In this discussion, we've only briefly touched on VLANs and have focused on port-based VLANs. We should also mention that VLANs can be defined in several other ways. In MAC-based VLANs, the network manager specifies the set of MAC addresses that belong to each VLAN; whenever a device attaches to a port, the port is connected into the appropriate VLAN based on the MAC address of the device. VLANs can also be defined based on network-layer protocols (e.g., IPv4, IPv6, or Appletalk) and other criteria. See the 802.1Q standard [IEEE 802.1q 2005] for more details.

5.7 PPP: The Point-to-Point Protocol

Most of our discussion of link-layer protocols thus far has focused on protocols for broadcast channels. In this section we cover a link-layer protocol for point-to-point links—PPP, the point-to-point protocol. Because PPP is typically the protocol of choice for a dial-up link from a residential host, it is undoubtedly one of the most widely deployed link-layer protocols today. The other important link-layer protocol in use today is the high-level data link control (HDLC) protocol; see [Spragins 1991] for a discussion of HDLC. Our discussion here of the simpler PPP protocol will allow us to explore many of the most important features of a point-to-point link-layer protocol.

As its name implies, the point-to-point protocol (PPP) [RFC 1661; RFC 2153] is a link-layer protocol that operates over a **point-to-point link**—a link directly connecting two nodes, one on each end of the link. The point-to-point link over which PPP operates might be a serial dial-up telephone line (for example, a 56K modem connection), a SONET/SDH link, an X.25 connection, or an ISDN circuit. As noted above, PPP is often the protocol of choice for connecting home users to their ISPs over a dial-up connection.

Before diving into the details of PPP, it is instructive to examine the original requirements that the IETF placed on the design of PPP [RFC 1547]:

- *Packet framing*. The PPP protocol link-layer sender must be able to take a network-level packet and encapsulate it within the PPP link-layer frame such that the receiver will be able to identify the start and end of both the link-layer frame and the network-layer packet within the frame.
- *Transparency.* The PPP protocol must not place any constraints on data appearing on the network-layer packet (headers or data). Thus, for example, PPP cannot forbid the use of certain bit patterns in the network-layer packet. We'll return to this issue shortly in our discussion of byte stuffing.
- *Multiple network-layer protocols.* The PPP protocol must be able to support multiple network-layer protocols (for example, IP and DECnet) running over the same physical link at the same time. Just as the IP protocol is required to multiplex different transport-level protocols (for example, TCP and UDP) over a

single end-to-end connection, so too must PPP be able to multiplex different network-layer protocols over a single point-to-point connection. This requirement means that at a minimum, PPP will likely require a protocol type field or some similar mechanism so the receiving-side PPP can demultiplex a received frame up to the appropriate network-layer protocol.

- *Multiple types of links.* In addition to being able to carry multiple higher-level protocols, PPP must also be able to operate over a wide variety of link types, including links that are either serial (transmitting a bit at a time in a given direction) or parallel (transmitting bits in parallel), synchronous (transmitting a clock signal along with the data bits) or asynchronous, low-speed or high-speed, electrical or optical.
- *Error detection.* A PPP receiver must be able to detect bit errors in the received frame.
- *Connection liveness.* PPP must be able to detect a failure at the link level (for example, the inability to transfer data from the sending side of the link to the receiving side of the link) and signal this error condition to the network layer.
- *Network-layer address negotiation.* PPP must provide a mechanism for the communicating network layers (for example, IP) to learn or configure each other's network-layer address.
- *Simplicity.* PPP was required to meet a number of additional requirements beyond those listed above. On top of all of these requirements, first and foremost is simplicity. RFC 1547 states, "The watchword for a point-to-point protocol should be simplicity." A tall order indeed, given all of the other requirements placed on the design of PPP! Nearly 100 RFCs now define the various aspects of this "simple" protocol.

While it may appear that many requirements were placed on the design of PPP, the situation could actually have been much more difficult! The design specifications for PPP also explicitly note protocol functionality that PPP was *not* required to implement:

- *Error correction.* PPP is required to detect bit errors but is *not* required to correct them.
- *Flow control.* A PPP receiver is expected to be able to receive frames at the full rate of the underlying physical layer. If a higher layer cannot receive packets at this full rate, it is then up to the higher layer to drop packets or throttle the sender at the higher layer. That is, rather than having the PPP sender throttle its own transmission rate, it is the responsibility of a higher-level protocol to throttle the rate at which packets are delivered to PPP for sending.
- *Sequencing.* PPP is *not* required to deliver frames to the link receiver in the same order in which they were sent by the link sender. It is interesting to note that while this flexibility is compatible with the IP service model (which allows IP packets to be delivered end-to-end in any order), other network-layer protocols that operate over PPP do require sequenced end-to-end packet delivery.

5.7.1 PPP Data Framing

Figure 5.33 shows a PPP data frame that uses HDLC-like framing [RFC 1662]. The PPP frame contains the following fields:

- *Flag field.* Every PPP frame begins and ends with a 1-byte flag field with a value of 01111110.
- *Address field.* The only possible value for this field is 11111111.
- *Control field.* The only possible value for this field is 00000011. Because both the address and control fields can take only a fixed value, you might wonder why the fields are defined in the first place. The PPP specification [RFC 1662] states that other values "may be defined at a later time," although none has been defined to date. Because these fields take fixed values, PPP allows the sender to simply not send the address and control bytes, thus saving 2 bytes of overhead in the PPP frame.
- *Protocol.* The protocol field tells the PPP receiver the upper-layer protocol to which the received encapsulated data (that is, the contents of the PPP frame's information field) belongs. On receipt of a PPP frame, the PPP receiver will check the frame for correctness and then pass the encapsulated data on to the appropriate protocol. RFC 1700 and RFC 3232 define the 16-bit protocol codes used by PPP. Of interest to us is the IP protocol (that is, the data encapsulated in the PPP frame is an IP datagram), which has a value of 21 hexadecimal; other network-layer protocols such as AppleTalk (29) and DECnet (27).
- *Information.* This field contains the encapsulated packet (data) that is being sent by an upper-layer protocol (for example, IP) over the PPP link. The default

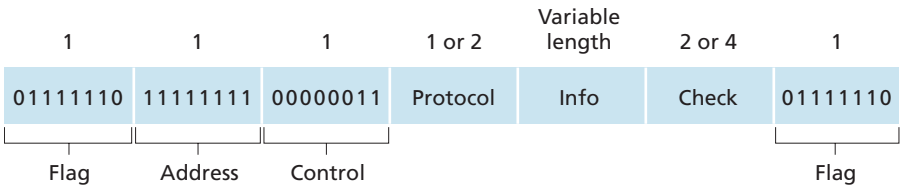

Figure 5.33 ♦ PPP data frame format

maximum length of the information field is 1,500 bytes, although this can be changed when the link is first configured, as discussed below.

- *Checksum.* The checksum field is used to detect bit errors in a transmitted frame. It uses either a 2- or 4-byte HDLC-standard cyclic redundancy code.

Byte Stuffing

Before closing our discussion of PPP framing, let us consider a problem that arises when any protocol uses a specific bit pattern in a flag field to delineate the beginning or end of the frame. What happens if the flag pattern itself occurs elsewhere in the packet? For example, what happens if the flag field value of 01111110 appears in the information field? Will the receiver incorrectly detect the end of the PPP frame?

One way to solve this problem would be for PPP to forbid the upper-layer protocol from sending data containing the flag field bit pattern. The PPP requirement of transparency discussed above obviates this possibility. An alternative solution, and the one taken in PPP and many other protocols, is to use a technique known as **byte stuffing**.

PPP defines a special control escape byte, 01111101. If the flag sequence, 01111110, appears anywhere in the frame, except in the flag field, PPP precedes that instance of the flag pattern with the control escape byte. That is, it "stuffs" (adds) a control escape byte into the transmitted data stream, before the 01111110, to indicate that the following 011111110 is *not* a flag value but is, in fact, actual data. A receiver that sees a 01111110 preceded by a 01111101 will, of course, remove the stuffed control escape to reconstruct the original data. Similarly, if the control escape byte bit pattern itself appears as actual data, it too must be preceded by a stuffed control escape byte. Thus, when the receiver sees a single control escape byte by itself in the data stream, it knows that the byte was stuffed into the data stream. A pair of control escape bytes occurring back to back means that one instance of the control escape byte appears in the original data being sent. Figure 5.34 illustrates PPP byte stuffing.

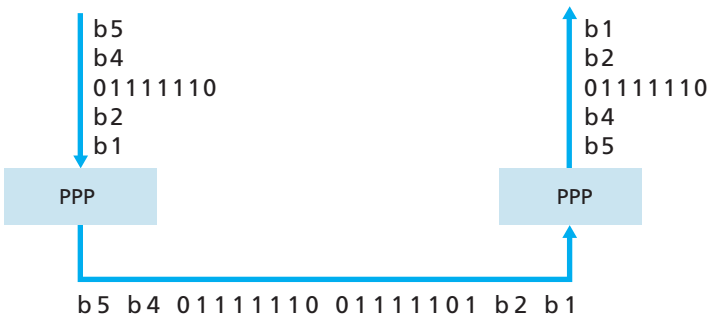

Figure 5.34 ♦ Byte stuffing

(Actually, PPP also XORs the data byte being escaped with 20 hexadecimal, a detail we omit here for simplicity.)

We remark that PPP also has a link control protocol (LCP) whose job it is to perform initialization, maintenance, and shutdown of a PPP link. LCP is discussed in some detail in the online material associated with this book.