot returns.

The appendix contains an algorithmic description of the protocol. One can argue, informally, that the protocol is livelock-free, as follows: Livelocks like the one in Figure 1 arise when there is more than one "permission to transmit" in one slot during one round trip. Every time a station executes the recovery procedure, the number of those permissions is reduced by one. It is possible that all permissions are removed in this way, but then a new permission is created by the monitor station.

Figure 2 shows an example of recovery in a ring with three non-monitor stations. The square box symbols in the figure denote the times when recovering stations send dummy data.

This technique is conceptually similar to that used in the token ring defined in [1], where the situation in which multiple tokens are created through an error is converted to the situation with no token. (Cf. [1], p. 76.)

It should be noted that only the data sender protocol is modified. The other parts of the original protocol remain the same. Moreover, the proposed protocol is compatible with the original one in a very strong sense: One can mix stations executing the original protocol with those executing the modified protocol; as long as the stations executing the modified protocol send their data often enough, the system recovers from any number of full-to-empty errors.

4 Other recovery procedures

Another approach to recovery is taken by the designers of the MAN protocol [4]. Although the protocol has a different frame format, the following rule, adapted from [4], may be added to the protocol in section 1 to prevent livelocks: The sending station changes the E field value in the returning slot back to *empty* only if the M value in the returning slot is *old.* The disadvantage of this scheme is that, since the outgoing value of the E bit depends on the incoming value of the M bit, the delay in every station has to be at least one bit longer than the minimum delay necessary to retransmit

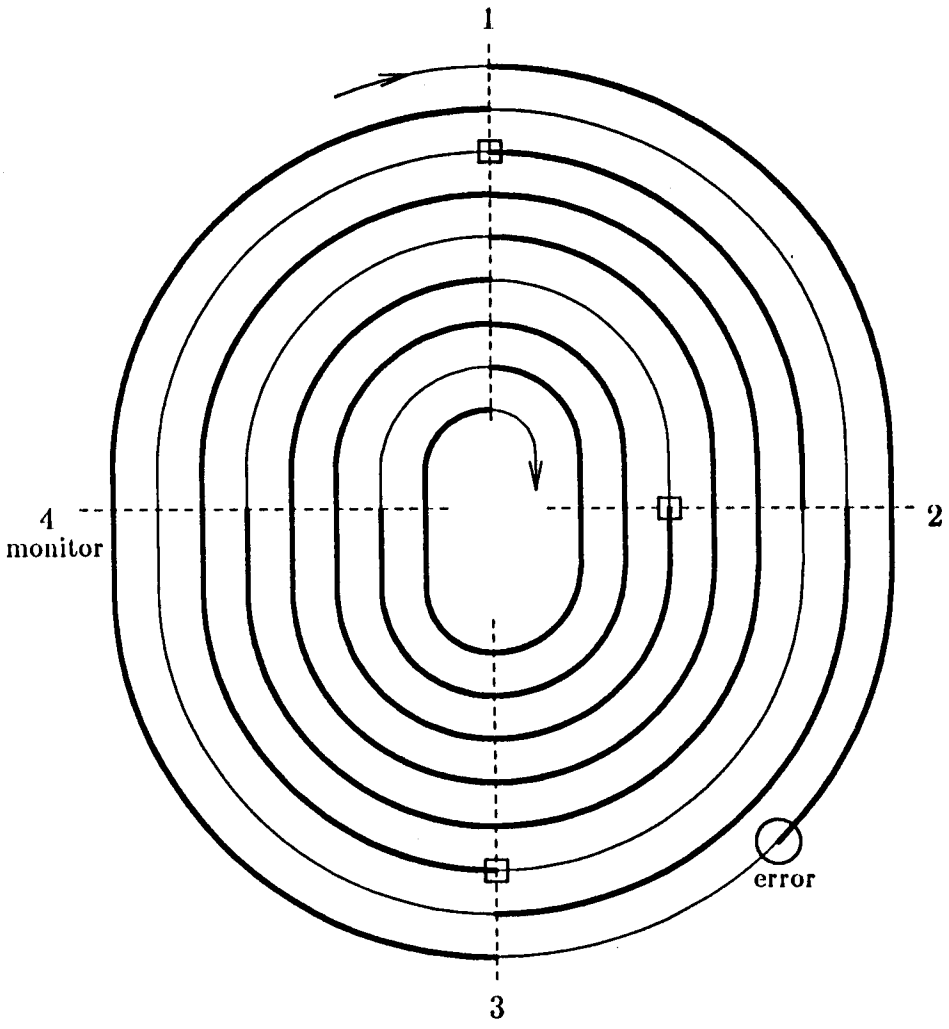

Figure 2. The recovery procedure.

thick line = slot marked full
thin line = slot marked empty

the received bit values. However, the scheme is very simple, and is to be recommended when longer than minimum delays are acceptable.

All the protocols in this paper are *centralized*, because they rely on one station (the monitor station) that is different from other stations in the ring. Zafiropulo and Rothauser [6] describe a *decentralized* protocol for slotted rings; several livelock-free modifications of the protocol are proposed in [2].

References

[1] W. Bux, F. Closs, P. A. Janson, K. Kummerle, H. R. Muller, and E. H. Rothauser. "A local-area communication network based on a reliable token-ring system", *Local Computer Networks* (Edited by P. C. Ravasio, G. Hopkins and N. Naffah), *Proc. IFIP TC 6 International In-Depth Symp. on Local Computer Networks*, Florence, Italy, April 19-21, 1982; pp. 69-82.

[2] J. K. Pachl and L. M. Casey. "A robust ring transmission protocol", *UW/ICR 86-11* (July 1986), Institute for Computer Research, University of Waterloo.

[3] B. K. Penney and A. A. Baghdadi. "Survey of computer communication loop networks: Part 1", *Computer Communications 2* (1979), 165-180.

[4] D. T. W. Sze. "A metropolitan area network", *IEEE J. Selected Areas in Communications* SAC-3 (1985), 815-824.

[5] M. V. Wilkes and D. J. Wheeler. "The Cambridge digital communication ring", *Proc. Local Area Networks Symp.* (Edited by N.B. Meisner and R. Rosenthal), Boston, May 1979.

[6] P. Zafiropulo and E. H. Rothauser. "Signaling and frame structures in highly decentralized loop systems", *Proc. 1st Internat. Conf. Computer Communications* , Washington, D. C., October 1972.

Appendix

This appendix contains a description of the modified sender protocol in an algorithmic notation. For the sake of simplicity it is assumed that there is only one slot in the ring (so that the sender need not skip any passing slots before receiving the transmitted slot after its trip around the ring).

The protocol is described by means of three low-level primitives:

```
bit_in  =  Repeat_bit();
bit_in  =  Send_bit( bit_out );
           Await_next_slot();
```

The primitive `Repeat_bit` copies the next bit from the input channel to the output channel, and returns its value. The primitive `Send_bit` receives the next bit from the input channel (`bit_in`) and sends the bit `bit_out` to the output channel. The primitive `Await_next_slot` copies the bits from the input channel to the output channel until a new slot begins.

The procedures

```
                   Send_address( address_out );
address_in  =      Receive_address();
```

which are easily written in terms `Send_bit` and `Repeat_bit` , are used to send and received address fields.

The modified sender protocol follows on the next page.

```
constants
    OWN_ADDRESS       : address
    INVALID_ADDRESS   : address
    FULL, EMPTY, NEW  : bit

variable
    recovering : boolean


recovering := FALSE;
repeat
  begin
    Await_next_slot();
    if  "ready to send"  and  not  recovering  then
      begin
        if  Send_bit( FULL ) = EMPTY  then     {* E field *}
          begin
            Send_bit( NEW );                   {* M field *}
            Send_address( "destination address" );
                                               {* DEST field *}
            Send_address( OWN_ADDRESS );       {* SRCE field *}
            "send data and response field"
            Await_next_slot();
            Send_bit( EMPTY );                 {* E field *}
            Send_bit( NEW );                   {* M field *}
            Receive_address();                 {* DEST field *}
            if  Receive_address() !=  OWN_ADDRESS  then
                                               {* SRCE field *}
                recovering := TRUE;
      end end
    else if  recovering  then
      begin
        if  Send_bit( FULL ) = EMPTY  then     {* E field *}
          begin
            Send_bit( NEW );                   {* M field *}
            Send_address( INVALID_ADDRESS );   {* DEST field *}
            Send_address( OWN_ADDRESS );
            Await_next_slot();
            recovering  := FALSE;
  end end end
```