to determine the length: either reserved framing codes or an explicit length count. A length count is chosen to keep the system simple. Framing codes would require that certain data values be off limits to the contents of the message. The payload length is bounded at a convenient binary threshold: 255 bytes. For simple control and data-acquisition applications, this is probably more than enough.

Based on these basic requirements and a couple of quick decisions, a packet format quickly emerges. A three-byte, fixed-length header shown in Table 5.4 is followed by a variable-length payload. No trailer is necessary in this network.

TABLE 5.4 Hypothetical Packed Header Format

Field Name	Byte	Bits	Description
DA	0	[7:0]	Destination address (0xFF = multicast)
SA	1	[7:0]	Source address
LEN	2	[7:0]	Payload length (0x0 = no payload present)

The eight-bit destination address field, DA, comes first to enable the receiving hardware to quickly determine whether the packet should be accepted by the node or ignored. A packet will be accepted if DA matches the receiver's node address, or if DA equals 0xFF, indicating a broadcast packet. At the end of the header is an eight-bit length field that indicates how many payload bytes are present after the fixed-length header. This limits the maximum packet size to 255 payload bytes plus the 3-byte header. A value of zero means that there is no payload, only a header in the packet.

Error detection can be handled by even parity. Each byte of the header and payload is sent with an accompanying parity bit. When an error is detected, the network's behavior must be clearly defined to prevent the system from either ceasing to function or acting on false data. Parity errors can manifest themselves in a variety of tricky ways. For example, if the length field has a parity error, how will the receiver know the true end of the frame? Without proper planning, a parity error on the length field can permanently knock the receivers out of sync and make automatic recovery impossible. This extreme situation can occur when an invalid length causes the receiver to either skip over the next frame header or prematurely interpret the end of the current frame as a new header. In both cases, the receiver will falsely interpret a bogus length field, and the cycle of false header detection can continue indefinitely.

If a parity error is detected on either the destination or source addresses, the receivers will not lose synchronization, but the packet should be ignored, because it cannot be known who the true recipient or sender of the packet is.

Fault tolerance in the case of an invalid payload length can be handled in a relatively simple manner. Requirements of no intrapacket gaps and a minimum interpacket gap assist in recovery from length-field parity errors. The absence of intrapacket gaps means that, once a packet has begun transmission, its bytes must be continuous without gaps. Related to this is the requirement of a minimum interpacket gap which forces a minimum idle period between the last byte of one packet and the start of the next packet. These requirements help each receiver determine when packets are starting and ending. Even if a packet has been subjected to parity errors, the receiver can wait until the current burst of traffic has ended, wait for the minimum interpacket gap, and then begin looking for the next packet to begin.

The parity error detection and accompanying recovery scheme greatly increases the probability that false data will not be acted upon as correct data and that the entire network will not stop functioning when it encounters an arbitrary parity error. However, error detection is all about probability.

A single parity bit cannot guarantee the detection of multiple errors in the same byte, because such errors can mask themselves. For example, two bit errors can flip a data bit and the parity bit itself, making it impossible for the receiver to detect the error. More complex error detection schemes are available and are more difficult to fool. Although no error detection solution is perfect, some schemes reduce the probability of undetected errors to nearly zero.

If a packet is received with an error, it cannot be acted upon normally, because its contents are suspect. For the purposes of devising a useful error-handling scheme, packet errors can be divided into two categories: those that corrupt the destination/source address information and those that do not. Parity errors that corrupt the packet's addresses must result in the packet being completely ignored, because the receiving node is unable to generate a reply message to the originator indicating that the packet was corrupted. If the source address is corrupted, the receiver does not know to whom to reply. If the destination address is corrupted, the receiver does not know whether it is the indented recipient.

In the case of an address error in which the received packet is ignored, the originator must implement some mechanism to recover from the packet loss rather than waiting indefinitely for a reply that will never arrive. A reply *timeout* can be implemented by an originator each time a packet is sent that requires a corresponding reply. A timeout is an arbitrary delay during which an originating node waits before giving up on a response from a remote node. Timeouts are common in networks because, if a packet is lost due to an error, the originator should not wait indefinitely for a response that will never come. Establishing a timeout value is a compromise between not giving up too quickly and missing a slower-than-normal reply and waiting too long and introducing unacceptable delays in system functionality when a packet is lost. Depending on the time it takes to send a packet on a network and the nodes' typical response time, timeouts can range from microseconds to minutes. Typical timeouts are often expressed in milliseconds.

When an originator times-out and concludes that its requested data somehow got lost, it can resend the request. If, for example, a security control node sends a request for a camera to pan across a room, and that request is not acknowledged within half a second, the request can be retransmitted.

In the case of a non-address error, the receiving node has enough information to send a reply back to the originator, informing it that the packet was not correctly received. Such behavior is desirable to enable the originator to retransmit the packet rather than waiting for a timeout before resending that data.

The preceding details of a hypothetical RS-485 network must be gathered into network driver software to enable proper communication across the network. While hardware controls the detection of parity errors and the flow of bits, it is usually software that generates reply messages and counts down timeouts. Figure 5.17 distills this information into a single flowchart from which software routines could be written.

As seen from this flowchart, transmit and receive processes run concurrently and are related. The transmit process does not complete until a positive acknowledgement is received from the destination node. This network control logic implemented in software is simple by mainstream networking standards, yet it is adequate for networks of limited size and complexity. Issues such as access sharing are handled inherently by the request/reply nature of this network, greatly simplifying the traffic patterns that must be handled by the software driver.

5.11 INTERCHIP SERIAL COMMUNICATIONS

Serial data links are not always restricted to long-distance communications. Within a single computer system, or even a single circuit board, serial links can provide attractive benefits as compared