M.I.T. Laboratory for Computer Science

Request for Comments No. 247
April 25, 1983

Possibilities for Improved Ring Control Protocols

by David C. Feldmeier

1. Introduction

Although the version 2 ring appears to work reliably in its current configuration, there are some possible improvements that could be made on the next version. In this document, several ideas for improvement are proposed. All of these proposals are for decentralized ring control and could be used on any token ring.

2. Multiple Control Character

2.1. Twin Tokens

The version 2 ring uses a variety of special characters to control traffic on the network. The two most important of these are tokens (free to transmit) and connectors (beginning of message). During normal ring operation, a station that wishes to place a message on the ring will wait for a token, change the token character into a connector, append the message and finish the transmission with an End of Message flag then a token. The station then waits for its connector to come back around the ring and then removes its connector and message, leaving the token.[1]

The conversion of a token into a connector is done simply by changing a bit in the token. It seems that occasionally a station would fail to append a token and to remove its connector or a burst of noise would cause this bit to flip, leaving a connector circulating and no token. Normally, this would be corrected as soon as a station wanted to send and it noticed that a token had not been seen in 700 milliseconds. The station would then force a token onto the ring and append its message in the normal way and as a side-effect reinitialize the ring. The problem with the circulating connector is that in some stations a large number of interrupts occur, causing the associated host computer to crash or to become inoperative due to successive interrupts by the network card for long periods of time. This is obviously undesirable. It would also be nice if the ring would recover in less than 700 milliseconds. With the current rate of traffic on the ring averaged over a day, over a 700 millisecond period three packets would normally be sent. This means that three stations could try to initialize the ring at approximately the same time since loss of token is noticed almost simultaneously at all stations. A collision would occur and this would necessitate the host software begin a random back-off algorithm.

A possible solution to this problem of a circulating connector is to use two different types of connectors (and therefore two different types of tokens) that must occur alternately. This could be achieved by extending the length of the connector/token by one bit. This would cause some new rules on the ring:

1. All tokens mean free to transmit.
2. Connector types must occur alternately.
3. As long as just tokens exist on the ring, each token must be of the same type.
4. The originating station must replace the token that it just removed from the ring with one of the *opposite* type.
5. Stations will inspect the circulating token or connector in order to determine which type of connector/token to expect next.

For example, let the two types of tokens(connectors) be *green* tokens and *red* tokens. The ring is up and a *green* token is circulating so everyone expects to see a *green* token (or connector) next. Finally, a packet is sent. The *green* token is replaced with a *green* connector, a packet is sent and it is ended by a *red* token. If chains of packets go by, the connectors will alternate in type. A station just joining the ring can synchronize correctly by examining the first token or connector to go by.

What happens if somehow a connector circulates on the ring? Say a *green* token becomes a *green* connector and never gets removed from the ring. After the *green* connector, everybody is looking for a *red* token or connector. When neither comes by in a certain amount of time (less than 4 milliseconds in the current ring), the station will determine that the ring is down and be in a position to force a message out when it has one to send.

2.2. Twin Flags

This section is slightly more complicated. All valid control characters on the ring are comprised of the very basic control character the *flag*. The flag is followed by two bits to determine a token, a connector, or an End of Message control character. The rules in this system are:

1. Flags must alternate.

2. A token is counted as a flag of the opposite type of that flag which comprises the token.

3. If the correct type of flag does not occur in the round-trip time of the ring, then the ring is down.

As a token circulates, a station will first detect a flag (let's say *green*). Then as the control character is recognized as a token, the station treats this just as if a flag of the opposite type has passed (pseudo-*red*). In this way, a circulating token appears to be alternating flags. This also ensures that when a token changed into a connector, the connector will be of the correct type. A problem with this occurs when two valid flags of opposite type (say a connector and an End of Message) are left circulating on the ring. In this case, trouble will only be noticed because no station has seen a token in 700 milliseconds. Needless to say, two circulating flags of opposite types would be much less probable than a single circulating flag.

2.3. Numbered Control Characters

This is basically the same as the two ideas above, except that instead of two control characters there are 2^n types of characters. The control characters are sent and should be received with the number incrementing by one modulo 2^n.

3. Station Counting

Since only 255 stations can be accommodated on the current ring, a simple thing to do is to add a new eight bit *count* field to the hardware header. This field would be initialized to zero and incremented by one at each station that repeats it. If the *count* field ever overflows, it must do so at the 256th station on the ring. Since only 255 stations can exist on the ring, the current packet must have gone around more than once. When the count field overflows, the station that notes the overflow should generate a token and drain the ring until the token returns.

The *count* field should be arranged such that the least significant bit comes first. In this way, the *count* field can be incremented on the fly a bit at a time. This counting scheme has the added advantage that a station may determine the number of nodes in the network by sending a packet to itself and examining the *count* field upon reception.

A system with more overhead than the one outlined above would have a *count* field on all control characters, not just connectors. Therefore, any control character that lives too long would be eliminated from the ring. Since the token is a control character, it also would be incremented and removed every 256 stations. This is OK though because the token is simply removed and replaced with another token, although it would probably be simpler to make tokens an exception and not replace them.

These systems seem fairly foolproof because even if the *count* field is corrupted, at worst a message would be removed prematurely from the ring and a new token forced.

4. Guaranteed Bandwidth

With certain types of data transfer, it might be desirable to have a guaranteed minimum bandwidth to insure that a packet is transferred within a certain time limit. It could be useful to have two priority levels, normal and high. A normal packet has guaranteed access to the ring at worst when every station before it transmits a full length packet (approximately 700 milliseconds on a maximum size ring). Ideally, high priority traffic could have access to the ring in less than one maximum size packet time (approximately 4 milliseconds in a large ring).

The idea here is to introduce a new type of control character on the ring that has a very special property. This special property is that it must immediately pass through every station on the ring and that it must be able to be transmitted without waiting for a token. Modifications to present control characters (such as having a priority bit in the connector) would not work because the connector would only travel between the station requesting priority and the station that transmitted the connector. A station that wishes to request priority would insert this special character anywhere in the bitstream (after a '0' so as not to interfere with other control characters on the ring). Since each node in the ring must have ten bits of delay in order to maintain a token in loopback mode, the *priority flag* could be inserted and removed by register insertion. When the priority flag returns, it is removed from the bitstream.

All other nodes on the ring must repeat the priority flag immediately and in the process of doing so, inhibit transmission of normal messages. This repeat requirement is automatically satisfied by all nodes except for the one (if any) that is currently transmitting a packet. A node that is transmitting must keep the last ten bits or so in its ten bit register so that when it determines that the last ten bits have been a priority flag, it can suspend its own transmission (making sure to end on a '0') and immediately clock out the priority flag (note that it need not store the flag - if a priority flag has just come in, the transmitter can simply create a new one in its place). This disabling of normal transmission allows the token to be used only by a priority station. When the token comes by, the station changes the token into a priority connector and sends the message as usual. The priority message is ended with a normal token and normal ring operation is resumed. Other stations may reenable normal transmission upon reception of two tokens without an intervening priority connector. This last rule assures that all priority messages are sent before any normal messages.

This system would seem to work for one priority message at a time, but what happens if two or more stations decide to send priority messages simultaneously? Say two stations place priority flags on the ring at the same time. Thus, instead of removing its own flag, each station removes the others' priority flag. This is not a problem because every node on the ring has seen one flag or the other and knows that someone wishes to send a priority message. Now both stations simply wait for the token to come by. Whichever receives the token first is the first to transmit. Since normal stations must wait for two tokens in a row in order to reenable normal transmission, the only stations that can send on the first pass of the token are priority stations. Each time a priority station sends, a priority connector must be sent. This resets the normal stations to wait for two tokens once again. In this way, all priority messages are sent, the token makes a total ring circulation untouched (to ensure that there are no more priority messages left) and then normal ring operation is resumed.

A problem with this scheme is what to do if there are no tokens presently circulating on the ring. There are two different types of problems here. One is no control characters on the ring, and the other is circulating control characters but no token. Circulating control characters are easy. If two normal priority connectors with no intervening priority connectors pass by after an assertion of a priority flag, then the problem must be a circulating connector and priority transmission may begin as soon as two normal connectors are seen in a row. The problem of no control characters at all means that the ring must be reinitialized in the usual way. Things should be simpler since only the high priority stations are involved (less contention).

Other considerations include loss of bandwidth and unfairness. The loss of bandwidth arises due to the fact that there exist special characters (priority flags) that use net resources but are not directly involved with the higher level transfer of information. Since there is only one priority flag per priority message, the loss of bandwidth probably would be very minor. Another loss of bandwidth is due to the fact that the token must circle the ring once after all priority messages and hence cannot be utilized for a ring circulation time. The only alternative to this (that I can think of) is to use the approach that is used on the IBM experimental token ring, which is to delay token release until the connector returns to the originating station.[2] If a priority station has marked the connector with a priority request, a priority token is released. Otherwise, a normal token is released. But this also has a loss of bandwidth due to the fact that a station must hold the token until its connector returns.

There is certain unfairness due to the existence of priority stations. The first free token for normal net transmission after a priority transmission is generated at the priority station. Assuming that the priority station is not fast enough to utilize its own token for normal transmission, the station directly after the priority station has first access to the free token. With a number of priority stations distributed on the net, this unfairness will most likely be very minor and in fact it could be that the stations that have the least allocation for normal transmission are the priority stations.

A special control character like the one above could be useful for many other types of ring control. Since it travels around the ring in the minimum possible time, it is the fastest means of low level communication between nodes on the ring.

5. Conclusion

All of the ideas above are possible ways to improve the low level control of a token ring network. For a good description of how the version 2 ring works, see *Operation and Maintenance Manual for the Pronet Local Area Data Communications Network.* Any suggestions or comments on these ideas or this paper would be greatly appreciated.

References

[1]David D. Clark, Kenneth T.Pogran and David Reed,
An Introduction to Local Area Networks
Proceedings of the IEEE, vol. 66, no. 11, November 1978.

[2]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,
Research Report RZ 1117, IBM Zurich Research Laboratory (December 22, 1981).