

APPENDIX B. ASYNCHRONOUS TOKEN RING COMMUNICATIONS

B.1 Design Criteria for the Local Area Network

A fully distributed expert system implies some means for the individual experts to communicate - in brief, a local area network. For this research, the network must meet the following goals:

- a) *Inexpensive*. It defeats the purpose of using inexpensive microcontrollers, if the network interface is expensive or requires custom logic. Ideally, the network interface should require only the "standard" asynchronous serial port, and no external hardware.
- b) *Fiber optic capable*. A particle accelerator is an "electrically hostile" environment. Also, it is desirable to place control microprocessors inside various high-voltage cages. A network which can be carried on fiber optic cable is needed.
- c) *Open-ended*. The network should support a variable, and possibly large, number of stations.
- d) *Peer-to-peer*. The reasoning model is a "society of experts," each of whom is free to consult with, or advise, any other. Thus peer-to-peer communication is essential.
- e) *Broadcast*. Frequently, one station must advise many, or all, stations of a change in some fact. The network must allow broadcast messages from any station.
- f) *Deterministic*. Deterministic protocols are preferred for real-time control, guaranteeing an upper bound on time for one station to access the net, and for a message to be sent. Maximizing the utilization of this low cost (and presumably low speed) network is also desirable.

B.2 Physical Topology

Three physical topologies were considered for the local area network: bus, star, and ring.

The physical bus is widely used. It is open-ended and can be implemented on many embedded microcontrollers (Butler, 1991, 1992). Deterministic protocols exist which support peer-to-peer and broadcast messages. Its chief disadvantage is the difficulty of using fiber optic links. While fiber optics

have been implemented (Tanenbaum 1989; Valenzano, Demartini, and Ciminiera 1992), for bussed local area networks, these installations require specialized bus transceivers which increase the cost.

A star topology, on the other hand, uses exclusively point-to-point links and is therefore readily adapted to fiber optic cable. The remote stations can use common (and inexpensive) serial ports. The "hub" of the network is the bottleneck: it requires many serial ports, and this ultimately places a limit on expansion. The hub must also actively relay peer-to-peer and broadcast messages, and, without special hardware, this load increases with the number of serial ports.

Many advantages of physical rings have been stated by (Tanenbaum 1989):

Among their many attractive features is the fact that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. Point-to-point links involve a well-understood and field-proven technology, and can run on twisted pair, coaxial cable, or fiber optics. Ring engineering is almost entirely digital, whereas [IEEE Standard] 802.3, for example, has a substantial analog component for collision detection. A ring is also fair and has a known upper bound on channel access.

A disadvantage frequently cited for physical rings is that, if one station on the ring fails, the entire ring fails. This, however, can be an advantage in a control system: for the proper operation of the accelerator, all stations must be functioning. If any station fails, safety requires that all stations enter their "safe" or shutdown mode.

The principal disadvantage of most ring networks is the requirement for specialized hardware. Inexpensive interfaces have long been available for rings such as the IBM Synchronous Data-Link Control (Intel, 1981; Motorola, 1983; Zilog, 1981). Unfortunately, SDLC, a "logical star," does not support peer-to-peer or broadcast messages.

The IBM Token Ring, standardized as IEEE 802.5 (IEEE, 1985; Strole, 1987), is more promising. This is a true peer-to-peer network in which ownership of the ring rotates among all of the stations on the ring. Both group and broadcast addresses are supported. The 802.5 protocol satisfies all of the functional requirements; unfortunately, it is a synchronous protocol requiring specialized interface hardware. The ideal would be a protocol like 802.5, using standard asynchronous serial ports.

B.3 The Token Ring Using Asynchronous Communications

Each computer in the Asynchronous Token Ring receives serial data from the "previous" station in the ring, and transmits to the "next" station in the ring. Thus each station requires only one serial port. At any time, one station is the "owner" (master) of the ring, entitled to put data on the ring, and responsible for removing this data from the ring. The other "listening" (slave) stations simply retransmit any data they receive. Serial data bytes are originated by the owner, handed around the ring by the slaves, and consumed when they return to the master. Each station in the ring introduces a delay of one byte period (rather than one bit period as in IEEE 802.5).

The ring owner transmits data in frames, using the format shown in Figure B-1. The beginning of each frame is uniquely identified by a START code (0FF hex). All listening stations interpret a START code, regardless of when received, to mean the beginning of a new frame. A frame can be aborted by simply sending a new frame.

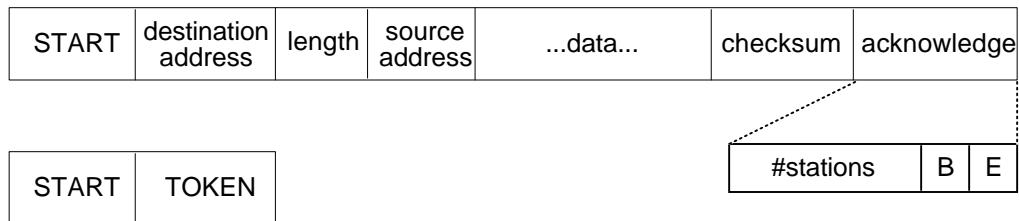


Figure B-1. Token Ring Frame Formats

The destination address byte may designate a specific recipient, or "broadcast" (00 hex). The station(s) designated must validate the frame, and if possible, receive it. Stations which do not recognize this address may return to passively "echoing" received bytes, while awaiting another START code. Nonaddressed stations need not buffer any of the frame: they can reject the frame immediately.

To simplify the logic of the software state machine, a message length byte is sent next. This is

the length of the *data* field, from 0 to 63 bytes. This is followed by the source address byte and the data field. During the transmission of the data field, reserved codes (such as START) are sent using "character stuffing" (Tanenbaum, 1989). The special code STUFF (0FD hex) is sent, followed by the data byte XOR 80h. This is transparent to the user and does not affect the transmitted length byte.

Following the data field, an FCS byte (Frame Check Sequence) is sent. This is the two's complement of the binary checksum over the destination, length, source, and data fields.

The acknowledge byte is sent after the checksum so that it may be modified by listening stations as they pass the frame around the ring. Each station which recognizes the destination address, whether or not it copies the message, increments the 6-bit "#stations" field. When this byte completes its trip around the ring, the sender can discover missing stations, duplicate station addresses, or (in the case of broadcast messages) the total number of stations on the ring. The "B" bit is set by any recipient which was unable to copy the message due to lack of buffer space. The "E" bit is set by any recipient which detects a checksum error in the frame. Thus the sender can determine if the frame needs to be resent. In the case of broadcast messages, these bits indicate that at least one recipient failed to receive the frame.

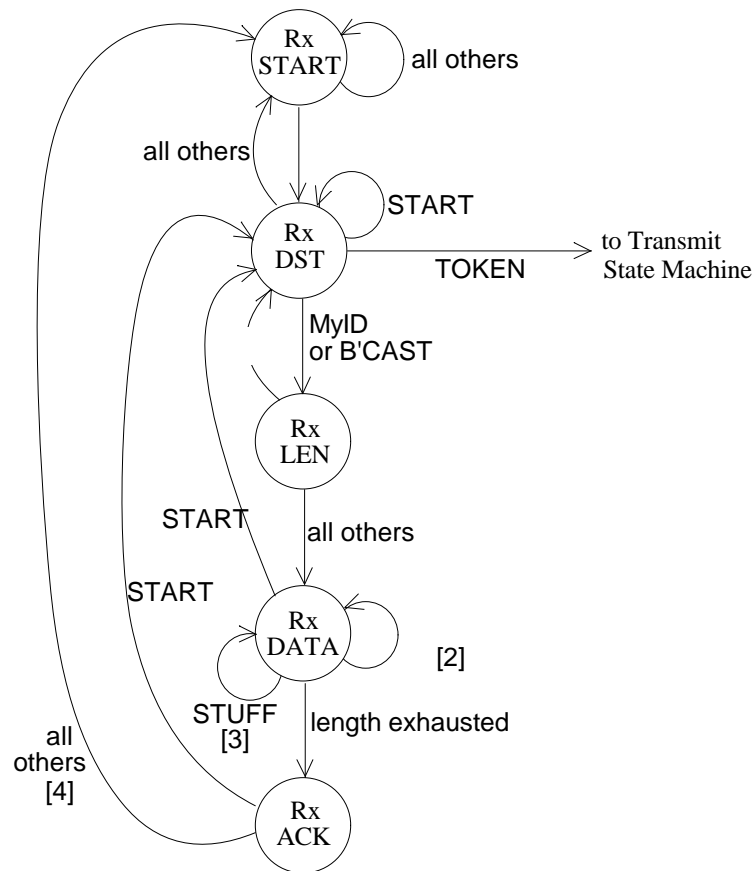
Since the ring is limited to 63 stations, the #stations field can only be incremented to 62, and the acknowledge byte can never be modified into any of the four reserved codes (0FC to 0FF hex).

When the ring owner is finished sending frames, it passes ownership to the next station in the ring with the two-byte sequence START TOKEN (0FF, 0FE hex). Using two bytes reduces the likelihood that a bit error will generate a spurious token. The station receiving the token may then assume ownership and begin transmitting frames, or, if it has no frames to transmit, will pass the token to its successor. When the ring is idle, the sequence START TOKEN continuously circulates, awaiting capture by the first station with data to send.

The Asynchronous Token Ring is managed by an interrupt-driven state machine in each processor. This software performs the functions of the IEEE Medium Access Control (MAC) sublayer. Unlike the 802.5 system, the failure of any processor will halt the flow of data around the ring. This was

deemed an acceptable compromise in order to use low-cost microprocessor hardware for the stations.

Figure B-2 illustrates the state machine for a listening station. All received bytes are retransmitted (except the ACK byte which may be modified before being resent). No data is added to or removed from the ring. Receive data for this station [2] is stored if a buffer is available; otherwise it is discarded, and the B bit is set in the ACK byte. Character stuffing is handled within the RxDATA state



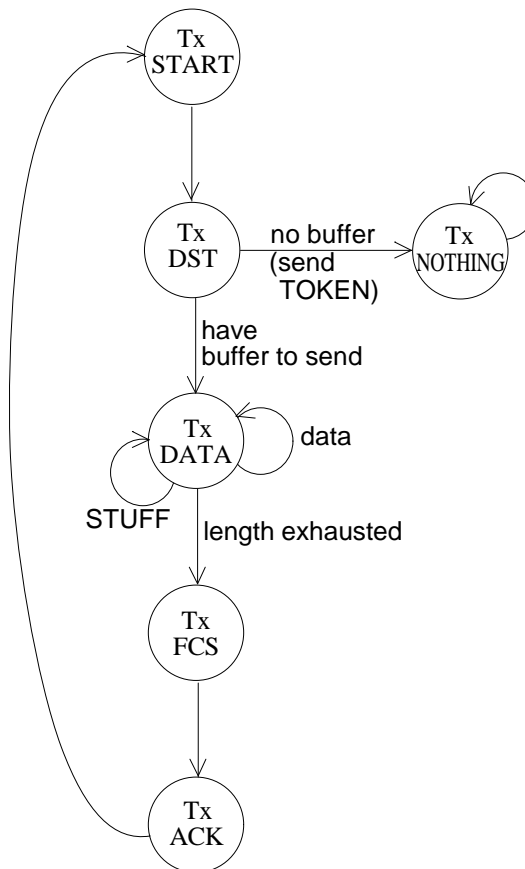
Notes:

1. All received characters are echoed (retransmitted).
2. Received data characters are stored if a buffer is available; otherwise they are discarded.
3. Character stuffing is handled within the RxDATA state, using an auxiliary state variable.
4. The ACK byte may be modified before it is echoed.

Figure B-2 Receive State Machine

[3], using an auxiliary state variable.

The state machine for a transmitting station (the ring owner) is much simpler, since it must merely put a frame as shown in Figure B-1 on the ring. As shown in Figure B-3, if no frame is waiting to be sent, the transmit state machine sends the token sequence, and then turns itself off. It will be restarted



Notes:

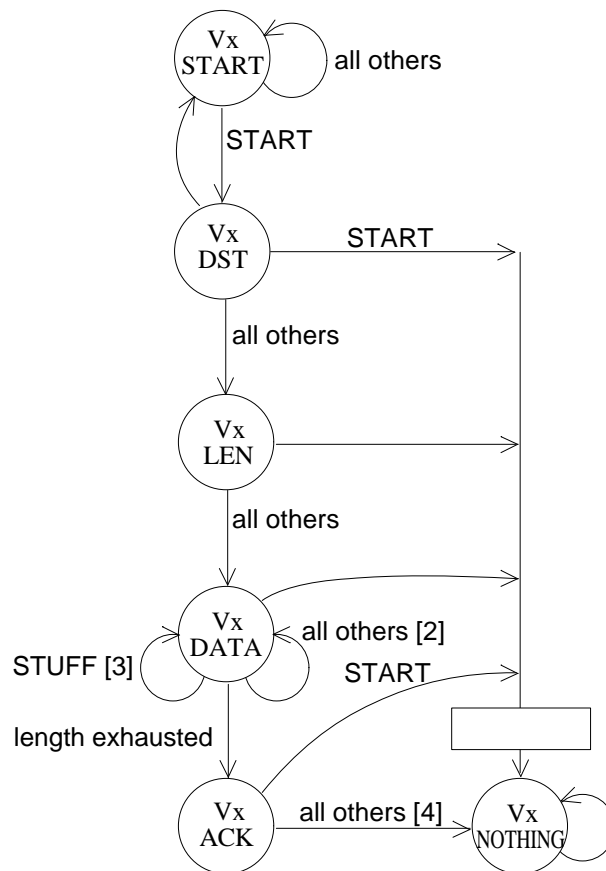
1. If no buffer is awaiting transmission, the TOKEN is sent and the transmit state machine halts.
2. Character stuffing is handled within the TxDATA state, using an auxiliary state variable.

Figure B-3 Transmit State Machine

by the receive state machine when a token is received.

While the transmit state machine is placing data on the ring, a parallel state machine (Figure B-

4) removes the data from the ring and compares it to the transmitted data. This "verify" operation cannot be performed synchronously with the transmission, because of the variable and unknown delay for bytes to transit the ring. The verify state machine is essentially identical to the receive state machine, with the following significant differences. Received bytes are consumed instead of being echoed (thus removing the bytes that were placed on the ring by the transmit state machine). Received bytes are not stored;



Notes:

1. All received characters are consumed.
2. Received data characters are compared to the contents of the transmit buffer.
3. Character stuffing is handled within the VxDATA state, using an auxiliary state variable.
4. No valid ACK byte can be a START code.

Figure B-4 Verify State Machine

instead, they are compared with the transmit buffer. If a mismatch is detected, the buffer is flagged with a "verify" error. Receipt of a START code during the frame will abort the verify, and flag the buffer with an "abort" error. Finally, all bytes preceding the START code are consumed: as will be seen shortly, this is the mechanism by which the ring is purged of spurious data.

Only one frame may be on the ring at any time. Once a frame has been sent, the sender must wait for that frame to be verified before sending the next frame (or passing the token). This decreases ring utilization; in a ring with five stations, an idle time of five byte periods will occur between frames. The advantage is a great simplification of the verify logic. More importantly, an idle period is needed to recover from asynchronous framing errors.

B.4 Error Handling and Ring Supervision

Most bit errors will result in a garbled byte of data. Single-bit errors and many multiple-bit errors in the destination, length, source, or data field will be identified by the checksum. The receiving station will discard the frame, and report the error to the sending station through the E bit of the acknowledge byte. (In addition, all multiple-bit errors will be detected by the verify logic.)

Bit errors in the length field will change the apparent length of the frame. It is extremely unlikely that the result will be a valid frame, but the erroneous length may lead to other problems. If the length is decreased by the error, the tail of the frame will appear to be spurious data on the ring. If the length is increased, a complete frame will never be seen. Recovery from both of these conditions will be discussed below. A length increase could also cause a buffer overflow; to prevent this, the receive state machine always limits the length to the maximum buffer size (63 bytes of data).

Bit errors in the START byte will prevent the receiving stations from recognizing the frame. A bit error which creates a spurious START code will convert one valid frame into two invalid frames, both of which will be rejected.

Bit errors in the TOKEN byte can cause the loss of the token, discussed below. An erroneous TOKEN code in the middle of a frame will be ignored (treated as bad data). Only a multiple-bit error in

the destination address field can create a spurious token. Should this happen, two stations may begin putting frames on the ring. It is conceivable that each station will exactly consume the other's frames (causing both stations to see verify errors). But eventually, one station will pass the TOKEN before the other. TOKENs are simply consumed by the verify state machine, since any station running the verify state machine already considers itself the ring owner. Thus the spurious TOKEN will be destroyed, and normal operation will resume.

Bit errors in the acknowledge byte are an unresolved problem. If a single-bit error clears the E or B bits, the transmitting station may incorrectly conclude that the frame was successfully received. (In the IEEE 802.5 protocol, this problem was addressed by sending the flag bits twice.) This will be a subject for future development.

Unlike IEEE 802.5 (and other bit synchronous systems), a garbled bit in an asynchronous ring can cause the apparent *loss* of a byte, or the addition of a *spurious* byte. Also, the corruption of a start bit can cause a loss of byte synchronization (continuous framing errors); this is handled by requiring each frame to "clear" the ring before another frame may be placed on the ring.

A spurious byte may occur before a frame, within a frame, or after a frame. Spurious bytes preceding a frame will be removed by the verify state machine, while it waits for a valid START code. Spurious bytes following a frame will appear to be spurious bytes preceding the next frame, and will likewise be removed. Note that even spurious bytes preceding the START TOKEN sequence will eventually appear as bytes preceding a frame. Thus by requiring the verify state machine to discard invalid START bytes, the ring is regularly purged.

A spurious byte inserted within a frame will corrupt the remainder of the frame and increase its length by one, appearing to the listening stations as corrupted data followed by a spurious byte. The frame will be flagged as erroneous, and the "appended" byte will be discarded as just described.

"Lost" data bytes are a difficult problem. If a data byte is lost around the ring, the verify state machine never sees a completed frame. Since it must remove the entire frame from the ring before

sending the next frame or passing the token, the ring will halt. This problem is addressed by a timeout counter. If the ring owner sees no receive (verify) data for 165 msec,¹ it declares that frame to be "timed out." The frame is flagged as unsuccessfully sent due to a timeout error, and the owner proceeds to send the next message or pass the token. Receiving stations awaiting completion of the frame will be reset by the START code of the new message or token.

It is also possible to lose the token, especially during idle periods when the token is circulating continuously on the ring. When this happens, the station sending the token sees that the token has been passed, and considers itself no longer the ring owner. But no station receives the token to become the new owner. If the token was garbled, the result is a ring of slaves endlessly circulating a nonsense byte. To handle this problem, one station is designated the "supervisor" of the ring. It monitors the ring constantly for the presence of a token. If 385 msec² elapses without a token being seen, it declares a "supervisor timeout" and generates a new token to be placed on the ring. The "non supervisor" stations also detect the loss of token; however, their action is merely to set a "slave timeout" flag. This flag may be used by higher level software to detect network failure.

Station number 01 is designated the supervisor. This is appropriate for this application, where the ring consists of several embedded controllers plus one PC as an operator's station. A second PC may be installed as a "monitor" station, to passively observe and tabulate ring traffic.

B.5 Broadcast Protocols

The handling of broadcast messages was a particular goal in the design of the Asynchronous Token Ring. A major advantage of physical ring protocols like IEEE 802.5 is the inherent ability to acknowledge a broadcast message, since every station may modify the acknowledge byte.

1. Three ticks of the 18.2 Hz system clock. If the first clock tick occurs immediately after the verify engine starts, the second tick could be seen in as little as 55 msec. At 9600 baud, each station adds a minimum delay of 1 msec. Thus a network of 55 stations could cause a false timeout, if the counter was set to two clock ticks.

2. Seven ticks of the 18.2 Hz clock. This timeout delay is a compromise. If too long, the network can go "down" for an excessively long period. If too short, heavy traffic on the network (each station sending packets before passing the token) can cause spurious timeouts. Further research may yield a more satisfactory solution to this problem.

IEEE 802.5 allows each station to flag "address recognized" and "frame copied" (successfully received). For this application, it is more useful to indicate "frame *not* copied." In the case of a broadcast message, any station may set this bit; when the sender sees this bit set, it knows that at least one station failed to copy the message, and a retransmission is required. The acknowledgement differentiates between messages which must be resent due to data errors (the E bit), and due to the lack of a receive buffer (the B bit).

An innovation here is the use of an address-recognized *count*, rather than a single bit. For messages to a single destination, this count will be returned to the sender as either 0 (address not recognized), or 1 (address recognized). For broadcast messages, the count is the number of stations recognizing the broadcast address (regardless of whether they successfully copied the message). If the sender knows the number of stations on the ring, it can detect whether any station failed to recognize the message.

The Asynchronous Token Ring essentially provides an "acknowledged datagram" service (Tanenbaum, 1989). This is a "connectionless" service which does not guarantee message sequence, but does indicate successful receipt. Two kinds of message delivery can be offered to the application. "At most once" transmits the message only once. No destination will receive a duplicate of the message, but some destinations may fail to receive it at all. "At least once" delivery retransmits the message until it is successfully received by all addressed destinations (the E and B bits return clear, and the message verifies successfully). Every destination receives the message, but some may receive duplicate copies. Rejection of duplicate messages is handled at a higher protocol layer.

B.6 Message Interpretation

It became necessary to write the received-message processing software, before the content of received messages was finalized. To do this, an "open ended" message format was devised. Each data field is a statement in the tokenized ITL³ language. When the token ring state machine receives a valid

3. The Inferencing Token Language is described in Chapter 3.

message, it is passed to the ITL interpreter. Some ITL tokens are defined specifically for message processing (see Appendix D); others -- such as ASK and TELL -- represent network-related actions which must be taken by the processor. Since new tokens can be added to the ITL language at any time, this provides a very flexible message processing mechanism.

This approach allowed a simple method to reject duplicate messages. The SEQUENCE token and a sequence number are placed by the sender at the beginning of a message. This token has the following action: if the sequence number matches the last sequence number received *from this sender*, terminate ITL processing, ignoring the rest of the message. The sender increments the sequence number for each new message to the same destination. The guaranteed delivery of the "at least once" service ensures that sequence numbers will not be skipped.

This also allows sequencing of broadcast messages. Each sender maintains a distinct "outgoing" sequence number for broadcasts. Each recipient of the broadcast message, upon encountering the SEQUENCE token, compares the sequence number to that of the last *broadcast* message from that sender. Thus each station maintains two "incoming" sequence numbers for each possible sender: one for point-to-point messages, and one for broadcast.

B.7 Future Work

Several refinements of the protocol are possible. The implementation of the acknowledge byte is weak; it should be sent redundantly (as in 802.5) or with an error-check code (such as a parity bit). With the current acknowledge format, only 64 of the 252 valid station addresses can be used. Some of the remaining 188 could be used for group addresses. Also, the maximum message length could be increased from 63 to 251, although in this case a better frame check sequence would be advisable.

At present, one station (the supervisor) is crucial to ring operation. It would be better if any station could assume the role of ring supervisor, as in IEEE 802.5.

The current network uses RS-232 links, star-wired to a passive hub -- that is, serial cables from each station are brought to a central switchbox. This allows any station to be manually bypassed. An

electronic switch that automatically bypasses any inactive station would let the ring recover from a "dead" CPU.

The potential of "active" messages which are executed by an interpreter has yet to be explored. Use of the ITL interpreter opens the possibility of distributed execution of ITL language -- processors sending logic expressions and even entire subprograms to other processors over the network, where they are executed to produce similar "return" messages.