3

Link Layer

3.1 Introduction

In Chapter 1, we saw that the purpose of the link layer in the TCP/IP protocol suite is to send and receive IP datagrams for the IP module. It is also used to carry a few other protocols that help support IP, such as ARP (see Chapter 4). TCP/IP supports many different link layers, depending on the type of networking hardware being used: wired LANs such as Ethernet, *metropolitan area networks* (MANs) such as cable TV and DSL connections available through service providers, and wired voice networks such as telephone lines with modems, as well as the more recent wireless networks such as Wi-Fi (wireless LAN) and various wireless data services based on cellular technlology such as HSPA, EV-DO, LTE, and WiMAX. In this chapter we shall look at some of the details involved in using the Ethernet and Wi-Fi link layers, how the *Point-to-Point Protocol* (PPP) is used, and how link-layer protocols can be carried inside other (link- or higher-layer) protocols, a technique known as tunneling. Covering the details of every link technology available today would require a separate text, so we instead focus on some of the most commonly used link-layer protocols and how they are used by TCP/IP.

Most link-layer technologies have an associated protocol format that describes how the corresponding PDUs must be constructed in order to be carried by the network hardware. When referring to link-layer PDUs, we usually use the term *frame*, so as to distinguish the PDU format from those at higher layers such as packets or segments, terms used to describe network- and transport-layer PDUs, respectively. Frame formats usually support a variable-length frame size ranging from a few bytes to a few kilobytes. The upper bound of the range is called the *maximum transmission unit* (MTU), a characteristic of the link layer that we shall encounter numerous times in the remaining chapters. Some network technologies, such as modems and serial lines, do not impose their own maximum frame size, so they can be configured by the user.

3.2 Ethernet and the IEEE 802 LAN/MAN Standards

The term *Ethernet* generally refers to a set of standards first published in 1980 and revised in 1982 by Digital Equipment Corp., Intel Corp., and Xerox Corp. The first common form of Ethernet is now sometimes called "10Mb/s Ethernet" or "shared Ethernet," and it was adopted (with minor changes) by the IEEE as standard number 802.3. Such networks were usually arranged like the network shown in Figure 3-1.

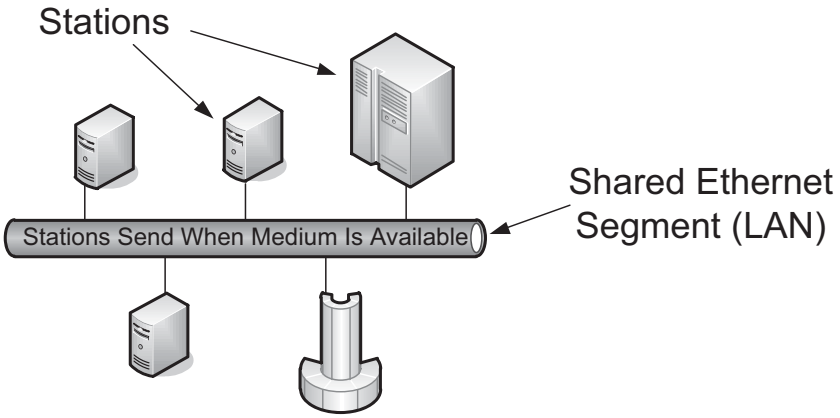

Figure 3-1 A basic shared Ethernet network consists of one or more stations (e.g., workstations, supercomputers) attached to a shared cable segment. Link-layer PDUs (frames) can be sent from one station to one or more others when the medium is determined to be free. If multiple stations send at the same time, possibly because of signal propagation delays, a collision occurs. Collisions can be detected, and they cause sending stations to wait a random amount of time before retrying. This common scheme is called carrier sense, multiple access with collision detection.

Because multiple stations share the same network, this standard includes a distributed algorithm implemented in each Ethernet network interface that controls when a station gets to send data it has. The particular method, known as *carrier sense, multiple access with collision detection* (CSMA/CD), mediates which computers can access the shared medium (cable) without any other special agreement or synchronization. This relative simplicity helped to promote the low cost and resulting popularity of Ethernet technology.

With CSMA/CD, a station (e.g., computer) first looks for a signal currently being sent on the network and sends its own frame when the network is free. This is the "carrier sense" portion of the protocol. If some other station happens to send at the same time, the resulting overlapping electrical signal is detected as a collision. In this case, each station waits a random amount of time before trying again. The amount of time is selected by drawing from a uniform probability distribution that doubles in length each time a subsequent collision is detected.

Eventually, each station gets its chance to send or times out trying after some number of attempts (16 in the case of conventional Ethernet). With CSMA/CD, only one frame is traveling on the network at any given time. Access methods such as CSMA/CD are more formally called *Media Access Control* (MAC) protocols. There are many types of MAC protocols; some are based on having each station try to use the network independently (contention-based protocols like CSMA/CD), and others are based on prearranged coordination (e.g., by allocating time slots for each station to send).

Since the development of 10Mb/s Ethernet, faster computers and infrastructure have driven the need for ever-increasing speeds in LANs. Given the popularity of Ethernet, significant innovation and effort have managed to increase its speed from 10Mb/s to 100Mb/s to 1000Mb/s to 10Gb/s, and now to even more. The 10Gb/s form is becoming popular in larger data centers and large enterprises, and speeds as high as 100Gb/s have been demonstrated. The very first (research) Ethernet ran at 3Mb/s, but the DIX (Digital, Intel, Xerox) standard ran at 10Mb/s over a shared physical cable or set of cable segments interconnected by electrical repeaters. By the early 1990s, the shared cable had largely been replaced by twisted-pair wiring (resembling telephone wires and often called "10BASE-T"). With the development of 100Mb/s (also called "fast Ethernet," the most popular version of which is known as "100BASE-TX"), contention-based MAC protocols have become less popular. Instead, the wiring between each LAN station is often not shared but instead provides a dedicated electrical path in a star topology. This can be accomplished with Ethernet *switches*, as shown in Figure 3-2.

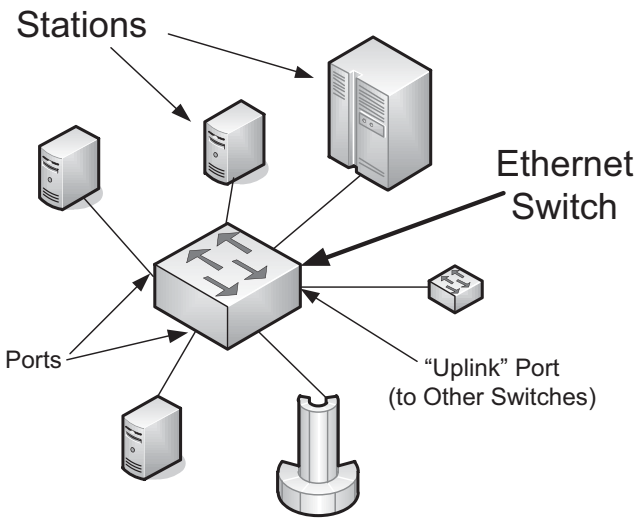

Figure 3-2 A switched Ethernet network consists of one or more stations, each of which is attached to a switch port using a dedicated wiring path. In most cases where switched Ethernet is used, the network operates in a full-duplex fashion and the CSMA/CD algorithm is not required. Switches may be cascaded to form larger Ethernet LANs by interconnecting switch ports, sometimes called "uplink" ports.

At present, switches are commonly used, providing each Ethernet station with the ability to send and receive data simultaneously (called "full-duplex Ethernet"). Although half-duplex (one direction at a time) operation is still supported even by 1000Mb/s Ethernet (1000BASE-T), it is rarely used relative to full-duplex Ethernet. We shall discuss how switches process PDUs in more detail later.

One of the most popular technologies used to access the Internet today is wireless networking, the most common for wireless local area networks (WLANs) being an IEEE standard known as Wireless Fidelity or *Wi-Fi*, and sometimes called "wireless Ethernet" or 802.11. Although this standard is distinct from the 802 wired Ethernet standards, the frame format and general interface are largely borrowed from 802.3, and all are part of the set of IEEE 802 LAN standards. Thus, most of the capabilities used by TCP/IP for Ethernet networks are also used for Wi-Fi networks. We shall explore each of these in more detail. First, however, it is useful to get a bigger picture of all of the IEEE 802 standards that are relevant for setting up home and enterprise networks. We also include references to those IEEE standards governing MAN standards, including IEEE 802.16 (WiMAX) and the standard for media-independent handoffs in cellular networks (IEEE 802.21).

3.2.1 The IEEE 802 LAN/MAN Standards

The original Ethernet frame format and operation were described by industry agreement, mentioned earlier. This format was known as the DIX format or Ethernet II format. This type of Ethernet network, with slight modification, was later standardized by the IEEE as a form of CSMA/CD network, called 802.3. In the world of IEEE standards, standards with the prefix 802 define the operations of LANs and MANs. The most popular 802 standards today include 802.3 (essentially Ethernet) and 802.11 (WLAN/Wi-Fi). These standards have evolved over time and have changed names as freestanding amendments (e.g., 802.11g) are ultimately incorporated in revised standards. Table 3-1 shows a fairly complete list of the IEEE 802 LAN and MAN standards relevant to supporting the TCP/IP protocols, as of mid-2011.

Table 3-1 LAN and MAN IEEE 802 standards relevant to the TCP/IP protocols (2011)

Name	Description	Official Reference
802.1ak	Multiple Registration Protocol (MRP)	[802.1AK-2007]
802.1AE	MAC Security (MACSec)	[802.1AE-2006]
802.1AX	Link Aggregation (formerly 802.3ad)	[802.1AX-2008]
802.1d	MAC Bridges	[802.1D-2004]
802.1p	Traffic classes/priority/QoS	[802.1D-2004]
802.1q	Virtual Bridged LANs/Corrections to MRP	[802.1Q-2005/Cor1-2008]
802.1s	Multiple Spanning Tree Protocol (MSTP)	[802.1Q-2005]

Table 3-1 LAN and MAN IEEE 802 standards relevant to the TCP/IP protocols (2011) (*continued*)

Name	Description	Official Reference
802.1w	Rapid Spanning Tree Protocol (RSTP)	[802.1D-2004]
802.1X	Port-Based Network Access Control (PNAC)	[802.1X-2010]
802.2	Logical Link Control (LLC)	[802.2-1998]
802.3	Baseline Ethernet and 10Mb/s Ethernet	[802.3-2008] (Section One)
802.3u	100Mb/s Ethernet ("Fast Ethernet")	[802.3-2008] (Section Two)
802.3x	Full-duplex operation and flow control	[802.3-2008]
802.3z/802.3ab	1000Mb/s Ethernet ("Gigabit Ethernet")	[802.3-2008] (Section Three)
802.3ae	10Gb/s Ethernet ("Ten-Gigabit Ethernet")	[802.3-2008] (Section Four)
802.3ad	Link Aggregation	[802.1AX-2008]
802.3af	Power over Ethernet (PoE) (to 15.4W)	[802.3-2008] (Section Two)
802.3ah	Access Ethernet ("Ethernet in the First Mile (EFM)")	[802.3-2008] (Section Five)
802.3as	Frame format extensions (to 2000 bytes)	[802.3-2008]
802.3at	Power over Ethernet enhancements ("PoE+", to 30W)	[802.3at-2009]
802.3ba	40/100Gb/s Ethernet	[802.3ba-2010]
802.11a	54Mb/s Wireless LAN at 5GHz	[802.11-2007]
802.11b	11Mb/s Wireless LAN at 2.4GHz	[802.11-2007]
802.11e	QoS enhancement for 802.11	[802.11-2007]
802.11g	54Mb/s Wireless LAN at 2.4GHz	[802.11-2007]
802.11h	Spectrum/power management extensions	[802.11-2007]
802.11i	Security enhancements/replaces WEP	[802.11-2007]
802.11j	4.9–5.0GHz operation in Japan	[802.11-2007]
802.11n	6.5–600Mb/s Wireless LAN at 2.4 and 5GHz using optional MIMO and 40MHz channels	[802.11n-2009]
802.11s (draft)	Mesh networking, congestion control	Under development
802.11y	54Mb/s wireless LAN at 3.7GHz (licensed)	[802.11y-2008]
802.16	Broadband Wireless Access Systems (WiMAX)	[802.16-2009]
802.16d	Fixed Wireless MAN Standard (WiMAX)	[802.16-2009]
802.16e	Fixed/Mobile Wireless MAN Standard (WiMAX)	[802.16-2009]
802.16h	Improved Coexistence Mechanisms	[802.16h-2010]
802.16j	Multihop Relays in 802.16	[802.16j-2009]
802.16k	Bridging of 802.16	[802.16k-2007]
802.21	Media Independent Handovers	[802.21-2008]

Other than the specific types of LAN networks defined by the 802.3, 802.11, and 802.16 standards, there are some related standards that apply across all of the IEEE standard LAN technologies. Common to all three of these is the 802.2 standard that defines the *Logical Link Control* (LLC) frame header common among many of the 802 networks' frame formats. In IEEE terminology, LLC and MAC are "sublayers" of the link layer, where the LLC (mostly frame format) is generally common to each type of network and the MAC layer may be somewhat different. While the original Ethernet made use of CSMA/CD, for example, WLANs often make use of CSMA/CA (CA is "collision avoidance").

Note

Unfortunately the combination of 802.2 and 802.3 defined a different frame format from Ethernet II until 802.3x finally rectified the situation. It has been incorporated into [802.3-2008]. In the TCP/IP world, the encapsulation of IP datagrams is defined in [RFC0894] and [RFC2464] for Ethernet networks, although the older LLC/SNAP encapsulation remains published as [RFC1042]. While this is no longer much of an issue, it was once a source of concern, and similar issues occasionally arise [RFC4840].

The frame format has remained essentially the same until fairly recently. To get an understanding of the details of the format and how it has evolved, we now turn our focus to these details.

3.2.2 The Ethernet Frame Format

All Ethernet (802.3) frames are based on a common format. Since its original specification, the frame format has evolved to support additional functions. Figure 3-3 shows the current layout of an Ethernet frame and how it relates to a relatively new term introduced by IEEE, the IEEE *packet* (a somewhat unfortunate term given its uses in other standards).

The Ethernet frame begins with a *Preamble* area used by the receiving interface's circuitry to determine when a frame is arriving and to determine the amount of time between encoded bits (called *clock recovery*). Because Ethernet is an asynchronous LAN (i.e., precisely synchronized clocks are not maintained in each Ethernet interface card), the space between encoded bits may differ somewhat from one interface card to the next. The preamble is a recognizable pattern (0xAA typically), which the receiver can use to "recover the clock" by the time the *start frame delimiter* (SFD) is found. The SFD has the fixed value 0xAB.

Note

The original Ethernet encoded bits using a *Manchester Phase Encoding* (MPE) with two voltage levels. With MPE, bits are encoded as voltage transitions rather than absolute values. For example, the bit 0 is encoded as a transition from -0.85 to +0.85V, and a 1 bit is encoded as a +0.85 to -0.85V transition (0V indicates

that the shared wire is idle). The 10Mb/s Ethernet specification required network hardware to use an oscillator running at 20MHz, because MPE requires two clock cycles per bit. The bytes 0xAA (10101010 in binary) present in the Ethernet preamble would be a square wave between +0.85 and -0.85V with a frequency of 10MHz. Manchester encoding was replaced with different encodings in other Ethernet standards to improve efficiency.

This basic frame format includes 48-bit (6-byte) *Destination* (*DST*) and *Source* (*SRC*) *Address* fields. These addresses are sometimes known by other names such as "MAC address," "link-layer address," "802 address," "hardware address," or "physical address." The destination address in an Ethernet frame is also allowed to address more than one station (called "broadcast" or "multicast"; see Chapter 9). The broadcast capability is used by the ARP protocol (see Chapter 4) and multicast capability is used by the ICMPv6 protocol (see Chapter 8) to convert between network-layer and link-layer addresses.

Following the source address is a *Type* field that doubles as a *Length* field. Ordinarily, it identifies the type of data that follows the header. Popular values used with TCP/IP networks include IPv4 (0x0800), IPv6 (0x86DD), and ARP (0x0806). The value 0x8100 indicates a Q-tagged frame (i.e., one that can carry a "virtual LAN" or VLAN ID according to the 802.1q standard). The size of a basic Ethernet frame is 1518 bytes, but the more recent standard extended this size to 2000 bytes.

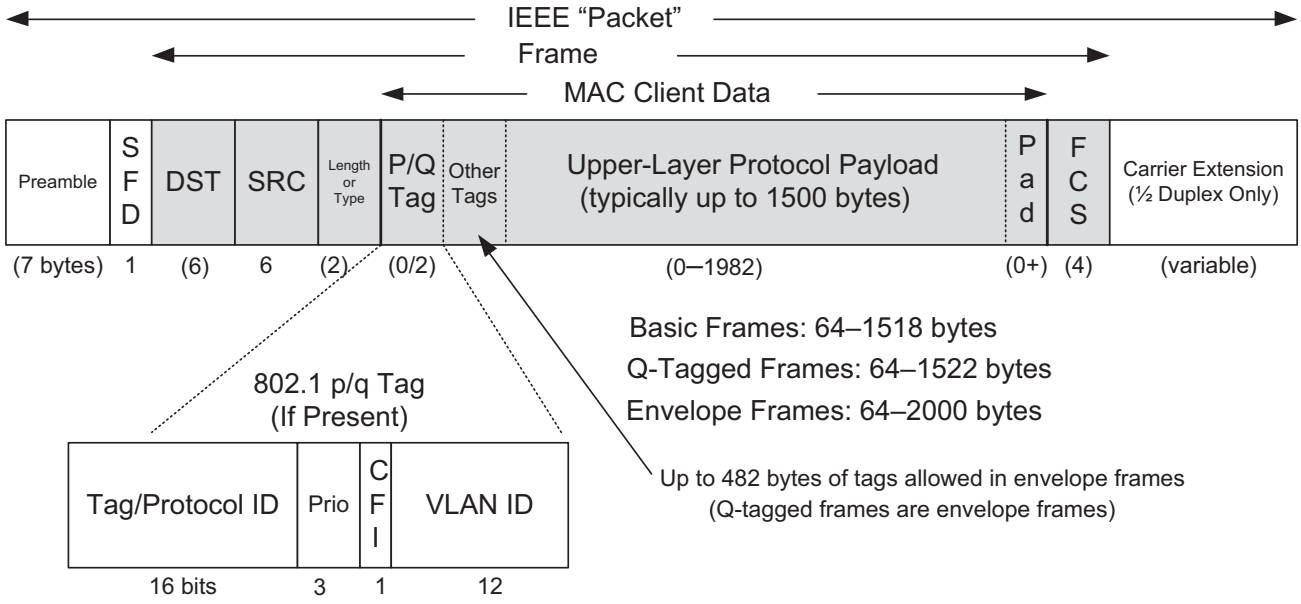

Figure 3-3 The Ethernet (IEEE 802.3) frame format contains source and destination addresses, an overloaded *Length/Type* field, a field for data, and a frame check sequence (a CRC32). Additions to the basic frame format provide for a tag containing a VLAN ID and priority information (802.1p/q) and more recently for an extensible number of tags. The preamble and SFD are used for synchronizing receivers. When half-duplex operation is used with Ethernet running at 100Mb/s or more, additional bits may be appended to short frames as a carrier extension to ensure that the collision detection circuitry operates properly.

Note

The original IEEE (802.3) specification treats the *Length/Type* field as a *Length* field instead of a *Type* field. The field is thereby *overloaded* (used for more than one purpose). The trick is to look at the value of the field. Today, if the value in the field is greater than or equal to 1536, the field must contain a type value, which is assigned by standards to have values exceeding 1536. If the value of the field is 1500 or less, the field indicates the length. The full list of types is given by [ETHERTYPES].

Following the *Destination* and *Source Address* fields, [802.3-2008] provides for a variable number of *tags* that contain various protocol fields defined by other IEEE standards. The most common of these are the tags used by 802.1p and 802.1q, which provide for virtual LANs and some *quality-of-service* (QoS) indicators. These are discussed in Section 3.2.3.

Note

The current [802.3-2008] standard incorporates the frame format modifications of 802.3 as that provides for a maximum of 482 bytes for holding "tags" to be carried with each Ethernet frame. These larger frames, called *envelope frames*, may be up to 2000 bytes in length. Frames containing 802.1p/q tags, called *Q-tagged frames*, are also envelope frames. However, not all envelope frames are necessarily Q-tagged frames.

Following the fields discussed so far is the data area or *payload* portion of the frame. This is the area where higher-layer PDUs such as IP datagrams are placed. Traditionally, the payload area for Ethernet has always been 1500 bytes, representing the MTU for Ethernet. Most systems today use the 1500-byte MTU size for Ethernet, although it is generally possible to configure a smaller value if this is desired. The payload sometimes is *padded* (appended) with 0 bytes to ensure that the overall frame meets the minimum length requirements we discuss in Section 3.2.2.2.

3.2.2.1 Frame Check Sequence/Cyclic Redundancy Check (CRC)

The final field of the Ethernet frame format follows the payload area and provides an integrity check on the frame. The *Cyclic Redundancy Check* (*CRC*) field at the end includes 32 bits and is sometimes known as the IEEE/ANSI standard CRC32 [802.3-2008]. To use an *n*-bit CRC for detection of data transmission in error, the message to be checked is first appended with *n* 0 bits, forming the *augmented message*. Then, the augmented message is divided (using modulo-2 division) by an (*n* + 1)-bit value called the *generator polynomial*, which acts as the divisor. The value placed in the *CRC* field of the message is the one's complement of the remainder of this division (the quotient is discarded). Generator polynomials are standardized

for a number of different values of n. For Ethernet, which uses $n = 32$, the CRC32 generator polynomial is the 33-bit binary number 100000100110000010001110110 110111. To get a feeling for how the remainder is computed using long (mod-2) binary division, we can examine a simpler case using CRC4. The ITU has standardized the value 10011 for the CRC4 generator polynomial in a standard called G.704 [G704]. If we wish to send the 16-bit message 1001111000101111, we first begin with the long (mod-2) binary division shown in Figure 3-4.

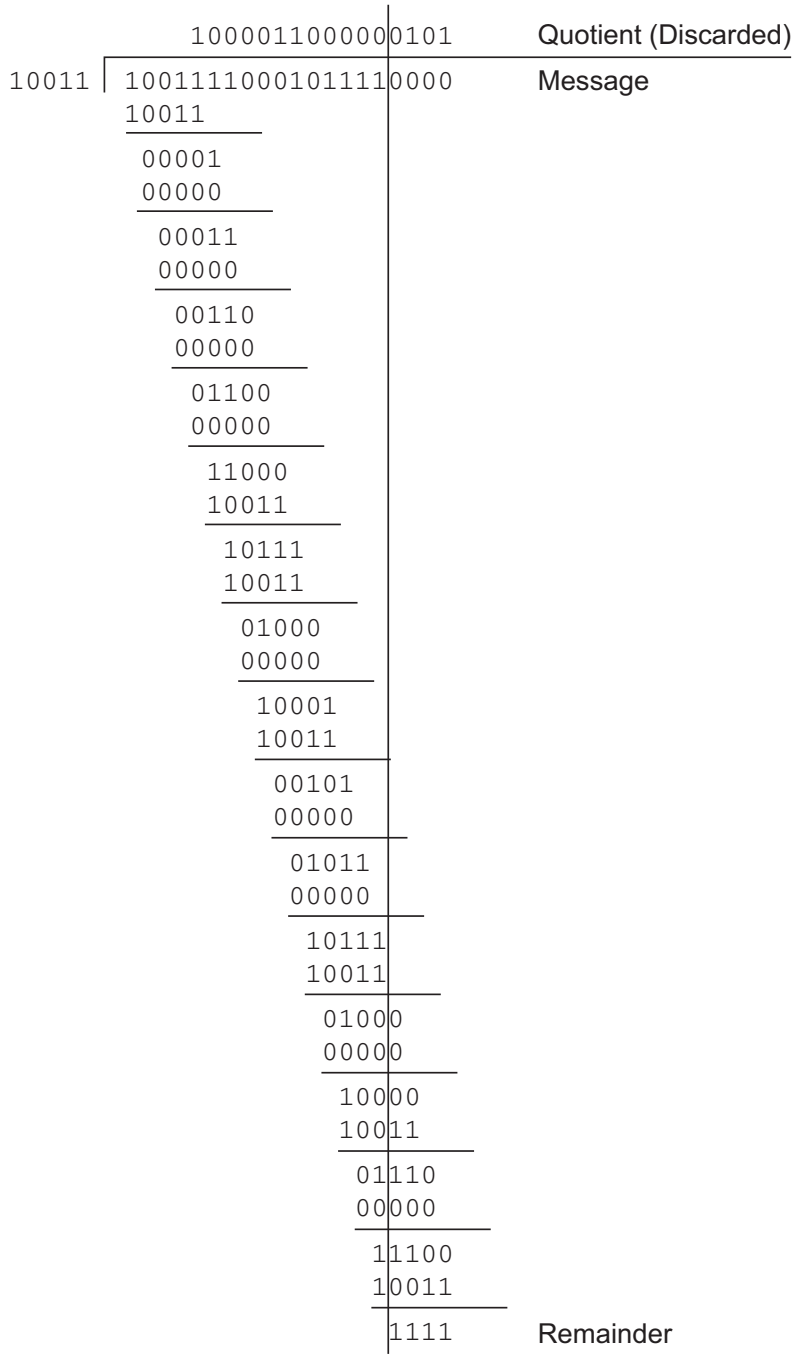

Figure 3-4 Long (mod-2) binary division demonstrating the computation of a CRC4

In this figure, we see that the remainder after division is the 4-bit value 1111. Ordinarily, the one's complement of this value (0000) would be placed in a *CRC* or *Frame Check Sequence* (*FCS*) field in the frame. Upon receipt, the receiver performs the same division and checks whether the value in the *FCS* field matches the computed remainder. If the two do not match, the frame was likely damaged in transit and is usually discarded. The CRC family of functions can be used to provide a strong indicator of corrupted messages because any change in the bit pattern is highly likely to cause a change in the remainder term.

3.2.2.2 Frame Sizes

There is both a minimum and a maximum size of Ethernet frames. The minimum is 64 bytes, requiring a minimum data area (payload) length of 48 bytes (no tags). In cases where the payload is smaller, pad bytes (value 0) are appended to the end of the payload portion to ensure that the minimum length is enforced.

Note

The minimum was important for the original 10Mb/s Ethernet using CSMA/CD. In order for a transmitting station to know which frame encountered a collision, a limit of 2500m (five 500m cable segments with four repeaters) was placed upon the length of an Ethernet network. Given that the propagation rate for electrons in copper is about .77c or 231M m/s, and given the transmission time of 64 bytes to be (64 * 8/10,000,000) = 51.2μs at 10Mb/s, a minimum-size frame could consume about 11,000m of cable. With a maximum of 2500m of cable, the maximum round-trip distance from one station to another is 5000m. The designers of Ethernet included a factor of 2 overdesign in fixing the minimum frame size, so in all compliant cases (and many noncompliant cases), the last bit of an outgoing frame would still be in the process of being transmitted after the time required for its signal to arrive at a maximally distant receiver and return. If a collision is detected, the transmitting station thus knows with certainty which frame collided—the one it is currently transmitting. In this case, the station sends a *jamming signal* (high voltage) to alert other stations, which then initiate a random binary exponential backoff procedure.

The maximum frame size of conventional Ethernet is 1518 bytes (including the 4-byte CRC and 14-byte header). This value represents a sort of trade-off: if a frame contains an error (detected on receipt by an incorrect CRC), only 1.5KB need to be retransmitted to repair the problem. On the other hand, the size limits the MTU to not more than 1500 bytes. In order to send a larger message, multiple frames are required (e.g., 64KB, a common larger size used with TCP/IP networks, would require at least 44 frames).

The unfortunate consequence of requiring multiple Ethernet frames to hold a larger upper-layer PDU is that each frame contributes a fixed overhead (14 bytes header, 4 bytes CRC). To make matters worse, Ethernet frames cannot be squished together on the network without any space between them, in order to allow the

Ethernet hardware receiver circuits to properly recover data from the network and to provide the opportunity for other stations to interleave their traffic with the existing Ethernet traffic. The Ethernet II specification, in addition to specifying a 7-byte preamble and 1-byte SFD that precedes any Ethernet frame, also specifies an *inter-packet gap* (IPG) of 12 byte times (9.6μs at 10Mb/s, 960ns at 100Mb/s, 96ns at 1000Mb/s, and 9.6ns at 10,000Mb/s). Thus, the per-frame efficiency for Ethernet II is at most 1500/(12 + 8 + 14 + 1500 + 4) = 0.975293, or about 98%. One way to improve efficiency when moving large amounts of data across an Ethernet would be to make the frame size larger. This has been accomplished using Ethernet *jumbo frames* [JF], a nonstandard extension to Ethernet (in 1000Mb/s Ethernet switches primarily) that typically allows the frame size to be as large as 9000 bytes. Some environments make use of so-called *super jumbo frames*, which are usually understood to carry more than 9000 bytes. Care should be taken when using jumbo frames, as these larger frames are not interoperable with the smaller 1518-byte frame size used by most legacy Ethernet equipment.

3.2.3 802.1p/q: Virtual LANs and QoS Tagging

With the growing use of switched Ethernet, it has become possible to interconnect every computer at a site on the same Ethernet LAN. The advantage of doing this is that any host can directly communicate with any other host, using IP and other network-layer protocols, and requiring little or no administrator configuration. In addition, broadcast and multicast traffic (see Chapter 9) is distributed to all hosts that may wish to receive it without having to set up special multicast routing protocols. While these represent some of the advantages of placing many stations on the same Ethernet, having broadcast traffic go to every computer can create an undesirable amount of network traffic when many hosts use broadcast, and there may be some security reasons to disallow complete any-to-any station communication.

To address some of these problems with running large, multiuse switched networks, IEEE extended the 802 LAN standards with a capability called *virtual LANs* (VLANs) in a standard known as 802.1q [802.1Q-2005]. Compliant Ethernet switches isolate traffic among hosts to common VLANs. Note that because of this isolation, two hosts attached to the same switch but operating on different VLANs require a router between them for traffic to flow. Combination switch/router devices have been created to address this need, and ultimately the performance of routers has been improved to match the performance of VLAN switching. Thus, the appeal of VLANs has diminished somewhat, in favor of modern high-performance routers. Nonetheless, they are still used, remain popular in some environments, and are important to understand.

Several methods are used to specify the station-to-VLAN mapping. Assigning VLANs by port is a simple and common method, whereby the switch port to which the station is attached is assigned a particular VLAN, so any station so attached becomes a member of the associated VLAN. Other options include MAC-address-based VLANs that use tables within Ethernet switches to map a station's

MAC address to a corresponding VLAN. This can become difficult to manage if stations change their MAC addresses (which they do sometimes, thanks to the behavior of some users). IP addresses can also be used as a basis for assigning VLANs.

When stations in different VLANs are attached to the same switch, the switch ensures that traffic does not leak from one VLAN to another, irrespective of the types of Ethernet interfaces being used by the stations. When multiple VLANs must span multiple switches (*trunking*), it becomes necessary to label Ethernet frames with the VLAN to which they belong before they are sent to another switch. Support for this capability uses a tag called the *VLAN tag* (or header), which holds 12 bits of *VLAN identifier* (providing for 4096 VLANs, although VLAN 0 and VLAN 4095 are reserved). It also contains 3 bits of priority for supporting QoS, defined in the 802.1p standard, as indicated in Figure 3-3. In many cases, the administrator must configure the ports of the switch to be used to send 802.1p/q frames by enabling trunking on the appropriate ports. To make this job somewhat easier, some switches support a *native VLAN* option on trunked ports, meaning that untagged frames are by default associated with the native VLAN. Trunking ports are used to interconnect VLAN-capable switches, and other ports are typically used to attach stations. Some switches also support proprietary methods for VLAN trunking (e.g., the Cisco *Inter-Switch Link* (ISL) protocol).

802.1p specifies a mechanism to express a QoS identifier on each frame. The 802.1p header includes a 3-bit-wide *Priority* field indicating a QoS level. This standard is an extension of the 802.1q VLAN standard. The two standards work together and share bits in the same header. With the 3 available bits, eight classes of service are defined. Class 0, the lowest priority, is for conventional, best-effort traffic. Class 7 is the highest priority and might be used for critical routing or network management functions. The standards specify how priorities are encoded in packets but leave the policy that governs which packets should receive which class, and the underlying mechanisms implementing prioritized services, to be defined by the implementer. Thus, the way traffic of one priority class is handled relative to another is implementation- or vendor-defined. Note that 802.1p can be used independently of VLANs if the *VLAN ID* field in the 802.1p/q header is set to 0.

The Linux command for manipulating 802.1p/q information is called `vconfig`. It can be used to add and remove virtual interfaces associating VLAN IDs to physical interfaces. It can also be used to set 802.1p priorities, change the way virtual interfaces are identified, and influence the mapping between packets tagged with certain VLAN IDs and how they are prioritized during protocol processing in the operating system. The following commands add a virtual interface to interface `eth1` with VLAN ID 2, remove it, change the way such virtual interfaces are named, and add a new interface:

```
Linux# vconfig add eth1 2
Added VLAN with VID == 2 to IF -:eth1:-
Linux# ifconfig eth1.2
```

```
eth1.2 Link encap:Ethernet HWaddr 00:04:5A:9F:9E:80
           BROADCAST MULTICAST MTU:1500 Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
Linux# vconfig rem eth1.2
Removed VLAN -:eth1.2:-
Linux# vconfig set_name_type VLAN_PLUS_VID
Set name-type for VLAN subsystem. Should be visible in
           /proc/net/vlan/config
Linux# vconfig add eth1 2
Added VLAN with VID == 2 to IF -:eth1:-
Linux# ifconfig vlan0002
vlan0002 Link encap:Ethernet HWaddr 00:04:5A:9F:9E:80
           BROADCAST MULTICAST MTU:1500 Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Here we can see that the default method of naming virtual interfaces in Linux is based on concatenating the associated physical interface with the VLAN ID. For example, VLAN ID 2 associated with the interface `eth1` is called `eth1.2`. This example also shows how an alternative naming method can be used, whereby the VLANs are enumerated by the names `vlan<n>` where *<n>* is the identifier of the VLAN. Once this is set up, frames sent on the VLAN device are tagged with the VLAN ID, as expected. We can see this using Wireshark, as shown in Figure 3-5.

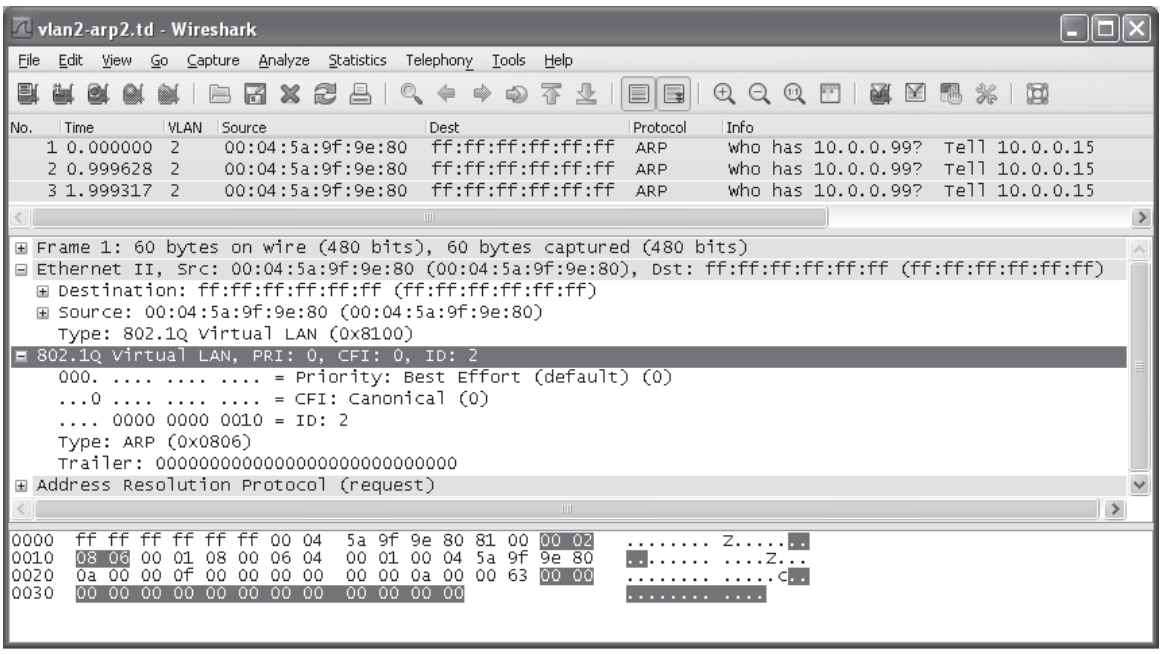

Figure 3-5 Frames tagged with the VLAN ID as shown in Wireshark. The default columns and settings have been changed to display the VLAN ID and raw Ethernet addresses.

This figure shows an ARP packet (see Chapter 4) carried on VLAN 2. We can see that the frame size is 60 bytes (not including CRC). The frame is encapsulated using the Ethernet II encapsulation with type 0x8100, indicating a VLAN. Other than the VLAN header, which indicates that this frame belongs to VLAN 2 and has priority 0, this frame is unremarkable. All the other fields are as we would expect with a regular ARP packet.

3.2.4 802.1AX: Link Aggregation (Formerly 802.3ad)

Some systems equipped with multiple network interfaces are capable of *bonding* or *link aggregation*. With link aggregation, two or more interfaces are treated as one in order to achieve greater reliability through redundancy or greater performance by splitting (striping) data across multiple interfaces. The IEEE Amendment 802.1AX [802.1AX-2008] defines the most common method for performing link aggregation and the *Link Aggregation Control Protocol* (LACP) to manage such links. LACP uses IEEE 802 frames of a particular format (called LACPDUs).

Using link aggregation on Ethernet switches that support it can be a cost-effective alternative to investing in switches with high-speed network ports. If more than one port can be aggregated to provide adequate bandwidth, higher-speed ports may not be required. Link aggregation may be supported not only on network switches but across multiple *network interface cards* (NICs) on a host computer. Often, aggregated ports must be of the same type, operating in the same mode (i.e., half- or full-duplex).

Linux has the capability to implement link aggregation (bonding) across different types of devices using the following commands:

```
Linux# modprobe bonding
Linux# ifconfig bond0 10.0.0.111 netmask 255.255.255.128
Linux# ifenslave bond0 eth0 wlan0
```

This set of commands first loads the bonding driver, which is a special type of device driver supporting link aggregation. The second command creates the `bond0` interface with the IPv4 address information provided. Although providing the IP-related information is not critical for creating an aggregated interface, it is typical. Once the `ifenslave` command executes, the bonding device, `bond0`, is labeled with the `MASTER` flag, and the `eth0` and `wlan0` devices are labeled with the `SLAVE` flag:

```
bond0 Link encap:Ethernet HWaddr 00:11:A3:00:2C:2A
          inet addr:10.0.0.111 Bcast:10.0.0.127 Mask:255.255.255.128
          inet6 addr: fe80::211:a3ff:fe00:2c2a/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1
          RX packets:2146 errors:0 dropped:0 overruns:0 frame:0
          TX packets:985 errors:0 dropped:0 overruns:0 carrier:0
          collisions:18 txqueuelen:0
          RX bytes:281939 (275.3 KiB) TX bytes:141391 (138.0 KiB)
```

```
eth0 Link encap:Ethernet HWaddr 00:11:A3:00:2C:2A
          UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
          RX packets:1882 errors:0 dropped:0 overruns:0 frame:0
          TX packets:961 errors:0 dropped:0 overruns:0 carrier:0
          collisions:18 txqueuelen:1000
          RX bytes:244231 (238.5 KiB) TX bytes:136561 (133.3 KiB)
          Interrupt:20 Base address:0x6c00
wlan0 Link encap:Ethernet HWaddr 00:11:A3:00:2C:2A
          UP BROADCAST SLAVE MULTICAST MTU:1500 Metric:1
          RX packets:269 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:38579 (37.6 KiB) TX bytes:4830 (4.7 KiB)
```

In this example, we have bonded together a wired Ethernet interface with a Wi-Fi interface. The master device, `bond0`, is assigned the IPv4 address information we would typically assign to either of the individual interfaces, and it receives the first slave's MAC address by default. When IPv4 traffic is sent out of the `bond0` virtual interface, there are a number of possibilities as to which of the slave interfaces will carry it. In Linux, the options are selected using arguments provided when the bonding driver is loaded. For example, a mode option determines whether round-robin delivery is used between the interfaces, one interface acts as a backup to the other, the interface is selected based on performing an XOR of the MAC source and destination addresses, frames are copied to all interfaces, 802.3ad standard link aggregation is performed, or more advance load-balancing options are used. The second mode is used for high-availability systems that can fail over to a redundant network infrastructure if one link has ceased functioning (detectable by MII monitoring; see [BOND] for more details). The third mode is intended to choose the slave interface based on the traffic flow. With enough different destinations, traffic between the two stations is pinned to one interface. This can be useful when trying to minimize reordering while also trying to load-balance traffic across multiple slave interfaces. The fourth mode is for fault tolerance. The fifth mode is for use with 802.3ad-capable switches, to enable dynamic aggregation over homogeneous links.

The LACP protocol is designed to make the job of setting up link aggregation simpler by avoiding manual configuration. Typically the LACP "actor" (client) and "partner" (server) send LACPDUs every second once enabled. LACP automatically determines which member links can be aggregated into a *link aggregation group* (LAG) and aggregates them. This is accomplished by sending a collection of information (MAC address, port priority, port number, and key) across the link. A receiving station can compare the values it sees from other ports and perform the aggregation if they match. Details of LACP are covered in [802.1AX-2008].

3.3 Full Duplex, Power Save, Autonegotiation, and 802.1X Flow Control

When Ethernet was first developed, it operated only in half-duplex mode using a shared cable. That is, data could be sent only one way at one time, so only one station was sending a frame at any given point in time. With the development of switched Ethernet, the network was no longer a single piece of shared wire, but instead many sets of links. As a result, multiple pairs of stations could exchange data simultaneously. In addition, Ethernet was modified to operate in full duplex, effectively disabling the collision detection circuitry. This also allowed the physical length of the Ethernet to be extended, because the timing constraints associated with half-duplex operation and collision detection were removed.

In Linux, the `ethtool` program can be used to query whether full duplex is supported and whether it is being used. This tool can also display and set many other interesting properties of an Ethernet interface:

```
Linux# ethtool eth0
Settings for eth0:
          Supported ports: [ TP MII ]
          Supported link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          Supports auto-negotiation: Yes
          Advertised link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          Advertised auto-negotiation: Yes
          Speed: 10Mb/s
          Duplex: Half
          Port: MII
          PHYAD: 24
          Transceiver: internal
          Auto-negotiation: on
          Current message level: 0x00000001 (1)
          Link detected: yes
Linux# ethtool eth1
Settings for eth1:
          Supported ports: [ TP ]
          Supported link modes: 10baseT/Half 10baseT/Full
                    100baseT/Half 100baseT/Full
                    1000baseT/Full
          Supports auto-negotiation: Yes
          Advertised link modes: 10baseT/Half 10baseT/Full
                    100baseT/Half 100baseT/Full
                    1000baseT/Full
          Advertised auto-negotiation: Yes
          Speed: 100Mb/s
          Duplex: Full
          Port: Twisted Pair
          PHYAD: 0
          Transceiver: internal
          Auto-negotiation: on
```

```
Supports Wake-on: umbg
Wake-on: g
Current message level: 0x00000007 (7)
Link detected: yes
```

In this example, the first Ethernet interface (`eth0`) is attached to a half-duplex 10Mb/s network. We can see that it is capable of *autonegotiation*, which is a mechanism originating with 802.3u to enable interfaces to exchange information such as speed and capabilities such as half- or full-duplex operation. Autonegotiation information is exchanged at the physical layer using signals sent when data is not being transmitted or received. We can see that the second Ethernet interface (`eth1`) also supports autonegotiation and has set its rate to 100Mb/s and operation mode to full duplex. The other values (`Port`, `PHYAD`, `Transceiver`) identify the physical port type, its address, and whether the physical-layer circuitry is internal or external to the NIC. The current message-level value is used to configure log messages associated with operating modes of the interface; its behavior is specific to the driver being used. We discuss the wake-on values after the following example.

In Windows, details such as these are available by navigating to Control Panel | Network Connections and then right-clicking on the interface of interest, selecting Properties, and then clicking the Configure box and selecting the Advanced tab. This brings up a menu similar to the one shown in Figure 3-6 (this particular example is from an Ethernet interface on a Windows 7 machine).

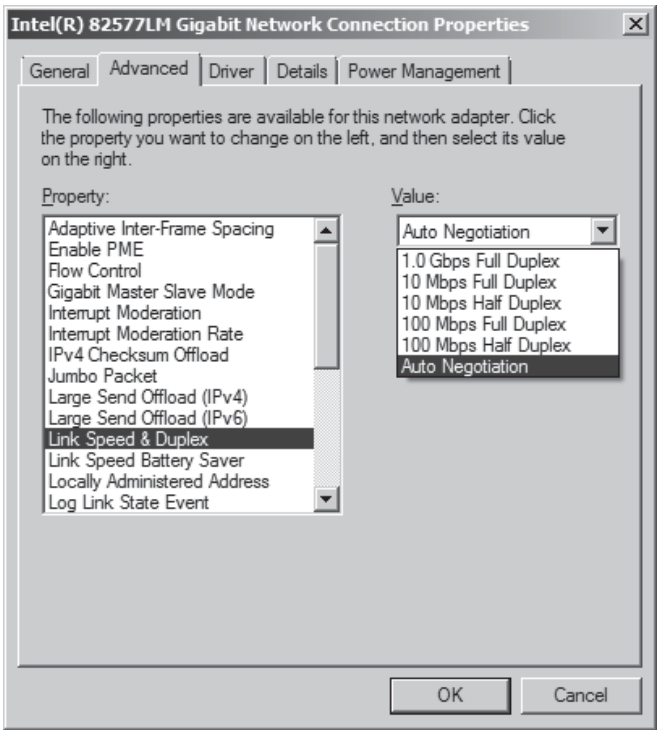

Figure 3-6 Advanced tab of network interface properties in Windows (7). This control allows the user to supply operating parameters to the network device driver.

In Figure 3-6, we can see the special features that can be configured using the adapter's device driver. For this particular adapter and driver, 802.1p/q tags can be enabled or disabled, as can flow control and wake-up capabilities (see Section 3.3.2). The speed and duplex can be set by hand, or to the more typical autonegotiation option.

3.3.1 Duplex Mismatch

Historically, there have been some interoperability problems using autonegotiation, especially when a computer and its associated switch port are configured using different duplex configurations or when autonegotiation is disabled at one end of the link but not the other. In this case, a so-called *duplex mismatch* can occur. Perhaps surprisingly, when this happens the connection does not completely fail but instead may suffer significant performance degradation. When the network has moderate to heavy traffic in both directions (e.g., during a large data transfer), a half-duplex interface can detect incoming traffic as a collision, triggering the exponential backoff function of the CSMA/CD Ethernet MAC. At the same time, the data triggering the collision is lost and may require higher-layer protocols such as TCP to retransmit. Thus, the performance degradation may be noticed only when there is sufficient traffic for the half-duplex interface to be receiving data at the same time it is sending, a situation that does not generally occur under light load. Some researchers have attempted to build analysis tools to detect this unfortunate situation [SC05].

3.3.2 Wake-on LAN (WoL), Power Saving, and Magic Packets

In both the Linux and Windows examples, we saw some indication of power management capabilities. In Windows the *Wake-Up Capabilities* and in Linux the *Wake-On* options are used to bring the network interface and/or host computer out of a lower-power (sleep) state based on the arrival of certain kinds of packets. The kinds of packets used to trigger the change to full-power state can be configured. In Linux, the Wake-On values are zero or more bits indicating whether receiving the following types of frames trigger a wake-up from a low-power state: any physical-layer (PHY) activity (`p`), unicast frames destined for the station (`u`), multicast frames (`m`), broadcast frames (`b`), ARP frames (`a`), magic packet frames (`g`), and magic packet frames including a password. These can be configured using options to `ethtool`. For example, the following command can be used:

```
Linux# ethtool -s eth0 wol umgb
```

This command configures the `eth0` device to signal a wake-up if any of the frames corresponding to the types `u`, `m`, `g`, or `b` is received. Windows provides a similar capability, but the standard user interface allows only magic packet frames and a predefined subset of the `u`, `m`, `b`, and `a` frame types. Magic packets contain

a special repeated pattern of the byte value 0xFF. Often, such frames are sent as a form of UDP packet (see Chapter 10) encapsulated in a broadcast Ethernet frame. Several tools are available to generate them, including `wol` [WOL]:

```
Linux# wol 00:08:74:93:C8:3C
Waking up 00:08:74:93:C8:3C...
```

The result of this command is to construct a magic packet, which we can view using Wireshark (see Figure 3-7).

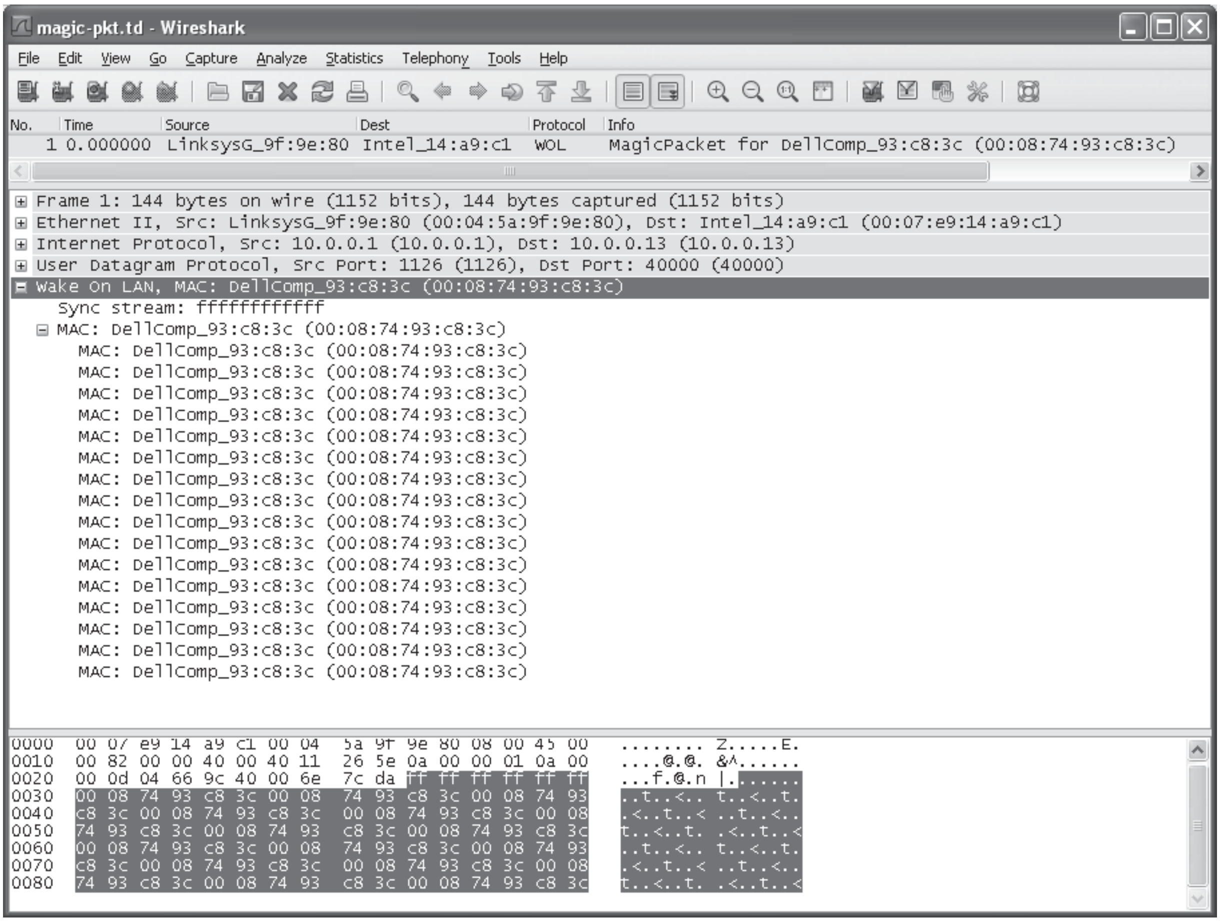

Figure 3-7 A magic packet frame in Wireshark begins with 6 0xFF bytes and then repeats the MAC address 16 times.

The packet shown in Figure 3-7 is mostly a conventional UDP packet, although the port numbers (1126 and 40000) are arbitrary. The most unusual part of the packet is the data area. It contains an initial 6 bytes with the value 0xFF. The rest of the data area includes the destination MAC address 00:08:74:93:C8:3C repeated 16 times. This data payload pattern defines the magic packet.

3.3.3 Link-Layer Flow Control

Operating an extended Ethernet LAN in full-duplex mode and across segments of different speeds may require the switches to buffer (store) frames for some period of time. This happens, for example, when multiple stations send to the same destination (called output port contention). If the aggregate traffic rate headed for a station exceeds the station's link rate, frames start to be stored in the intermediate switches. If this situation persists for a long time, frames may be dropped.

One way to mitigate this situation is to apply *flow control* to senders (i.e., slow them down). Some Ethernet switches (and interfaces) implement flow control by sending special signal frames between switches and NICs. Flow control signals to the sender that it must slow down its transmission rate, although the specification leaves the details of this to the implementation. Ethernet uses an implementation of flow control called *PAUSE messages* (also called *PAUSE frames*), specified by 802.3x [802.3-2008].

PAUSE messages are contained in MAC control frames, identified by the Ethernet *Length/Type* field having the value 0x8808 and using the MAC control opcode of 0x0001. A receiving station seeing this is advised to slow its rate. PAUSE frames are always sent to the MAC address 01:80:C2:00:00:01 and are used only on full-duplex links. They include a *hold-off* time value (specified in *quantas* equal to 512 bit times), indicating how long the sender should pause before continuing to transmit.

The MAC control frame is a frame format using the regular encapsulation from Figure 3-3, but with a 2-byte opcode immediately following the *Length/Type* field. PAUSE frames are essentially the only type of frames that uses MAC control frames. They include a 2-byte quantity encoding the hold-off time. Implementation of the "entire" MAC control layer (basically, just 802.3x flow control) is optional.

Using Ethernet-layer flow control may have a significant negative side effect, and for this reason it is typically not used. When multiple stations are sending through a switch (see the next section) that is becoming overloaded, the switch may naturally send PAUSE frames to all hosts. Unfortunately, the utilization of the switch's memory may not be symmetric with respect to the sending hosts, so some may be penalized (flow-controlled) even though they were not responsible for much of the traffic passing through the switch.

3.4 Bridges and Switches

The IEEE 802.1d standard specifies the operation of bridges, and thus switches, which are essentially high-performance bridges. A bridge or switch is used to join multiple physical link-layer networks (e.g., a pair of physical Ethernet segments) or groups of stations. The most basic setup involves connecting two switches to form an extended LAN, as shown in Figure 3-8.

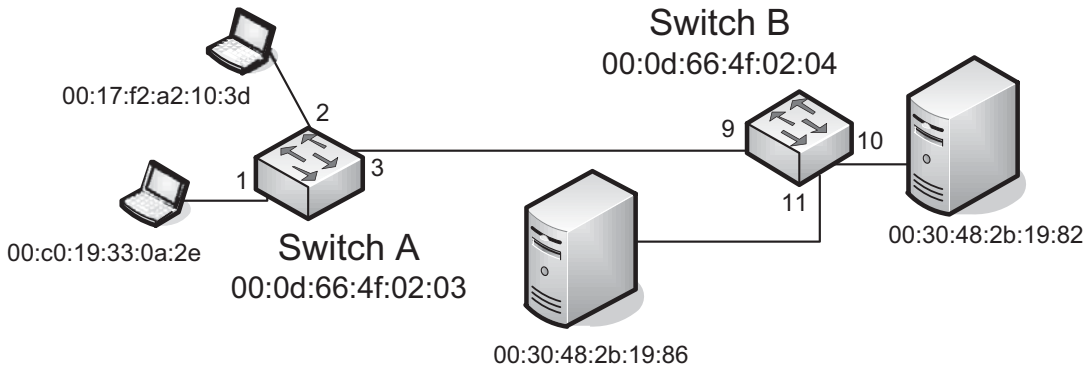

Figure 3-8 A simple extended Ethernet LAN with two switches. Each switch port has a number for reference, and each station (including each switch) has its own MAC address.

Switches A and B in the figure have been interconnected to form an extended LAN. In this particular example, client systems are connected to A and servers to B, and ports are numbered for reference. Note that every network element, including each switch, has its own MAC address. Nonlocal MAC addresses are "learned" by each bridge over time so that eventually every switch knows the port upon which every station can be reached. These lists are stored in tables (called *filtering databases*) within each switch on a per-port (and possibly per-VLAN) basis. As an example, after each switch has learned the location of every station, these databases would contain the information shown in Figure 3-9.

Station	Port
00:17:f2:a2:10:3d	2
00:c0:19:33:0a:2e	1
00:0d:66:4f:02:03	
00:0d:66:4f:02:04	3
00:30:48:2b:19:82	3
00:30:48:2b:19:86	3

Switch A's Database

Station	Port
00:17:f2:a2:10:3d	9
00:c0:19:33:0a:2e	9
00:0d:66:4f:02:03	9
00:0d:66:4f:02:04	
00:30:48:2b:19:82	10
00:30:48:2b:19:86	11

Switch B's Database

Figure 3-9 Filtering databases on switches A and B from Figure 3-8 are created over time ("learned") by observing the source address on frames seen on switch ports.

When a switch (bridge) is first turned on, its database is empty, so it does not know the location of any stations except itself. Whenever it receives a frame destined for a station other than itself, it makes a copy for each of the ports other than the one on which the frame arrived and sends a copy of the frame out of each

one. If switches (bridges) never learned the location of stations, every frame would be delivered across every network segment, leading to unwanted overhead. The learning capability reduces overhead significantly and is a standard feature of switches and bridges.

Today, most operating systems support the capability to bridge between network interfaces, meaning that a standard computer with multiple interfaces can be used as a bridge. In Windows, for example, interfaces may be bridged together by navigating to the Network Connections menu from the Control Panel, highlighting the interfaces to bridge, right-clicking the mouse, and selecting Bridge Connections. When this is done, a new icon appears that represents the bridging function itself. Most of the normal network properties associated with the interfaces are gone and instead appear on the bridge device (see Figure 3-10).

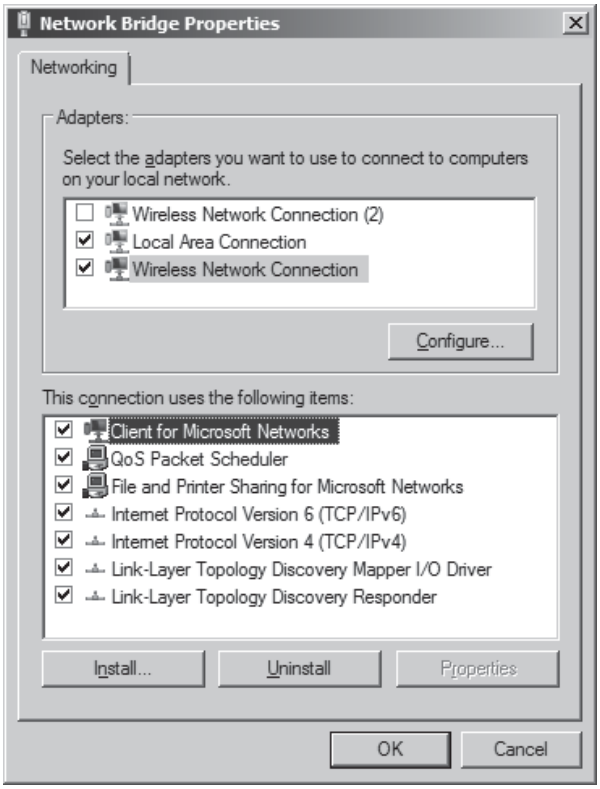

Figure 3-10 In Windows, the bridge device is created by highlighting the network interfaces to be bridged, right-clicking, and selecting the Bridge Network Interfaces function. Once the bridge is established, further modifications are made to the bridge device.

Figure 3-10 shows the Properties panels for the network bridge virtual device on Windows 7. The bridge device's properties include a list of the underlying devices being bridged and the set of services running on the bridge (e.g., the Microsoft Networks client, File and Printer Sharing, etc.). Linux works in a similar way, using command-line arguments. We use the topology shown in Figure 3-11 for this example.

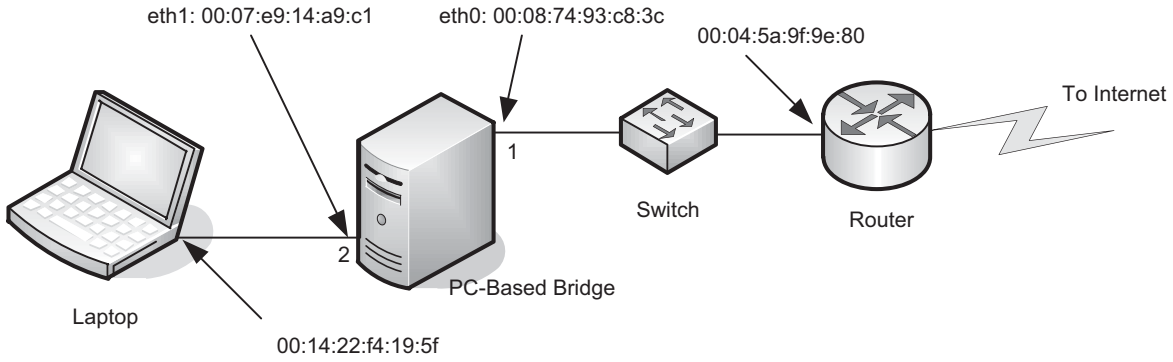

Figure 3-11 In this simple topology, a Linux-based PC is configured to operate as a bridge between the two Ethernet segments it interconnects. As a learning bridge, it accumulates tables of which port should be used to reach the various other systems on the extended LAN.

The simple network in Figure 3-11 uses a Linux-based PC with two Ethernet ports as a bridge. Attached to port 2 is a single station, and the rest of the network is attached to port 1. The following commands enable the bridge:

```
Linux# brctl addbr br0
Linux# brctl addif br0 eth0
Linux# brctl addif br0 eth1
Linux# ifconfig eth0 up
Linux# ifconfig eth1 up
Linux# ifconfig br0 up
```

This series of commands creates a bridge device `br0` and adds the interfaces `eth0` and `eth1` to the bridge. Interfaces can be removed using the `brctl delif` command. Once the interfaces are established, the `brctl showmacs` command can be used to inspect the filter databases (called *forwarding databases* or *fdb*s in Linux terminology):

```
Linux# brctl show
bridge name bridge id         STP enabled interfaces
br0         8000.0007e914a9c1 no          eth0 eth1

Linux# brctl showmacs br0
port no mac addr is local? ageing timer
  1 00:04:5a:9f:9e:80 no 0.79
  2 00:07:e9:14:a9:c1 yes 0.00
  1 00:08:74:93:c8:3c yes 0.00
  2 00:14:22:f4:19:5f no 0.81
  1 00:17:f2:e7:6d:91 no 2.53
  1 00:90:f8:00:90:b7 no 17.13
```

The output of this command reveals one other detail about bridges. Because stations may move around, have their network cards replaced, have their MAC address changed, or other things, once the bridge discovers that a MAC address

is reachable via a certain port, this information cannot be assumed to be correct forever. To deal with this issue, each time an address is learned, a timer is started (commonly defaulted to 5 minutes). In Linux, a fixed amount of time associated with the bridge is applied to each learned entry. If the address in the entry is not seen again within the specified "`ageing`" time, the entry is removed, as indicated here:

```
Linux# brctl setageing br0 1
Linux# brctl showmacs br0
port no mac addr is local? ageing timer
  1 00:04:5a:9f:9e:80 no 0.76
  2 00:07:e9:14:a9:c1 yes 0.00
  1 00:08:74:93:c8:3c yes 0.00
  2 00:14:22:f4:19:5f no 0.78
  1 00:17:f2:e7:6d:91 no 0.00
```

Here, we have set the `ageing` value unusually low for demonstration purposes. When an entry is removed because of aging, subsequent frames for the removed destination are once again sent out of every port except the receiving one (called *flooding*), and the entry is placed anew into the filtering database. The use of filtering databases and learning is really a performance optimization—if the tables are empty, the network experiences more overhead but still functions. Next we turn our attention to the case where more than two bridges are interconnected with redundant links. In this situation, flooding of frames could lead to a sort of flooding catastrophe with frames looping forever. Obviously, we require a way of dealing with this problem.

3.4.1 Spanning Tree Protocol (STP)

Bridges may operate in isolation, or in combination with other bridges. When more than two bridges are in use (or in general when switch ports are cross-connected), the possibility exists for a cascading, looping set of frames to be formed. Consider the network shown in Figure 3-12.

Assume that the switches in Figure 3-12 have just been turned on and their filtering databases are empty. When station S sends a frame, switch B replicates the frame on ports 7, 8, and 9. So far, the initial frame has been "amplified" three times. These frames are received by switches A, D, and C. Switch A produces copies of the frame on ports 2 and 3. Switches D and C produce more copies on ports 20, 22 and 13, 14, respectively. The amplification factor has grown to 6, with copies of the frames traveling in both directions among switches A, C, and D. Once these frames arrive, the forwarding databases begin to oscillate as the bridge attempts to figure out which port is really the one through which station S should be reached. Obviously, this situation is intolerable. If it were allowed to occur, bridges used in such configurations would be useless. Fortunately, there is a protocol that is used to avoid this situation called the *Spanning Tree Protocol* (STP). We describe STP in

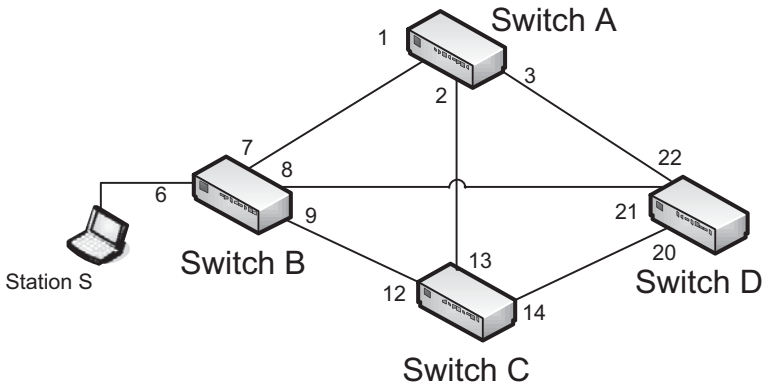

Figure 3-12 An extended Ethernet network with four switches and multiple redundant links. If simple flooding were used in forwarding frames through this network, a catastrophe would occur because of excess multiplying traffic (a so-called broadcast storm). This type of situation requires the use of the STP.

some detail to explain why some approach to duplicate suppression is needed for bridges and switches. In the current standard [802.1D-2004], conventional STP is replaced with the *Rapid Spanning Tree Protocol* (RSTP), which we describe after the conventional STP preliminaries.

STP works by disabling certain ports at each bridge so that topological loops are avoided (i.e., no duplicate paths between bridges are permitted), yet the topology is not partitioned—all stations can be reached. Mathematically, a spanning tree is a collection of all of the nodes and some of the edges of a graph such that there is a path or route from any node to any other node (*spanning* the graph), but there are no loops (the edge set forms a *tree*). There can be many spanning trees on a graph. STP finds one of them for the graph formed by bridges as nodes and links as edges. Figure 3-13 illustrates the idea.

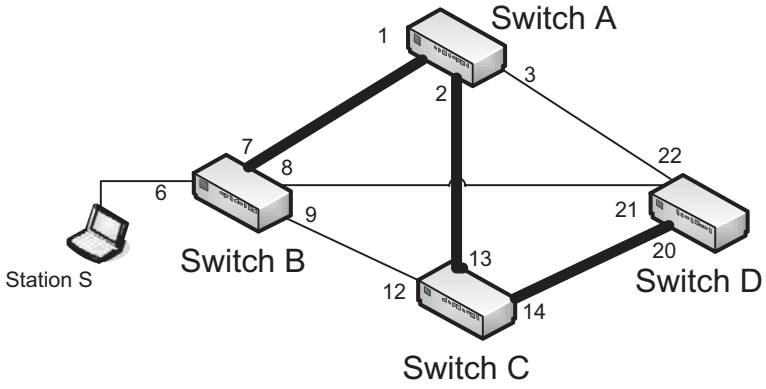

Figure 3-13 Using STP, the B-A, A-C, and C-D links have become active on the spanning tree. Ports 6, 7, 1, 2, 13, 14, and 20 are in the forwarding state; all other ports are blocked (i.e., not forwarding). This keeps frames from looping and avoids broadcast storms. If a configuration change occurs or a switch fails, the blocked ports are changed to the forwarding state and the bridges compute a new spanning tree.

In this figure, the dark lines represent the links in the network selected by STP for forwarding frames. None of the other links are used—ports 8, 9, 12, 21, 22, and 3 are *blocked*. With STP, the various problems raised earlier do not occur, as frames are created only as the result of another frame arriving. There is no amplification. Furthermore, looping is avoided because there is only one path between any two stations. The spanning tree is formed and maintained by bridges using a distributed algorithm running in each bridge.

As with forwarding databases, STP must deal with the situation where bridges are turned off and on, interface cards are replaced, or MAC addresses are changed. Clearly, such changes could affect the operation of the spanning tree, so the STP adapts to these changes. The adaptation is implemented using an exchange of special frames called *Bridge Protocol Data Units* (BPDUs). These frames are used for forming and maintaining the spanning tree. The tree is "grown" from a bridge elected by the others and known as the "root bridge."

As mentioned previously, there are many possible spanning trees for a given network. Determining which one might be the best to use for forwarding frames depends on a set of *costs* that can be associated with each link and the location of the root bridge. Costs are simply integers that are (recommended to be) inversely proportional to the link speeds. For example, a 10Mb/s link has a recommended cost of 100, and 100Mb/s and 1000Mb/s links have recommended cost values of 19 and 4, respectively. STP operates by computing least-cost paths to the root bridge using these costs. If multiple links must be traversed, the corresponding cost is simply the sum of the link costs.

3.4.1.1 Port States and Roles

To understand the basic operation of STP, we need to understand the operation of the state machine for each port at each bridge, as well as the contents of BPDUs. Each port in each bridge may be in one of five states: blocking, listening, learning, forwarding, and disabled. The relationship among them can be seen in the state transition diagram shown in Figure 3-14.

The normal transitions for ports on the spanning tree are indicated in Figure 3-14 by solid arrows, and the smaller arrows with dashed lines indicate changes due to administrative configuration. After initialization, a port enters the blocking state. In this state, it does not learn addresses, forward frames, or transmit BPDUs, but it does monitor received BPDUs in case it needs to be included in the future on a path to the root bridge, in which case the port transitions to the listening state. In the listening state, the port is now permitted to send as well as receive BPDUs but not learn addresses or forward data. After a typical forwarding delay timeout of 15s, a port enters the learning state. Here it is permitted to do all procedures except forward data. It waits another forwarding delay before entering the forwarding state and commencing to forward frames.

Related to the port state machine, each port is said to have a *role*. This terminology becomes more important with RSTP (see Section 3.4.1.6). A port may have the role of *root port*, *designated port*, *alternate port*, or *backup port*. Root ports are those

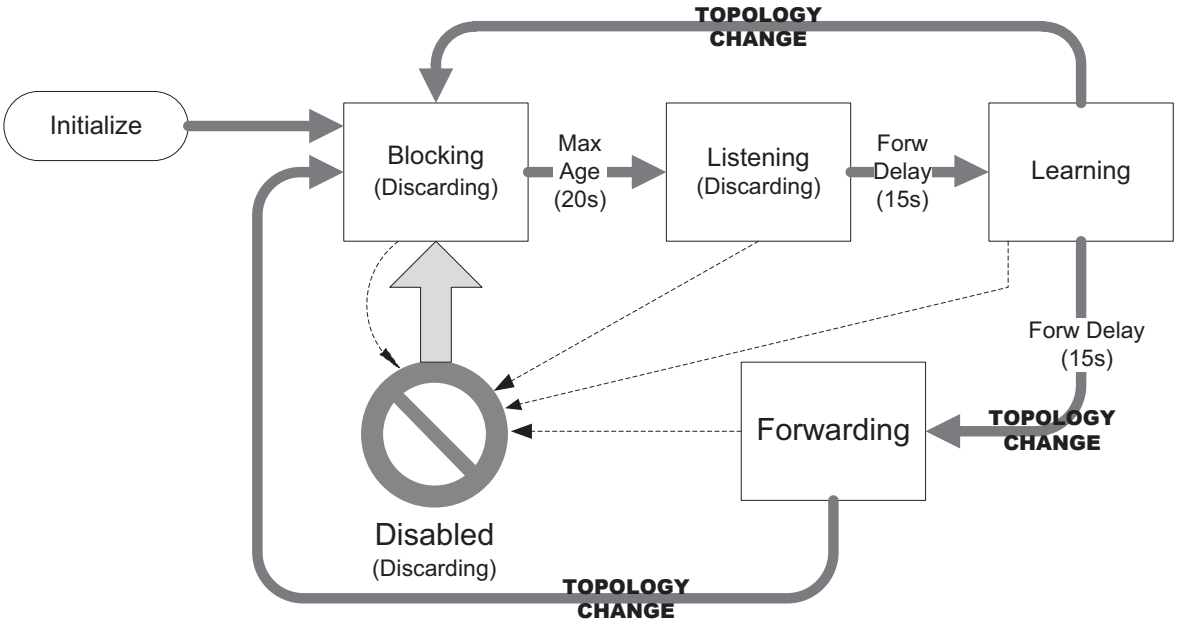

Figure 3-14 Ports transition among four major states in normal STP operation. In the blocking state, frames are not forwarded, but a topology change or timeout may cause a transition to the listening state. The forwarding state is the normal state for active switch ports carrying data traffic. The state names in parentheses indicate the port states according to the RSTP.

ports at the end of an edge on the spanning tree headed toward the root. Designated ports are ports in the forwarding state acting as the port on the least-cost path to the root from the attached segment. Alternate ports are other ports on an attached segment that could also reach the root but at higher cost. They are not in the forwarding state. A backup port is a port connected to the same segment as a designated port *on the same bridge*. Thus, backup ports could easily take over for a failing designated port without disrupting any of the rest of the spanning tree topology but do not offer an alternate path to the root should the entire bridge fail.

3.4.1.2 BPDU Structure

To determine the links in the spanning tree, STP uses BPDUs that adhere to the format shown in Figure 3-15.

The format shown in Figure 3-15 applies to both the original STP as well as the newer RSTP (see Section 3.4.1.6). BPDUs are always sent to the group address 01:80:C2:00:00:00 (see Chapter 9 for details of link-layer group and Internet multicast addressing) and are not forwarded through a bridge without modification. In the figure, the *DST*, *SRC*, and *L/T* (*Length/Type*) fields are part of the conventional Ethernet (802.3) header of the frame carrying the example BPDU. The 3-byte *LLC/SNAP* header is defined by 802.1 and for BPDUs is set to the constant 0x424203. Not all BPDUs are encapsulated using LLC/SNAP, but this is a common option.

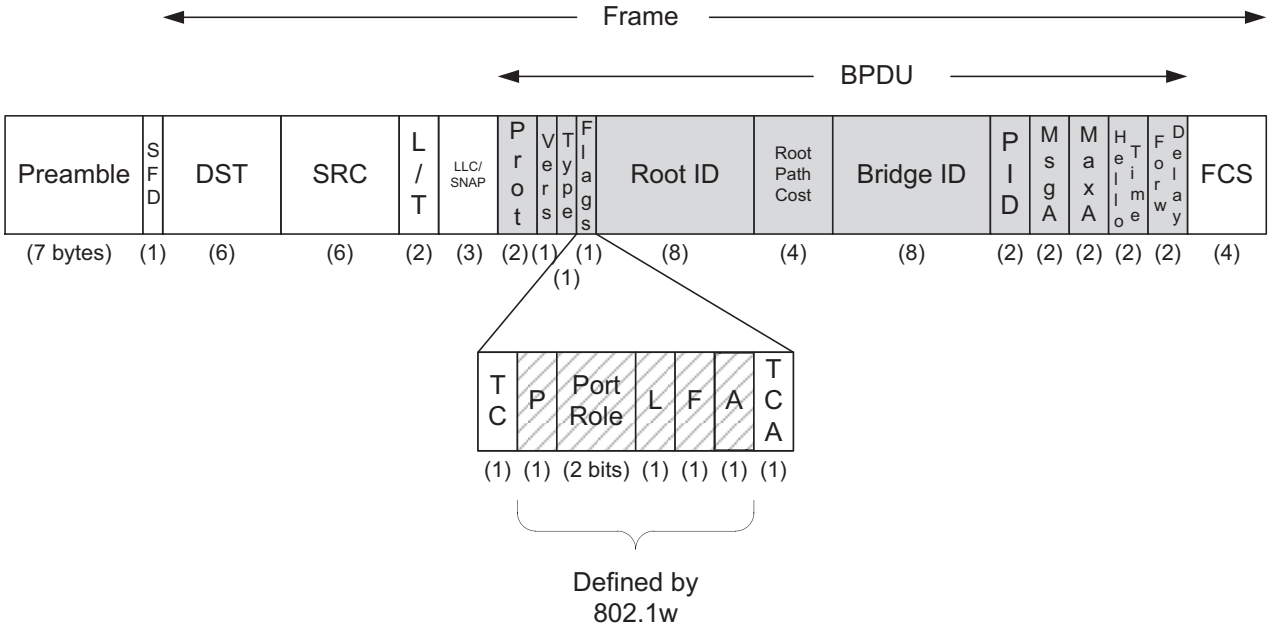

Figure 3-15 BPDUs are carried in the payload area of 802 frames and exchanged between bridges to establish the spanning tree. Important fields include the source, root node, cost to root, and topology change indication. With 802.1w and [802.1D-2004] (including Rapid STP or RSTP), additional fields indicate the state of the ports.

The *Protocol* (*Prot*) field gives the protocol ID number, set to 0. The *Version* (*Vers*) field is set to 0 or 2, depending on whether STP or RSTP is in use. The *Type* field is assigned similarly. The *Flags* field contains *Topology Change* (*TC*) and *Topology Change Acknowledgment* (*TCA*) bits, defined by the original 802.1d standard. Additional bits are defined for *Proposal* (*P*), *Port Role* (00, unknown; 01, alternate; 10, root; 11, designated), *Learning* (*L*), *Forwarding* (*F*), and *Agreement* (*A*). These are discussed in the context of RSTP in Section 3.4.1.6. The *Root ID* field gives the identifier of the root bridge in the eyes of the sender of the frame, whose MAC address is given in the *Bridge ID* field. Both of these ID fields are encoded in a special way that includes a 2-byte *Priority* field immediately preceding the MAC address. The priority values can be manipulated by management software in order to force the spanning tree to be rooted at any particular bridge (Cisco, for example, uses a default value of 0x8000 in its Catalyst switches).

The root path cost is the computed cost to reach the bridge specified in the *Root ID* field. The *PID* field is the port identifier and gives the number of the port from which the frame was sent appended to a 1-byte configurable *Priority* field (default 0x80). The *Message A* (*MsgA*) field gives the message age (see the next paragraph). The *Maximum Age* (*MaxA*) field gives the maximum age before timeout (default: 20s). The *Hello Time* field gives the time between periodic transmissions of configuration frames. The *Forward Delay* (*Forw Delay*) field gives the time spent in the learning and listening states. All of the age and time fields are given in units of 1/256s.

The *Message Age* field is not a fixed value like the other time-related fields. When the root bridge sends a BPDU, it sets this field to 0. Any bridge receiving the frame emits frames on its non-root ports with the *Message Age* field incremented by 1. In essence, the field acts as a hop count, giving the number of bridges by which the BPDU has been processed before being received. When a BPDU is received on a port, the information it contains is kept in memory and participates in the STP algorithm until it is timed out, which happens at time (*MaxA* – *MsgA*). Should this time pass on a root port without receipt of another BPDU, the root bridge is declared "dead" and the bridge starts the root bridge election process over again.

3.4.1.3 Building the Spanning Tree

The first job of STP is to elect the root bridge. The root bridge is discovered as the bridge in the network (or VLAN) with the smallest identifier (priority combined with MAC address). When a bridge initializes, it assumes itself to be the root bridge and sends configuration BPDUs with the *Root ID* field matching its own bridge ID, but if it detects a bridge with a smaller ID, it ceases sending its own frames and instead adopts the frame it received containing the smaller ID to be the basis for further BPDUs it sends. The port where the BPDU with the smaller root ID was received is then marked as the root port (i.e., the port on the path to the root bridge). The remaining ports are placed in either blocked or forwarding states.

3.4.1.4 Topology Changes

The next important job of STP is to handle topology changes. Although we could conceivably use the basic database aging mechanism described earlier to adapt to changing topologies, this is a poor approach because the aging timers can take a long time (5 minutes) to delete incorrect entries. Instead, STP incorporates a way to detect topology changes and inform the network about them quickly. In STP, a topology change occurs when a port has entered the blocking or forwarding states. When a bridge detects a connectivity change (e.g., a link goes down), the bridge notifies its parent bridges on the tree to the root by sending *topology change notification* (TCN) BPDUs out of its root port. The next bridge on the tree to the root acknowledges the TCN BPDUs to the notifying bridge and also forwards them on toward the root. Once informed of the topology change, the root bridge sets the *TC* bit field in subsequent periodic configuration messages. Such messages are relayed by every bridge in the network and are received by ports in either the blocking or forwarding states. The setting of this bit field allows bridges to reduce their aging time to that of the forward delay timer, on the order of seconds instead of the 5 minutes normally recommended for the aging time. This allows database entries that may now be incorrect to be purged and relearned more quickly, yet it also allows stations that are actively communicating to not have their entries deleted erroneously.

3.4.1.5 Example

In Linux, the bridge function disables STP by default, on the assumption that topologies are relatively simple in most cases where a regular computer is being

used as a bridge. To enable STP on the example bridge we are using so far, we can do the following:

```
Linux# brctl stp br0 on
```

The consequences of executing this command can be inspected as follows:

```
Linux# brctl showstp br0

br0

 bridge id              8000.0007e914a9c1
 designated root        8000.0007e914a9c1
 root port                 0                path cost              0
 max age                  19.99             bridge max age        19.99
 hello time                1.99             bridge hello time      1.99
 forward delay            14.99             bridge forward delay  14.99
 ageing time               0.99
 hello timer               1.26             tcn timer              0.00
 topology change timer     3.37             gc timer               3.26

 flags                    TOPOLOGY_CHANGE TOPOLOGY_CHANGE_DETECTED

eth0 (0)
 port id                0000                     state         forwarding
 designated root        8000.0007e914a9c1        path cost          100
 designated bridge      8000.0007e914a9c1        message age timer   0.00
 designated port        8001                     forward delay timer 0.00

 designated cost        0                        hold timer          0.26

 flags

eth1 (0)
 port id                0000                     state         forwarding
 designated root        8000.0007e914a9c1        path cost           19
 designated bridge      8000.0007e914a9c1        message age timer   0.00
 designated port        8002                     forward delay timer 0.00
 designated cost        0                        hold timer          0.26

 flags
```

Here we can see the STP setup for a simple bridged network. The bridge device, `br0`, holds information for the bridge as a whole. This includes the bridge ID (`8000.0007e914a9c1`), derived from the smallest MAC address on the PC-based bridge (port 1) of Figure 3-11. The major configuration parameters (e.g., hello time, topology change timer, etc.) are given in seconds. The `flags` values indicate a recent topology change, which is expected given the fact that the network was recently connected. The rest of the output describes per-port information for `eth0`

(bridge port 1) and `eth1` (bridge port 2). Note that the path cost for `eth0` is about ten times greater than the cost of `eth1`. This is consistent with the observation that `eth0` is a 10Mb/s Ethernet network and `eth1` is a full-duplex 100Mb/s network.

We can use Wireshark to look at a BPDU. In Figure 3-16 we see the contents of a 52-byte BPDU. The length of 52 bytes (less than the Ethernet minimum of 64 bytes because the Linux capture facility removed the padding) is derived from the *Length/Type* field of the Ethernet header by adding 14, in this case giving the length of 52. The destination address is the group address, 01:80:C2:00:00:00, as expected. The payload length is 38 bytes, the value contained in the *Length* field. The *SNAP/LLC* field contains the constant 0x424243, and the encapsulated frame is a spanning tree (version 0) frame. The rest of the protocol fields indicate that the station 00:07:e9:14:a9:c1 believes it is the root of the spanning tree, using priority 32768 (a low priority), and the BPDU has been sent from port 2 with priority 0x80. It also indicates a maximum age of 20s, a hello time of 2s, and a forwarding delay of 15s.

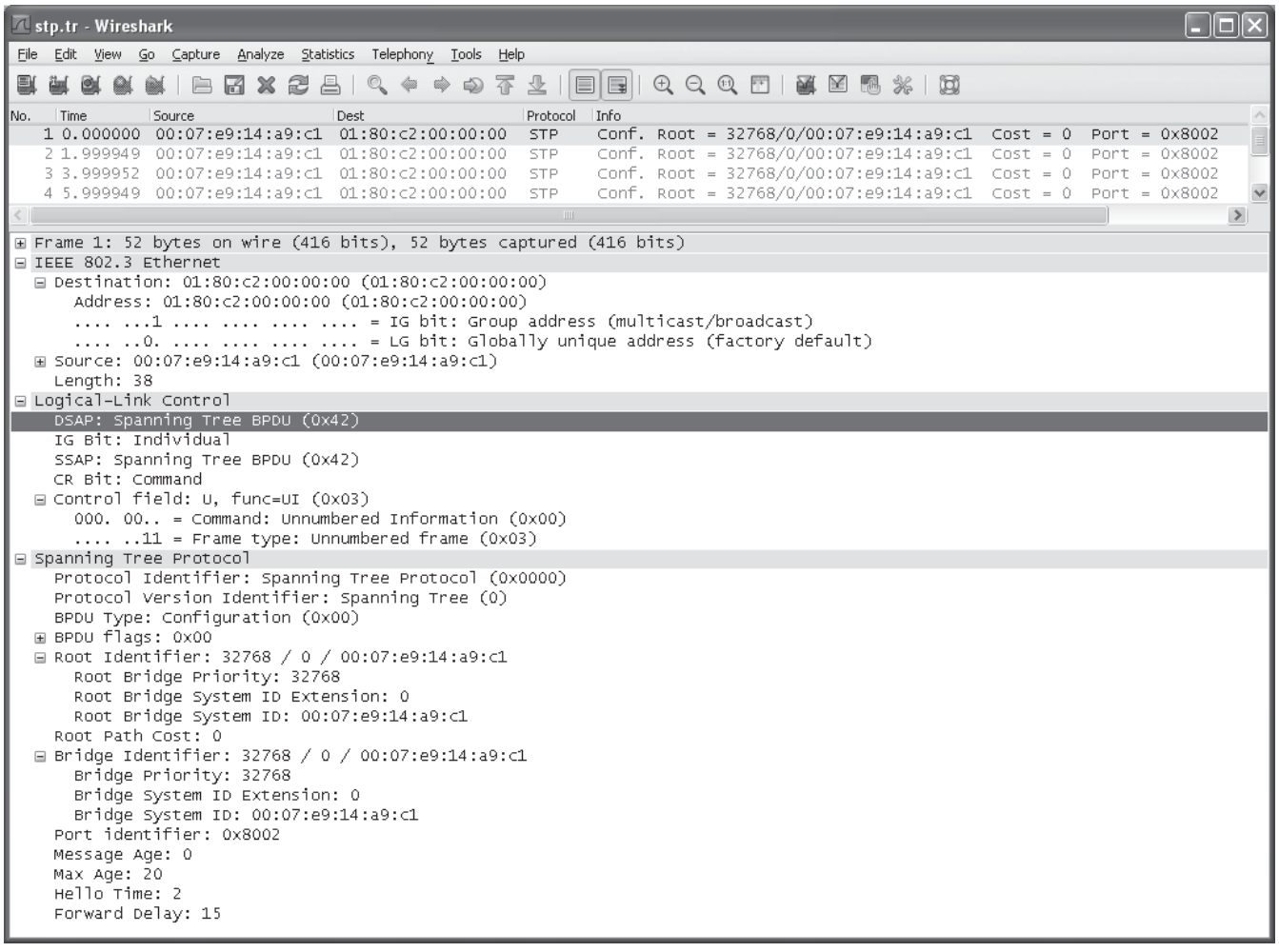

Figure 3-16 Wireshark showing a BPDU. The Ethernet destination is a group address for bridges (01:80:c2:00:00:00).

3.4.1.6 Rapid Spanning Tree Protocol (RSTP) (Formerly 802.1w)

One of the perceived problems with conventional STP is that a change in topology is detected only by the failure to receive a BPDU in a certain amount of time. If the timeout is large, the convergence time (time to reestablish data flow along the spanning tree) could be larger than desired. The IEEE 802.1w standard (now part of [802.1D-2004]) specifies enhancements to the conventional STP and adopts the new name *Rapid Spanning Tree Protocol* (RSTP). The main improvement in RSTP over STP is to monitor the status of each port and upon indication of failure to *immediately* trigger a topology change indication. In addition, RSTP uses all 6 bits in the *Flag* field of the BPDU format to support agreements between bridges that avoid some of the need for timers to initiate protocol operations. It reduces the normal STP five port states to three (discarding, learning, and forwarding, as indicated by the state names in parentheses in Figure 3-14). The discarding state in RSTP absorbs the disabled, blocking, and listening states in conventional STP. RSTP also creates a new port role called an *alternate port*, which acts as an immediate backup should a root port cease to operate.

RSTP uses only one type of BPDU, so there are no special topology change BPDUs, for example. RSTP BPDUs, as they are called, use version and type number 2 instead of 0. In RSTP, any switch detecting a topology change sends BPDUs indicating a topology change, and any switch receiving them clears its filtering databases immediately. This change can significantly affect the protocol's convergence time. Instead of waiting for the topology change to migrate to the root bridge and back followed by the forwarding delay wait time, entries are cleared immediately. Overall, convergence time can be cut from tens of seconds down to a fraction of a second in most cases.

RSTP makes a distinction between *edge ports* (those attached only to end stations) and normal spanning tree ports and also between point-to-point links and shared links. Edge ports and ports on point-to-point links do not ordinarily form loops, so they are permitted to skip the listening and learning states and move directly to the forwarding state. Of course, the assumption of being an edge port could be violated if, for example, two ports were cross-connected, but this is handled by reclassifying ports as spanning tree ports if they ever carry any form of BPDUs (simple end stations do not normally generate BPDUs). Point-to-point links are inferred from the operating mode of the interface; if the interface is running in full-duplex mode, the link is classified as a point-to-point link.

In regular STP, BPDUs are ordinarily relayed from a notifying or root bridge. In RSTP, BPDUs are sent periodically by all bridges as "keepalives" to determine if connections to neighbors are operating properly. This is what most higher-layer routing protocols do also. If a bridge fails to receive an updated BPDU within three times the hello interval, the bridge concludes that it has lost its connection with its neighbor. Note that in RSTP, topology changes are not induced as a result of edge ports being connected or disconnected as they are in regular STP. When a topology change is detected, the notifying bridge sends BPDUs with the *TC* bit

field set, not only to the root but also to all other bridges. Doing so allows the entire network to be notified of the topology change much faster than with conventional STP. When a bridge receives these messages, it flushes all table entries except those associated with edge ports and restarts the learning process.

Many of RSTP's features were developed by Cisco Systems and other companies that had for some time provided proprietary enhancements to regular STP in their products. The IEEE committee incorporated many of these enhancements into the updated 802.1d standard, which covers both types of STP, so extended LANs can run regular STP on some segments and RSTP on others (although the RSTP benefits are lost). RSTP has been extended to include VLANs [802.1Q-2005]—a protocol called the *Multiple Spanning Tree Protocol* (MSTP). This protocol retains the RSTP (and hence STP) BPDU format, so backward compatibility is possible, but it also supports the formation of multiple spanning trees (one for each VLAN).

3.4.2 802.1ak: Multiple Registration Protocol (MRP)

The *Multiple Registration Protocol* (MRP) provides a general method for registering attributes among stations in a bridged LAN environment. [802.1ak-2007] defines two particular "applications" of MRP called MVRP (for registering VLANs) and MMRP (for registering group MAC addresses). MRP replaces the earlier GARP framework; MVRP and MMRP replace the older GVRP and GMRP protocols, respectively. All were originally defined by 802.1q.

With MVRP, once an end station is configured as a member of a VLAN, this information is communicated to its attached switch, which in turn propagates the fact of the station's participation in the VLAN to other switches. This allows switches to augment their filtering tables based on station VLAN IDs and allows changes of VLAN topology without necessarily triggering a recalculation of the existing spanning tree via STP. Avoiding STP recalculation was one of the reasons for migrating from GVRP to MVRP.

MMRP is a method for stations to register their interest in group MAC addresses (multicast addresses). This information may be used by switches to establish the ports through which multicast traffic must be delivered. Without such a facility, switches would have to broadcast all multicast traffic, potentially leading to unwanted overhead. MMRP is a layer 2 protocol with similarities to IGMP and MLD, layer 3 protocols, and the "IGMP/MLD snooping" capability supported in many switches. We discuss IGMP, MLD and snooping in Chapter 9.

3.5 Wireless LANs—IEEE 802.11(Wi-Fi)

One of the most popular technologies being used to access the Internet today is *wireless fidelity* (Wi-Fi), also known by its IEEE standard name 802.11, effectively a wireless version of Ethernet. Wi-Fi has developed to become an inexpensive, highly convenient way to provide connectivity and performance levels acceptable

for most applications. Wi-Fi networks are easy to set up, and most portable computers and smartphones now include the necessary hardware to access Wi-Fi infrastructure. Many coffee shops, airports, hotels, and other facilities include Wi-Fi "hot spots," and Wi-Fi is even seeing considerable advancement in developing countries where other infrastructure may be difficult to obtain. The architecture of an IEEE 802.11 network is shown in Figure 3-17.

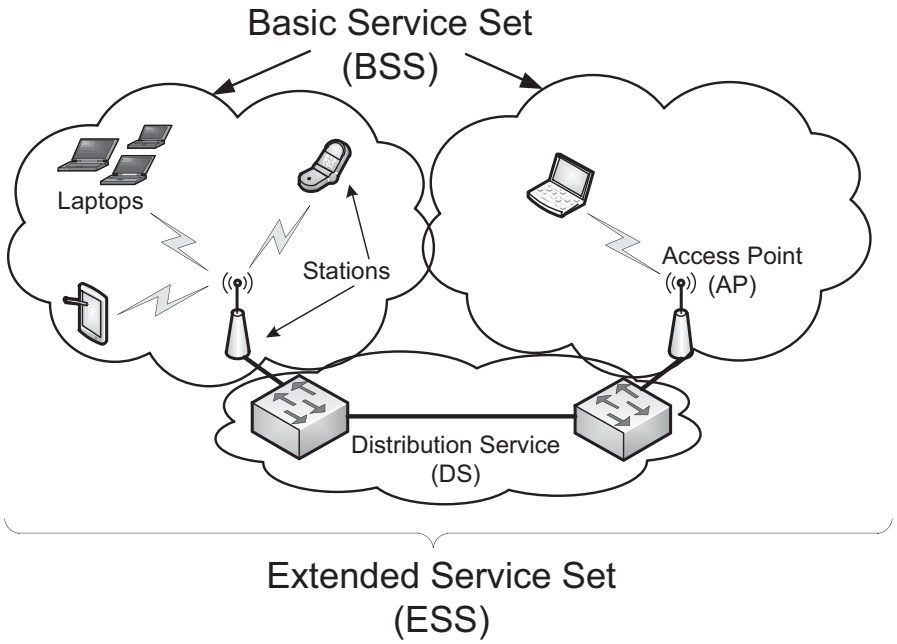

Figure 3-17 The IEEE 802.11 terminology for a wireless LAN. Access points (APs) can be connected using a distribution service (DS, a wireless or wired backbone) to form an extended WLAN (called an ESS). Stations include both APs and mobile devices communicating together that form a basic service set (BSS). Typically, an ESS has an assigned ESSID that functions as a name for the network.

The network in Figure 3-17 includes a number of *stations* (STAs). Typically stations are organized with a subset operating also as *access points* (APs). An AP and its associated stations are called a *basic service set* (BSS). The APs are generally connected to each other using a wired *distribution service* (called a DS, basically a "backbone"), forming an *extended service set* (ESS). This setup is commonly termed *infrastructure mode*. The 802.11 standard also provides for an *ad hoc mode*. In this configuration there is no AP or DS; instead, direct station-to-station (peer-to-peer) communication takes place. In IEEE terminology, the STAs participating in an ad hoc network form an *independent basic service set* (IBSS). A WLAN formed from a collection of BSSs and/or IBSSs is called a *service set*, identified by a *service set identifier* (SSID). An *extended service set identifier* (ESSID) is an SSID that names a collection of connected BSSs and is essentially a name for the LAN that can be up to 32 characters long. Such names are ordinarily assigned to Wi-Fi APs when a WLAN is first installed.

3.5.1 802.11 Frames

There is one common overall frame format for 802.11 networks but multiple types of frames. Not all the fields are present in every type of frame. Figure 3-18 shows the format of the common frame and a (maximal-size) data frame.

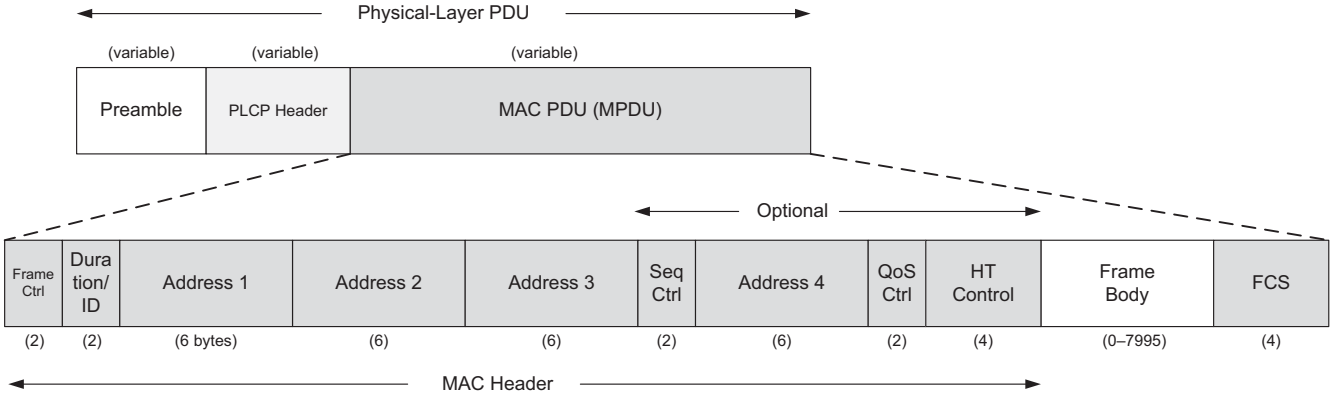

Figure 3-18 The 802.11 basic data frame format (as of [802.11n-2009]). The MPDU format resembles that of Ethernet but has additional fields depending on the type of DS being used among access points, whether the frame is headed to the DS or from it, and if frames are being aggregated. The *QoS Control* field is used for special performance features, and the *HT Control* field is used for control of 802.11n's "high-throughput" features.

The frame shown in Figure 3-18 includes a preamble for synchronization, which depends on the particular variant of 802.11 being used. Next, the *Physical Layer Convergence Procedure* (PLCP) header provides information about the specific physical layer in a somewhat PHY-independent way. The PLCP portion of the frame is generally transmitted at a lower data rate than the rest of the frame. This serves two purposes: to improve the probability of correct delivery (lower speeds tend to have better error resistance) and to provide compatibility with and protection from interference from legacy equipment that may operate in the same area at slower rates. The *MAC PDU* (MPDU) corresponds to a frame similar to Ethernet, but with some additional fields.

At the head of the MPDU is the *Frame Control Word*, which includes a 2-bit *Type* field identifying the frame type. There are three types of frames: *management frames*, *control frames*, and *data frames*. Each of these can have various subtypes, depending on the type. The full table of types and subtypes is given in [802.11n-2009, Table 7-1]. The contents of the remaining fields, if present, are determined by the frame type, which we discuss individually.

3.5.1.1 Management Frames

Management frames are used for creating, maintaining, and ending associations between stations and access points. They are also used to determine whether encryption is being used, what the name (SSID or ESSID) of the network is, what

transmission rates are supported, and a common time base. These frames are used to provide the information necessary when a Wi-Fi interface "scans" for nearby access points.

Scanning is the procedure by which a station discovers available networks and related configuration information. This involves switching to each available frequency and passively listening for traffic to identify available access points. Stations may also actively probe for networks by transmitting a particular management frame ("probe request") while scanning. There are some limitations on such probe requests to ensure that 802.11 traffic is not transmitted on a frequency that is being used for non-802.11 purposes (e.g., medical services). Here is an example of initiating a scan by hand on a Linux system:

```
Linux# iwlist wlan0 scan
wlan0 Scan completed :
            Cell 01 - Address: 00:02:6F:20:B5:84
                      ESSID:"Grizzly-5354-Aries-802.11b/g"
                      Mode:Master
                      Channel:4
                       Frequency:2.427 GHz (Channel 4)
                      Quality=5/100 Signal level=47/100
                      Encryption key:on
                      IE: WPA Version 1
                         Group Cipher : TKIP
                         Pairwise Ciphers (2) : CCMP TKIP
                         Authentication Suites (1) : PSK
                      Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s;
                             6 Mb/s; 12 Mb/s; 24 Mb/s; 36 Mb/s; 9 Mb/s;
                             18 Mb/s; 48 Mb/s; 54 Mb/s
                      Extra:tsf=0000009d832ff037
```

Here we see the result of a hand-initiated scan using wireless interface `wlan0`. An AP with MAC address `00:02:6F:20:B5:84` is acting as a master (i.e., is acting as an AP in infrastructure mode). It is broadcasting the ESSID `"Grizzly-5354-Aries-802.11b/g"` on channel 4 (2.427GHz). (See Section 3.5.4 on channels and frequencies for more details on channel selection.) The quality and signal level give indications of how well the scanning station is receiving a signal from the AP, although the meaning of these values varies among manufacturers. WPA encryption is being used on this link (see Section 3.5.5), and bit rates from 1Mb/s to 54Mb/s are available. The `tsf` (*time sync function*) value indicates the AP's notion of time, which is used for synchronizing various features such as power-saving mode (see Section 3.5.2).

When an AP broadcasts its SSID, any station may attempt to establish an association with the AP. When an association is established, most Wi-Fi networks today also set up the necessary configuration information to provide Internet access to the station (see Chapter 6). However, an AP's operator may wish to control which stations make use of the network. Some operators intentionally make this more difficult by having the AP not broadcast its SSID, as a security measure.

This approach provides little security, as the SSID may be guessed. More robust security is provided by link encryption and passwords, which we discuss in Section 3.5.5.

3.5.1.2 Control Frames: RTS/CTS and ACKs

Control frames are used to handle a form of flow control as well as acknowledgments for frames. Flow control helps ensure that a receiver can slow down a sender that is too fast. Acknowledgments help a sender know what frames have been received correctly. These concepts also apply to TCP at the transport layer (see Chapter 15). 802.11 networks support optional *request-to-send* (RTS)/*clear-to-send* (CTS) moderation of transmission for flow control. When these are enabled, prior to sending a data frame a station transmits an RTS frame, and when the recipient is willing to receive additional traffic, it responds with a CTS. After the RTS/CTS exchange, the station has a window of time (identified in the CTS frame) to transmit data frames that are acknowledged when successfully received. Such transmission coordination schemes are common in wireless networks and mimic the flow control signaling that has been used on wired serial lines for years (sometimes called hardware flow control).

The RTS/CTS exchange helps to avoid the *hidden terminal problem* by instructing each station when it is permitted to transmit, so as to avoid simultaneous transmissions from stations that cannot hear each other. Because RTS and CTS frames are short, they do not use the channel for long. An AP generally initiates an RTS/CTS exchange for a packet if the size of the packet is large enough. Typically, an AP has a configuration option called the *packet size threshold* (or similar). Frames larger than the threshold cause an RTS to be sent prior to transmission of the data. Most vendors use a default setting for this value of approximately 500 bytes if RTS/CTS exchanges are desired. In Linux, the RTS/CTS threshold can be set in the following way:

```
Linux# iwconfig wlan0 rts 250
wlan0 IEEE 802.11g ESSID:"Grizzly-5354-Aries-802.11b/g"
          Mode:Managed
          Frequency:2.427 GH
          Access Point: 00:02:6F:20:B5:84
          Bit Rate=24 Mb/s Tx-Power=0 dBm
          Retry min limit:7 RTS thr=250 B Fragment thr=2346 B
          Encryption key:xxxx- ... -xxxx [3]
          Link Quality=100/100 Signal level=46/100
          Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
          Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

The `iwconfig` command can be used to set many variables, including the RTS and fragmentation thresholds (see Section 3.5.1.3). It can also be used to determine statistics such as the number of frame errors due to wrong network ID (`ESSID`) or wrong encryption key. It also gives the number of excessive retries (i.e., the number of retransmission attempts), a rough indicator of the reliability of the link that

is popular for guiding routing decisions in wireless networks [ETX]. In WLANs with limited coverage, where hidden terminal problems are unlikely to occur, it may be preferable to disable RTS/CTS by adjusting the stations' RTS thresholds to be a high value (1500 or larger). This avoids the overhead imposed by requiring RTS/CTS exchanges for each packet.

In wired Ethernet networks, the absence of a collision indicates that a frame has been received correctly with high probability. In wireless networks, there is a wider range of reasons a frame may not be delivered correctly, such as insufficient signal or interference. To help address this potential problem, 802.11 extends the 802.3 retransmission scheme with a retransmission/acknowledgment (ACK) scheme. An acknowledgment is expected to be received within a certain amount of time for each unicast frame sent (802.11a/b/g) or each group of frames sent (802.11n or 802.11e with "block ACKs"). Multicast and broadcast frames do not have associated ACKs to avoid "ACK implosion" (see Chapter 9). Failure to receive an ACK within the specified time results in retransmission of the frame(s).

With retransmissions, it is possible to have duplicate frames formed within the network. The *Retry* bit field in the *Frame Control Word* is set when any frame represents a retransmission of a previously transmitted frame. A receiving station can use this to help eliminate duplicate frames. Stations are expected to keep a small cache of entries indicating addresses and sequence/fragment numbers seen recently. When a received frame matches an entry, the frame is discarded.

The amount of time necessary to send a frame and receive an ACK for it relates to the distance of the link and the *slot time* (a basic unit of time related to the 802.11 MAC protocol; see Section 3.5.3). The time to wait for an ACK (as well as the slot time) can be configured in most systems, although the method for doing so varies. In most cases such as home or office use, the default values are adequate. When using Wi-Fi over long distances, these values may require adjusting (see, for example, [MWLD]).

3.5.1.3 Data Frames, Fragmentation, and Aggregation

Most frames seen on a busy network are data frames, which do what one would expect—carry data. Typically, there is a one-to-one relationship between 802.11 frames and the link-layer (LLC) frames made available to higher-layer protocols such as IP. However, 802.11 supports frame *fragmentation*, which can divide frames into multiple fragments. With the 802.11n specification, it also supports frame *aggregation*, which can be used to send multiple frames together with less overhead.

When fragmentation is used, each fragment has its own MAC header and trailing CRC and is handled independently of other fragments. For example, fragments to different destinations can be interleaved. Fragmentation can help improve performance when the channel has significant interference. Unless block ACKs are used, each fragment is sent individually, producing one ACK per fragment by the receiver. Because fragments are smaller than full-size frames, if a retransmission needs to be invoked, a smaller amount of data will need to be repaired.

Fragmentation is applied only to frames with a unicast (non-broadcast or multicast) destination address. To enable this capability, the *Sequence Control* field contains a *fragment number* (4 bits) and a *sequence number* (12 bits). If a frame is fragmented, all fragments contain a common sequence number value, and each adjacent fragment has a fragment number differing by 1. A total of 15 fragments for the same frame is possible, given the 4-bit-wide field. The *More Frag* field in the *Frame Control Word* indicates that further fragments are yet to come. Terminal fragments have this bit set to 0. A destination *defragments* the original frame from fragments it receives by assembling the fragments in order based on fragment number order within the frame sequence number. Provided that all fragments constituting a sequence number have been received and the last fragment has a *More Frag* field of 0, the frame is reconstructed and passed to higher-layer protocols for processing.

Fragmentation is not often used because it does require some tuning. If used without tuning, it can worsen performance slightly. When smaller frames are used, the chance of having a bit error (see the next paragraph) can be reduced. The fragment size can usually be set from 256 bytes to 2KB as a threshold (only those frames that exceed the threshold in size are fragmented). Many APs default to not using fragmentation by setting the threshold high (such as 2437 bytes on a Linksys-brand AP).

The reason fragmentation can be useful is a fairly simple exercise in probability. If the bit error rate (BER) is P, the probability of a bit being successfully delivered is $(1 - P)$ and the probability that N bits are successfully delivered is $(1 - P)^N$. As N grows, this value shrinks. Thus, if we can shorten a frame, we can in principle improve its error-free delivery probability. Of course, if we divide a frame of size N bits into K fragments, we have to send at least $\lceil N/K \rceil$ fragments. As a concrete example, assume that we wish to send a 1500-byte (12,000-bit) frame. If we assume $P = 10^{-4}$ (a relatively high BER), the probability of successful delivery without fragmentation would be $(1 - 10^{-4})^{12,000} = .301$. So we have only about a 30% chance of such a frame being delivered without errors the first time, and on average we would have to send the frame three or four times for it to be received successfully.

If we use fragmentation for the same example and set the fragmentation threshold to 500, we produce three fragments of about 4000 bits each. The probability of one such fragment being delivered without error is about $(1 - 10^{-4})^{4000} = .670$. Thus, each fragment has about a 67% chance of being delivered successfully. Of course, we have to have three of them delivered successfully to reconstruct the whole frame. The probabilities of 3, 2, 1, and 0 fragments being delivered successfully are $(.67)^3 = 0.30$, $3(.67)^2(.33) = 0.44$, $3(0.67)(.33)^2 = .22$, and $(.33)^3 = .04$, respectively. So, although the chances that all three are delivered successfully without retries are about the same as for the nonfragmented frame being delivered successfully, the chances that two or three fragments are delivered successfully are fairly good. If this should happen, at most a single fragment would have to be retransmitted, which would take significantly less time (about a third) than sending the original 1500-byte unfragmented frame. Of course, each fragment consumes some

overhead, so if the BER is effectively 0, fragmentation only decreases performance by creating more frames to handle.

One of the enhancements provided by 802.11n is the support of frame aggregation, in two forms. One form, called the *aggregated MAC service data unit* (A-MSDU), allows for multiple complete 802.3 (Ethernet) frames to be aggregated within an 802.11 frame. The other form, called the *aggregated MAC protocol data unit* (A-MPDU), allows multiple MPDUs with the same source, destination, and QoS settings to be aggregated by being sent in short succession. The two aggregation types are depicted in Figure 3-19.

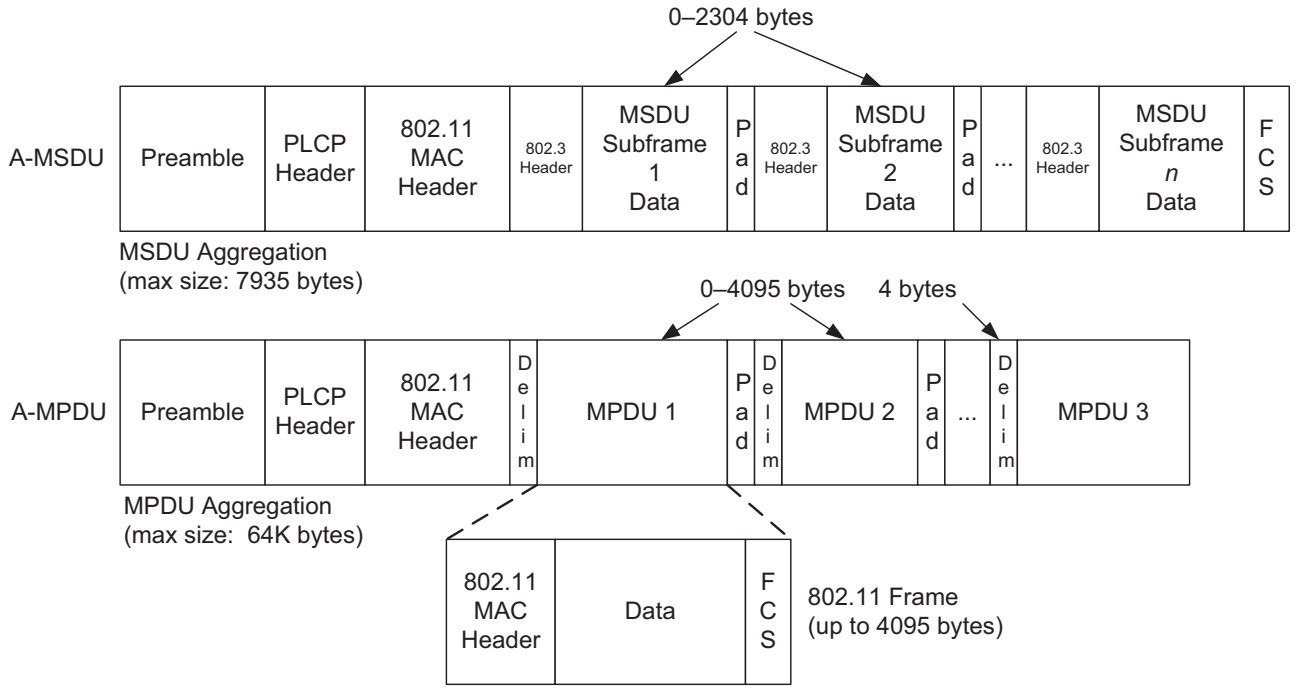

Figure 3-19 Frame aggregation in 802.11n includes A-MSDU and A-MPDU. A-MSDU aggregates frames using a single FCS. A-MPDU aggregation uses a 4-byte delimiter between each aggregated 802.11 frame. Each A-MPDU subframe has its own FCS and can be individually acknowledged using block ACKs and retransmitted if necessary.

For a single aggregate, the A-MSDU approach is technically more efficient. Each 802.3 header is ordinarily 14 bytes, which is relatively small compared to an 802.11 MAC header that could be as long as 36 bytes. Thus, with only a single 802.11 MAC header for multiple 802.3 frames, a savings of up to 22 bytes per extra aggregated frame could be achieved. An A-MSDU may be up to 7935 bytes, which can hold over 100 small (e.g., 50-byte) packets, but only a few (5) larger (1500-byte) data packets. The A-MSDU is covered by a single FCS. This larger size of an A-MSDU frame increases the chances it will be delivered with errors, and because there is only a single FCS for the entire aggregate, the entire frame would have to be retransmitted on error.

A-MPDU aggregation is a different form of aggregation whereby multiple (up to 64) 802.11 frames, each with its own 802.11 MAC header and FCS and up to 4095 bytes each, are sent together. A-MPDUs may carry up to 64KB of data—enough for more than 1000 small packets and about 40 larger (1.5KB) packets. Because each constituent frame (subframe) carries its own FCS, it is possible to selectively retransmit only those subframes received with errors. This is made possible by the *block acknowledgment* facility in 802.11n (originating in 802.11e), which is a form of extended ACK that provides feedback to a transmitter indicating which particular A-MPDU subframes were delivered successfully. This capability is similar in purpose, but not in its details, to the selective acknowledgments we will see in TCP (see Chapter 14). So, although the type of aggregation offered by A-MSDUs may be more efficient for error-free networks carrying large numbers of small packets, in practice it may not perform as well as A-MPDU aggregation [S08].

3.5.2 Power Save Mode and the Time Sync Function (TSF)

The 802.11 specification provides a way for stations to enter a limited power state, called *power save mode* (PSM). PSM is designed to save power by allowing an STA's radio receive circuitry to be powered down some of the time. Without PSM, the receiver circuitry would always be running, draining power. When in PSM, an STA's outgoing frames have a bit set in the *Frame Control Word*. A cooperative AP noticing this bit being set buffers any frames for the station until the station requests them. APs ordinarily send out beacon frames (a type of management frame) indicating various things like SSID, channel, and authentication information. When supporting stations that use PSM, APs can also indicate the presence of buffered frames to a station by setting an indication in the *Frame Control Word* of frames it sends. When stations enter PSM, they do so until the next AP beacon time, when they wake up and determine if there are pending frames stored at the AP for them.

PSM should be used with care and understanding. Although it may extend battery life, the NIC is not the only module drawing power in most wireless devices. Other parts of the system such as the screen and hard drive can be significant consumers of power, so overall battery life may not be extended much. Furthermore, using PSM can affect throughput performance significantly as idle periods are added between frame transmissions and time is spent switching modes [SHK07].

The ability to awaken an STA to check for pending frames at exactly the correct time (i.e., when an AP is about to send a beacon frame) depends on a common sense of time at the AP and the PSM stations it serves. Wi-Fi synchronizes time using the *time synchronization function* (TSF). Each station maintains a 64-bit counter reference time (in microseconds) that is synchronized with other stations in the network. Synchronization is maintained to within 4μs plus the maximum propagation delay of the PHY (for PHYs of rate 1Mb/s or more). This is accomplished by having any station that receives a TSF update (basically, a copy of the 64-bit counter sent from another station) check to see if the provided value is larger than

its own. If so, the receiving station updates its own notion of time to be the larger value. This approach ensures that clocks always move forward, but it also raises some concern that, given stations with slightly differing clock rates, the slower ones will tend to be synced to the fastest one.

With the incorporation of 802.11e (QoS) features into 802.11, the basic PSM of 802.11 has been extended to include the ability to schedule periodic batch processing of buffered frames. The frequency is expressed in terms of the number of beacon frames. The capability, called *automatic power save delivery* (APSD), uses some of the subfields of the QoS control word. APSD may be especially useful for small power-constrained devices, as they need not necessarily awaken at each beacon interval as they do in conventional 802.11 PSM. Instead, they may elect to power down their radio transceiver circuitry for longer periods of their own choosing. 802.11n also extends the basic PSM by allowing an STA equipped with multiple radio circuits operating together (see MIMO, Section 3.5.4.2) to power down all but one of the circuits until a frame is ready. This is called *spatial multiplexing* power save mode. The specification also includes an enhancement to APSD called *Power Save Multi-Poll* (PSMP) that provides a way to schedule transmissions of frames in both directions (e.g., to and from AP) at the same time.

3.5.3 802.11 Media Access Control

In wireless networks, it is much more challenging to detect a "collision" than in wired networks such as 802.3 LANs. In essence, the medium is effectively simplex, and multiple simultaneous transmitters must be avoided, by coordinating transmissions in either a centralized or a distributed manner. The 802.11 standard has three approaches to control sharing of the wireless medium, called the *point coordination function* (PCF), the *distributed coordinating function* (DCF), and the *hybrid coordination function* (HCF). HCF was brought into the 802.11 specification [802.11-2007] with the addition of QoS support in 802.11e and is also used by 802.11n. Implementation of the DCF is mandatory for any type of station or AP, but implementation of the PCF is optional and not widespread (so we shall not discuss it in detail). HCF is found in relatively new QoS-capable Wi-Fi equipment, such as 802.11n APs and earlier APs that support 802.11e. We turn our attention to DCF for now and describe HCF in the context of QoS next.

DCF is a form of CSMA/CA for contention-based access to the medium. It is used for both infrastructure and ad hoc operation. With CSMA/CA, stations listen to see if the medium is free and, if so, may have an opportunity to transmit. If not, they avoid sending for a random amount of time before checking again to see if the medium is free. This behavior is similar to how a station sensing a collision backs off when using CSMA/CD on wired LANs. Channel arbitration in 802.11 is based on CSMA/CA with enhancements to provide priority access to certain stations or frame types.

802.11 carrier sense is performed in both a physical and a virtual way. Generally, stations wait for a period of time when ready to send (called the *distributed*

inter-frame space or DIFS) to allow higher-priority stations to access the channel. If the channel becomes busy during the DIFS period, a station starts the waiting period again. When the medium appears idle, a would-be transmitter initiates the collision avoidance/backoff procedure described in Section 3.5.3.3. This procedure is also initiated after a successful (unsuccessful) transmission is indicated by the receipt (lack of receipt) of an ACK. In the case of unsuccessful transmission, the backoff procedure is initiated with different timing (using the *extended inter-frame space* or EIFS). We now discuss the implementation of DCF in more detail, including the virtual and physical carrier sense mechanisms.

3.5.3.1 Virtual Carrier Sense, RTS/CTS, and the Network Allocation Vector (NAV)

In the 802.11 MAC protocol, a *virtual carrier sense* mechanism operates by observing the *Duration* field present in each MAC frame. This is accomplished by a station listening to traffic not destined for it. The *Duration* field is present in both RTS and CTS frames optionally exchanged prior to transmission, as well as conventional data frames, and provides an estimate of how long the medium will be busy carrying the frame.

The transmitter sets the *Duration* field based on the frame length, transmit rate, and PHY characteristics (e.g., rate, etc.). Each station keeps a local counter called the *Network Allocation Vector* (NAV) that estimates how long the medium will be busy carrying the current frame, and consequently how long it will need to wait before attempting its next transmission. A station overhearing traffic with a *Duration* field greater than its NAV updates its NAV to the new value. Because the *Duration* field is present in both RTS and CTS frames, if used, any station in range of either the sender or the receiver is able to ascertain the *Duration* field value. The NAV is maintained in time units and decremented based on a local clock. The medium is considered busy when the local NAV is nonzero. It is reset to 0 upon receipt of an ACK.

3.5.3.2 Physical Carrier Sense (CCA)

Each 802.11 PHY specification (e.g., for different frequencies and radio technology) is required to provide a function for assessing whether the channel is clear based upon energy and waveform recognition (usually recognition of a well-formed PLCP). This function is called *clear channel assessment* (CCA) and its implementation is PHY-dependent. The CCA capability represents the physical carrier sense capability for the 802.11 MAC to understand whether the medium is currently busy. It is used in conjunction with the NAV to determine when a station must *defer* (wait) prior to transmission.

3.5.3.3 DCF Collision Avoidance/Backoff Procedure

Upon determining that the channel is likely to be free (i.e., because the NAV duration has been met and CCA does not indicate a busy channel), a station defers access prior to transmission. Because many stations may have been waiting for the channel to become free, each station computes and waits for a *backoff time* prior

to sending. The backoff time is equal to the product of a random number and the *slot time* (unless the station attempting to transmit already has a nonzero backoff time, in which case it is not recomputed). The slot time is PHY-dependent but is generally a few tens of microseconds. The random number is drawn from a uniform distribution over the interval [0, CW], where the *contention window* (CW) is an integer containing a number of time slots to wait, with limits $aCWmin \leq CW \leq aCWmax$ defined by the PHY. The set of CW values increases by powers of 2 (minus 1) beginning with the PHY-specific constant *aCWmin* value and continuing up to and including the constant *aCWmax* value for each successive transmission attempt. This is similar in effect to Ethernet's backoff procedure initiated during a collision detection event.

In a wireless environment, collision *detection* is not practical because it is difficult for a transmitter and receiver to operate simultaneously in the same piece of equipment and hear any transmissions other than its own, so collision *avoidance* is used instead. In addition, ACKs are generated in response to unicast frames to determine whether a frame has been delivered successfully. A station receiving a correct frame begins transmitting an ACK after waiting a small period of time (called the *Short Interframe Space* or SIFS), without regard to the busy/idle state of the medium. This should not cause a problem because the SIFS value is always smaller than DIFS, so in effect stations generating ACKs get priority access to the channel to complete their transactions. The source station waits a certain amount of time without receiving an ACK frame before concluding that a transmission has failed. Upon failure, the backoff procedure discussed previously is initiated and the frame is retried. The same procedure is initiated if a CTS is not received in response to an earlier RTS within a certain (different) amount of time (a constant called *CTStimeout*).

3.5.3.4 HCF and 802.11e/n QoS

Clauses 5, 6, 7, and 9 of the 802.11 standard [802.11-2007] are based in part on the work of the 802.11e group within IEEE, and the terms 802.11e, Wi-Fi QoS, and WMM (for Wi-Fi Multimedia) are often used. They cover the *QoS facility*—changes to the 802.11 MAC-layer and system interfaces in support of multimedia applications such as voice over IP (VoIP) and streaming video. Whether the QoS facility is really necessary or not often depends on the congestion level of the network and the types of applications to be supported. If utilization of the network tends to be low, the QoS MAC support may be unnecessary, although some of the other 802.11e capabilities may still be useful (e.g., block ACKs and APSD). In situations where utilization and congestion are high and there is a need to support a low-jitter delivery capability for services such as VoIP, QoS support may be desirable. These specifications are relatively new, so QoS-capable Wi-Fi equipment is likely to be more expensive and complex than non-QoS equipment.

The QoS facility introduces new terminology such as *QoS stations* (QSTAs), *QoS access points* (QAPs), and the *QoS BSS* (QBSS, a BSS supporting QoS). In general, any of the devices supporting QoS capabilities also support conventional

non-QoS operation. 802.11n "high-throughput" stations (called HT STAs) are also QSTAs. A new form of coordination function, the *hybrid coordination function* (HCF), supports both contention-based and controlled channel access, although the controlled channel variant is seldom used. Within the HCF, there are two specified channel access methods that can operate together: *HFCA-controlled channel access* (HCCA) and the more popular *enhanced DCF channel access* (EDCA), corresponding to reservation-based and contention-based access, respectively. There is also some support for *admission control*, which may deny connectivity entirely under high load.

EDCA builds upon the basic DCF access. With EDCA, there are eight *user priorities* (UPs) that are mapped to four *access categories* (ACs). The user priorities use the same structure as 802.1d priority tags and are numbered 1 through 7, with 7 being the highest priority. (There is also a 0 priority between 2 and 3.) The four ACs are nominally intended for background, best-effort, video, and audio traffic. Priorities 1 and 2 are intended for the background AC, priorities 0 and 3 are for the best-effort AC, 4 and 5 are for the video AC, and 6 and 7 are for the voice AC. For each AC, a variant of DCF contends for channel access credits called *transmit opportunities* (TXOPs), using alternative MAC parameters that tend to favor the higher-priority traffic. In EDCA, many of the various MAC parameters from DCF (e.g., DIFS, *aCWmin*, *aCWmax*) become adjustable as configuration parameters. These values are communicated to QSTAs using management frames.

HCCA builds loosely upon PCF and uses polling-controlled channel access. It is designed for synchronous-style access control and takes precedence ahead of the contention-based access of EDCA. A *hybrid coordinator* (HC) is located within an AP and has priority to allocate channel accesses. Prior to transmission, a station can issue a *traffic specification* (TSPEC) for its traffic and use UP values between 8 and 15. The HC can allocate reserved TXOPs to such requests to be used during short-duration controlled access phases of frame exchange that take place before EDCA-based frame transmission. The HC can also deny TXOPs to TSPECs based on admission control policies set by the network administrator. The HCF exploits the virtual carrier sense mechanism discussed earlier with DCF to keep contention-based stations from interfering with contention-free access. Note that a single network comprising QSTAs and conventional stations can have both HCF and DCF running simultaneously by alternating between the two, but ad hoc networks do not support the HC and thus do not handle TSPECs and do not perform admission control. Such networks might still run HCF, but TXOPs are gained through EDCA-based contention.

3.5.4 Physical-Layer Details: Rates, Channels, and Frequencies

The [802.11-2007] standard now includes the following earlier amendments: 802.11a, 802.11b, 802.11d, 802.11g, 802.11h, 802.11i, 802.11j, and 802.11e. The 802.11n standard was adopted as an amendment to 802.11 in 2009 [802.11n-2009]. Most of these amendments provide additional modulation, coding, and operating

frequencies for 802.11 networks, but 802.11n also adds multiple data streams and a method for aggregating multiple frames (see Section 3.5.1.3). We will avoid detailed discussion of the physical layer, but to appreciate the breadth of options, Table 3-2 includes those parts of the 802.11 standard that describe this layer in particular.

Table 3-2 Parts of the 802.11 standard that describe the physical layer

Standard (Clause)	Speeds (Mb/s)	Frequency Range; Modulation	Channel Set
802.11a (Clause 17)	6, 9, 12, 18, 24, 36, 48, 54	5.16–5.35 and 5.725–5.825GHz; OFDM	34–165 (varies by country) 20MHz/10MHz/5MHz channel width options
802.11b (Clause 18)	1, 2, 5.5, 11	2.401–2.495GHz; DSSS	1–14 (varies by country)
802.11g (Clause 19)	1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48, 54 (plus 22, 33)	2.401–2.495GHz; OFDM	1–14 (varies by country)
802.11n	6.5–600 with many options (up to 4 MIMO streams)	2.4 and 5GHz modes with 20MHz- or 40MHz-wide channels; OFDM	1–13 (2.4GHz band); 36–196 (5GHz band) (varies by country)
802.11y	(Same as 802.11-2007)	3.650–3.700GHz (licensed); OFDM	1–25, 36–64, 100–161 (varies by country)

The first column gives the original standard name and its present location in [802.11-2007], plus details for the 802.11n and 802.11y amendments. It is important to note from this table that 802.11b/g operate in the 2.4GHz *Industrial, Scientific, and Medical* (ISM) band, 802.11a operates only in the higher 5GHz *Unlicensed National Information Infrastructure* (U-NII) band, and 802.11n can operate in both. The 802.11y amendment provides for licensed use in the 3.65–3.70GHz band within the United States. An important practical consequence of the data in this table is that 802.11b/g equipment does not interoperate or interfere with 802.11a equipment, but 802.11n equipment may interfere with either if not deployed carefully.

3.5.4.1 Channels and Frequencies

Regulatory bodies (e.g., the Federal Communications Commission in the United States) divide the electromagnetic spectrum into frequency ranges allocated for various uses across the world. For each range and use, a license may or may not be required, depending on local policy. In 802.11, there are sets of channels that may be used in various ways at various power levels depending on the regulatory domain or country. Wi-Fi channels are numbered in 5MHz units starting at some base center frequency. For example, channel 36 with a base center frequency

of 5.00GHz gives the frequency 5000 + 36 * 5 = 5180MHz, the center frequency of channel 36. Although channel center frequencies are 5MHz apart from each other, channels may be *wider* than 5MHz (up to 40MHz for 802.11n). Consequently, some channels within channel sets of the same band usually overlap. Practically speaking, this means that transmissions on one channel might interfere with transmissions on nearby channels.

Figure 3-20 presents the channel-to-frequency mapping for the 802.11b/g channels in the 2.4GHz ISM band. Each channel is 22MHz wide. Not all channels are available for legal use in every country. For example, channel 14 is authorized at present for use only in Japan, and channels 12 and 13 are authorized for use in Europe, while the United States permits channels 1 through 11 to be used. Other countries may be more restrictive (see Annex J of the 802.11 standard and amendments). Note that policies and licensing requirements may change over time.

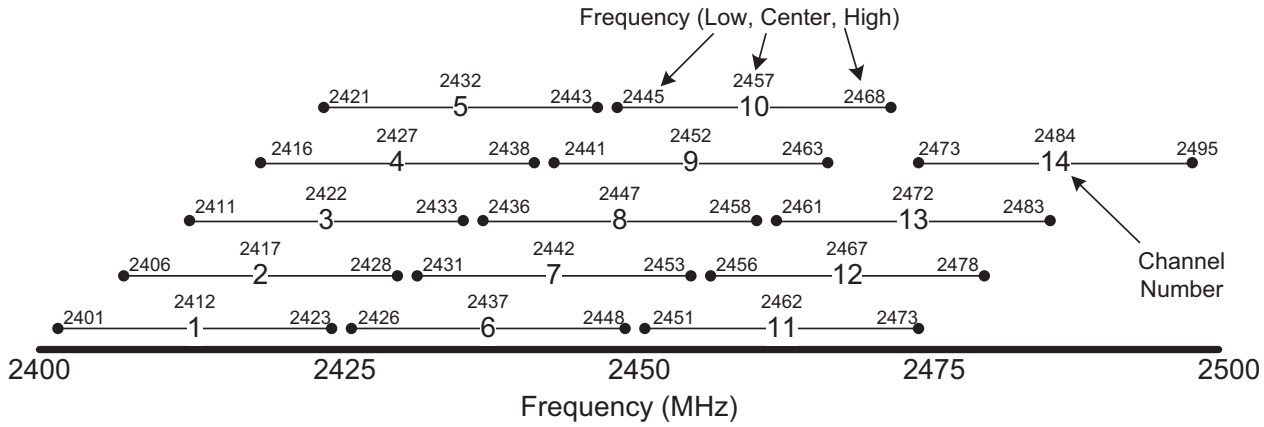

Figure 3-20 The 802.11b and 802.11g standards use a frequency band between about 2.4GHz and 2.5GHz. This band is divided into fourteen 22MHz-wide overlapping channels, of which some subset is generally available for legal use depending on the country of operation. It is advisable to assign nonoverlapping channels, such as 1, 6, and 11 in the United States, to multiple base stations operating in the same area. Only a single 40MHz 802.11n channel may be used in this band without overlap.

As shown in Figure 3-20, the effect of overlapping channels is now clear. A transmitter on channel 1, for example, overlaps with channels 2, 3, 4, and 5 but not higher channels. This becomes important when selecting which channels to assign for use in environments where multiple access points are to be used and even more important when multiple access points serving multiple different networks in the same area are to be used. One common approach in the United States is to assign up to three APs in an area using nonoverlapping channels 1, 6, and 11, as channel 11 is the highest-frequency channel authorized for unlicensed use in the United States. In cases where other WLANs may be operating in the same bands, it is worth considering jointly planning channel settings with all the affected WLAN administrators.

As shown in Figure 3-21, 802.11a/n/y share a somewhat more complicated channel set but offer a larger number of nonoverlapping channels to use (i.e., 12 unlicensed 20MHz channels in the United States).

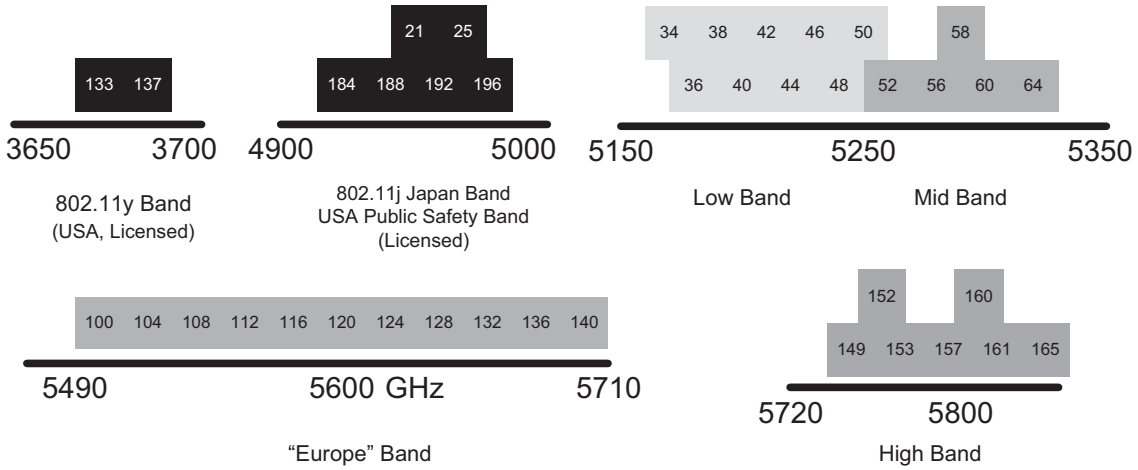

Figure 3-21 Many of the approved 802.11 channel numbers and center frequencies for 20MHz channels. The most common range for unlicensed use involves the U-NII bands, all above 5GHz. The lower band is approved for use in most countries. The "Europe" band is approved for use in most European countries, and the high band is approved for use in the United States and China. Channels are typically 20MHz wide for 802.11a/y but may be 40MHz wide for 802.11n. Narrower channels and some channels available in Japan are also available (not shown).

In Figure 3-21, the channels are numbered in 5MHz increments, but different channel widths are available: 5MHz, 10MHz, 20MHz, and 40MHz. The 40MHz channel width is an option with 802.11n (see Section 3.5.4.2), along with several proprietary Wi-Fi systems that aggregate two 20MHz channels (called channel bonding).

For typical Wi-Fi networks, an AP has its operating channel assigned during installation, and client stations change channels in order to associate with the AP. When operating in ad hoc mode, there is no controlling AP, so a station is typically hand-configured with the operating channel. The sets of channels available and operating power may be constrained by the regulatory environment, the hardware capabilities, and possibly the supporting driver software.

3.5.4.2 802.11 Higher Throughput/802.11n

In late 2009, the IEEE standardized 802.11n [802.11n-2009] as an amendment to [802.11-2007]. It makes a number of important changes to 802.11. To support higher throughput, it incorporates support for *multiple input, multiple output* (MIMO) management of multiple simultaneously operating data streams carried on multiple antennas, called *spatial streams*. Up to four such spatial streams are supported on a given channel. 802.11n channels may be 40MHz wide (using two adjacent 20MHz channels), twice as wide as conventional channels in 802.11a/b/g/y. Thus, there is an immediate possibility of having up to eight times the maximum data rate of

802.11a/g (54Mb/s), for a total of 432Mb/s. However, 802.11n also improves the single-stream performance by using a more efficient modulation scheme (802.11n uses MIMO- orthogonal frequency division multiplexing (OFDM) with up to 52 data subcarriers per 20MHz channel and 108 per 40MHz channel, instead of 48 in 802.11a and 802.11g), plus a more efficient forward error-correcting code (rate 5/6 instead of 3/4), bringing the per-stream performance to 65Mb/s (20MHz channel) or 135Mb/s (40MHz channel). By also reducing the *guard interval* (GI, a forced idle time between symbols) duration to 400ns from the legacy 800ns, the maximum per-stream performance is raised to about 72.2Mb/s (20MHz channel) and 150Mb/s (40MHz channel). With four spatial streams operating in concert perfectly, this provides a maximum of about 600Mb/s.

Some 77 combinations of modulation and coding options are supported by 802.11n, including 8 options for a single stream, 24 using the same or *equal modulation* (EQM) on all streams, and 43 using *unequal modulation* (UEQM) on multiple streams. Table 3-3 gives some of the combinations for modulation and coding scheme according to the first 33 values of the *modulation and coding scheme* (MCS) value. Higher values (33–76) include combinations for two channels (values 33–38), three channels (39–52), and four channels (53–76). MCS value 32 is a special combination where the signals in the two halves of the 40MHz channel

Table 3-3 MCS values for 802.11n include combinations of equal and unequal modulation, different FEC coding rates, up to four spatial streams using 20MHz- or 40MHz-wide channels, and an 800ns or 400ns GI. The 77 combinations provide data rates from 6Mb/s to 600Mb/s.

MCS Value	Modulation Type	FEC Code Rate	Spatial Streams	Rates (Mb/s) (20MHz) [800/400ns]	Rates (Mb/s) (40MHz) [800/400ns]
0	BPSK	1/2	1	6.5/7.2	13.5/15
1	QPSK	1/2	1	13/14.4	27/30
2	QPSK	3/4	1	19.5/21.7	40.5/45
3	16-QAM	1/2	1	26/28.9	54/60
4	16-QAM	3/4	1	39/43.3	81/90
5	64-QAM	2/3	1	52/57.8	108/120
6	64-QAM	3/4	1	58.5/65	121.5/135
7	64-QAM	5/6	1	65/72.2	135/150
8	BPSK	1/2	2	13/14.4	27/30
...	...	...	...	...	...
15	64-QAM	5/6	2	130/144.4	270/300
16	BPSK	1/2	3	19.5/21.7	40.5/45
...	...	...	...	...	...
31	64-QAM	5/6	4	260/288.9	540/600
32	BPSK	1/2	1	N/A	6/6.7
...	...	...	...	...	...
76	64x3/16x1-QAM	3/4	4	214.5/238.3	445.5/495

contain the same information. Each data rate column gives two values, one using the legacy 800ns GI and one giving the greater data rate available using the shorter 400ns GI. The underlined values, 6Mb/s and 600Mb/s, represent the smallest and largest throughput rates, respectively.

Table 3-3 shows the various combinations of coding, including *binary phase shift keying* (BPSK), *quadrature phase shift keying* (QPSK), and various levels of *quadrature amplitude modulation* (16- and 64-QAM), available with 802.11n. These modulation schemes provide an increasing data rate for a given channel bandwidth. However, the more high-performance and complex a modulation scheme, the more vulnerable it tends to be to noise and interference. *Forward error correction* (FEC) includes a set of methods whereby redundant bits are introduced at the sender that can be used to detect and repair bit errors introduced during delivery. For FEC, the *code rate* is the ratio of the effective useful data rate to the rate imposed on the underlying communication channel. For example, a ½ code rate would deliver 1 useful bit for every 2 bits sent.

802.11n may operate in one of three modes. In 802.11n-only environments, the optional so-called *greenfield mode*, the PLCP contains special bit arrangements ("training sequences") known only to 802.11n equipment and does not interoperate with legacy equipment. To maintain compatibility, 802.11n has two other interoperable modes. However, both of these impose a performance penalty to native 802.11n equipment. One mode, called *non-HT mode*, essentially disables all 802.11n features but remains compatible with legacy equipment. This is not a very interesting mode, so we shall not discuss it further. However, a required mode called *HT-mixed mode* supports both 802.11n and legacy operation, depending on which stations are communicating. The information required to convey an AP's 802.11n capability to HT STAs yet protect legacy STAs is provided in the PLCP, which is augmented to contain both HT and legacy information and is transmitted at a slower rate than in greenfield mode so that it can be processed by legacy equipment. HT protection also requires an HT AP to use self-directed CTS frames (or RTS/CTS frame exchanges) at the legacy rate to inform legacy stations when it will use shared channels. Even though RTS/CTS frames are short, the requirement to send them at the legacy rate (6Mb/s) can significantly reduce an 802.11n WLAN's performance.

When deploying an 802.11n AP, care should be taken to set up appropriate channel assignments. When using 40MHz channels, 802.11n APs should be operated in the U-NII bands above 5GHz as there is simply not enough useful spectrum to use these wider channels in the 2.4GHz ISM band. An optional BSS feature called *phased coexistence operation* (PCO) allows an AP to periodically switch between 20MHz and 40MHz channel widths, which can provide better coexistence between 802.11n APs operating near legacy equipment at the cost of some additional throughput. Finally, it is worth mentioning that 802.11n APs generally require more power than conventional APs. This higher power level exceeds the basic 15W provided by 802.3af *power-over-Ethernet* (PoE) system wiring, meaning that PoE+ (802.3at, capable of 30W) should be used unless some other form of power such as a direct external power supply is available.

3.5.5 Wi-Fi Security

There has been considerable evolution in the security model for 802.11 networks. In its early days, 802.11 used an encryption method known as *wired equivalent privacy* (WEP). WEP was later shown to be so weak that some replacement was required. Industry responded with *Wi-Fi protected access* (WPA), which replaced the way keys are used with encrypted blocks (see Chapter 18 for the basics of cryptography). In WPA, a scheme called the *Temporal Key Integrity Protocol* (TKIP) ensures, among other things, that each frame is encrypted with a different encryption key. It also includes a message integrity check, called *Michael*, that fixed one of the major weaknesses in WEP. WPA was created as a placeholder that could be used on fielded WEP-capable equipment by way of a firmware upgrade while the IEEE 802.11i standards group worked on a stronger standard that was ultimately absorbed into Clause 8 of [802.11-2007] and dubbed "WPA2" by industry. Both WEP and WPA use the RC4 encryption algorithm [S96]. WPA2 uses the *Advanced Encryption Standard* (AES) algorithm [AES01].

The encryption techniques we just discussed are aimed at providing privacy between the station and AP, assuming the station has legitimate authorization to be accessing the network. In WEP, and small-scale environments that use WPA or WPA2, authorization is typically implemented by pre-placing a shared key or password in each station as well as in the AP during configuration. A user knowing the key is assumed to have legitimate access to the network. These keys are also frequently used to initialize the encryption keys used to ensure privacy. Using such *pre-shared keys* (PSKs) has limitations. For example, an administrator may have considerable trouble in providing keys only to authorized users. If a user becomes de-authorized, the PSK has to be replaced and all legitimate users informed. This approach does not scale to environments with many users. As a result, WPA and later standards support a *port-based network access control* standard called 802.1X [802.1X-2010]. It provides a way to carry the *Extensible Authentication Protocol* (EAP) [RFC3748] in IEEE 802 LANs (called EAPOL), including 802.3 and 802.11 [RFC4017]. EAP, in turn, can be used to carry many other standard and non-standard authentication protocols. It can also be used to establish keys, including WEP keys. Details of these protocols are given in Chapter 18, but we shall also see the use of EAP when we discuss PPP in Section 3.6.

With the completion of the IEEE 802.11i group's work, the RC4/TKIP combination in WPA was extended with a new algorithm called CCMP as part of WPA2. CCMP is based on using the *counter mode* (CCM [RFC3610]) of the AES for confidentiality with *cipher block chaining message authentication code* (CBC-MAC; note the "other" use of the term MAC here) for authentication and integrity. All AES processing is performed using a 128-bit block size and 128-bit keys. CCMP and TKIP form the basis for a Wi-Fi security architecture named the *Robust Security Network* (RSN), which supports *Robust Security Network Access* (RSNA). Earlier methods, such as WEP, are called *pre-RSNA methods*. RSNA compliance requires support for CCMP (TKIP is optional), and 802.11n does away with TKIP entirely. Table 3-4 provides a summary of this somewhat complicated situation.

Table 3-4 Wi-Fi security has evolved from WEP, which was found to be insecure, to WPA, to the now-standard WPA2 collection of algorithms.

Name/Standard	Cipher	Key Stream Management	Authentication
WEP (pre-RSNA)	RC4	(WEP)	PSK, (802.1X/EAP)
WPA	RC4	TKIP	PSK, 802.1X/EAP
WPA2/802.11(i)	CCMP	CCMP, (TKIP)	PSK, 802.1X/EAP

In all cases, both pre-shared keys as well as 802.1X can be used for authentication and initial keying. The major attraction of using 802.1X/EAP is that a managed authentication server can be used to provide access control decisions on a per-user basis to an AP. For this reason, authentication using 802.1X is sometimes referred to as "Enterprise" (e.g., WPA-Enterprise). EAP itself can encapsulate various specific authentication protocols, which we discuss in more detail in Chapter 18.

3.5.6 Wi-Fi Mesh (802.11s)

The IEEE is working on the 802.11s standard, which covers Wi-Fi *mesh* operation. With mesh operation, wireless stations can act as data-forwarding agents (like APs). The standard is not yet complete as of writing (mid-2011). The draft version of 802.11s defines the *Hybrid Wireless Routing Protocol* (HWRP), based in part on the IETF standards for *Ad-Hoc On-Demand Distance Vector* (AODV) routing [RFC3561] and the *Optimized Link State Routing* (OLSR) protocol [RFC3626]. *Mesh stations* (mesh STAs) are a type of QoS STA and may participate in HWRP or other routing protocols, but compliant nodes must include an implementation of HWRP and the associated *airtime link metric*. Mesh nodes coordinate using EDCA or may use an optional coordinating function called *mesh deterministic access*. *Mesh points* (MPs) are those nodes that form mesh links with neighbors. Those that also include AP functionality are called *mesh APs* (MAPs). Conventional 802.11 stations can use either APs or MAPs to access the rest of the wireless LAN.

The 802.11s draft specifies a new optional form of security for RSNA called *Simultaneous Authentication of Equals* (SAE) authentication [SAE]. This security protocol is a bit different from others because it does not require lockstep operation between a specially designated initiator and responder. Instead, stations are treated as equals, and any station that first recognizes another may initiate a security exchange (or this may happen simultaneously as two stations initiate an association).

3.6 Point-to-Point Protocol (PPP)

PPP stands for the *Point-to-Point Protocol* [RFC1661][RFC1662][RFC2153]. It is a popular method for carrying IP datagrams over serial links—from low-speed dial-up modems to high-speed optical links [RFC2615]. It is widely deployed by some DSL

service providers, which also use it for assigning Internet system parameters (e.g., initial IP address and domain name server; see Chapter 6).

PPP should be considered more of a collection of protocols than a single protocol. It supports a basic method to establish a link, called the *Link Control Protocol* (LCP), as well as a family of NCPs, used to establish network-layer links for various kinds of protocols, including IPv4 and IPv6 and possibly non-IP protocols, after LCP has established the basic link. A number of related standards cover control of compression and encryption for PPP, and a number of authentication methods can be employed when a link is brought up.

3.6.1 Link Control Protocol (LCP)

The LCP portion of PPP is used to establish and maintain a low-level two-party communication path over a point-to-point link. PPP's operation therefore need be concerned only with two ends of a single link; it does not need to handle the problem of mediating access to a shared resource like the MAC-layer protocols of Ethernet and Wi-Fi.

PPP generally, and LCP more specifically, imposes minimal requirements on the underlying point-to-point link. The link must support bidirectional operation (LCP uses acknowledgments) and operate either asynchronously or synchronously. Typically, LCP establishes a link using a simple bit-level framing format based on the *High-Level Data Link Control* (HDLC) protocol. HDLC was already a well-established framing format by the time PPP was designed [ISO3309] [ISO4335]. IBM modified it to form *Synchronous Data Link Control* (SDLC), a protocol used as the link layer in its proprietary *System Network Architecture* (SNA) protocol suite. HDLC was also used as the basis for the LLC standard in 802.2 and ultimately for PPP as well. The format is shown in Figure 3-22.

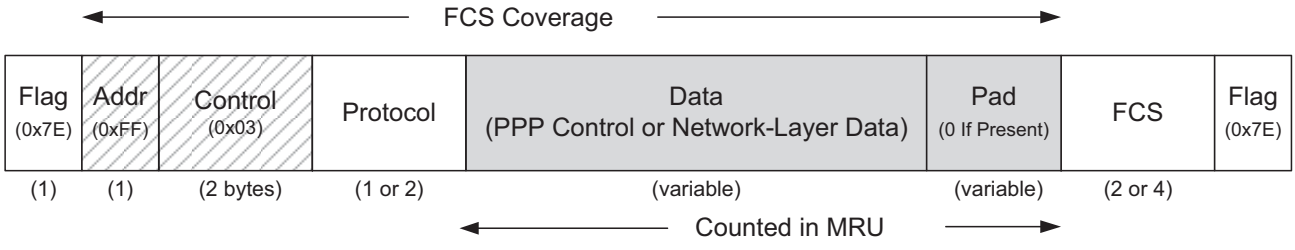

Figure 3-22 The PPP basic frame format was borrowed from HDLC. It provides a protocol identifier, payload area, and 2- or 4-byte FCS. Other fields may or may not be present, depending on compression options.

The PPP frame format, in the common case when HDLC-like framing is used as shown in Figure 3-22, is surrounded by two 1-byte *Flag* fields containing the fixed value 0x7E. These fields are used by the two stations on the ends of the point-to-point link for finding the beginning and end of the frame. A small problem arises if the value 0x7E itself occurs inside the frame. This is handled in one of

two ways, depending on whether PPP is operating over an asynchronous or a synchronous link. For asynchronous links, PPP uses *character stuffing* (also called *byte stuffing*). If the flag character appears elsewhere in the frame, it is replaced with the 2-byte sequence 0x7D5E (0x7D is known as the "PPP escape character"). If the escape character itself appears in the frame, it is replaced with the 2-byte sequence 0x7D5D. Thus, the receiver replaces 0x7D5E with 0x7E and 0x7D5D with 0x7D upon receipt. On synchronous links (e.g., T1 lines, T3 lines), PPP uses *bit stuffing*. Noting that the flag character has the bit pattern 01111110 (a contiguous sequence of six 1 bits), bit stuffing arranges for a 0 bit to be inserted after any contiguous string of five 1 bits appearing in a place other than the flag character itself. Doing so implies that bytes may be sent as more than 8 bits, but this is generally OK, as low layers of the serial processing hardware are able to "unstuff" the bit stream, restoring it to its prestuffed pattern.

After the first *Flag* field, PPP adopts the HDLC *Address* (*Addr*) and *Control* fields. In HDLC, the *Address* field would specify which station is being addressed, but because PPP is concerned only with a single destination, this field is always defined to have the value 0xFF (all stations). The *Control* field in HDLC is used to indicate frame sequencing and retransmission behavior. As these link-layer reliability functions are not ordinarily implemented by PPP, the *Control* field is set to the fixed value 0x03. Because both the *Address* and *Control* fields are fixed constants in PPP, they are often omitted during transmission with an option called *Address and Control Field Compression* (ACFC), which essentially eliminates the two fields.

Note

There has been considerable debate over the years as to how much reliability link-layer networks should provide, if any. With Ethernet, up to 16 retransmission attempts are made before giving up. Typically, PPP is configured to do no retransmission, although there do exist specifications for adding retransmission [RFC1663]. The trade-off can be subtle and is dependent on the types of traffic to be carried. A detailed discussion of the considerations is contained in [RFC3366].

The *Protocol* field of the PPP frame indicates the type of data being carried. Many different types of protocols can be carried in a PPP frame. The official list and the assigned number used in the *Protocol* field are given by the "Point-to-Point Protocol Field Assignments" document [PPPn]. In conforming to the HDLC specification, any protocol numbers are assigned such that the least significant bit of the most significant byte equals 0 and the least significant bit of the least significant byte equals 1. Values in the (hexadecimal) range 0x0000–0x3FFF identify network-layer protocols, and values in the 0x8000–0xBFFF range identify data belonging to an associated NCP. Protocol values in the range 0x4000–0x7FFF are used for "low-volume" protocols with no associated NCP. Protocol values in the range 0xC000–0XEFFF identify control protocols such as LCP. In some circumstances the *Protocol*

field can be compressed to a single byte, if the *Protocol Field Compression* (PFC) option is negotiated successfully during link establishment. This is applicable to protocols with protocol numbers in the range 0x0000–0x00FF, which includes most of the popular network-layer protocols. Note, however, that LCP packets always use the 2-byte uncompressed format.

The final portion of the PPP frame contains a 16-bit FCS (a CRC16, with generator polynomial 10001000000100001) covering the entire frame except the *FCS* field itself and *Flag* bytes. Note that the FCS value covers the frame before any byte or bit stuffing has been performed. With an LCP option (see Section 3.6.1.2), the CRC can be extended from 16 to 32 bits. This case uses the same CRC32 polynomial mentioned previously for Ethernet.

3.6.1.1 LCP Operation

LCP has a simple encapsulation beyond the basic PPP packet. It is illustrated in Figure 3-23.

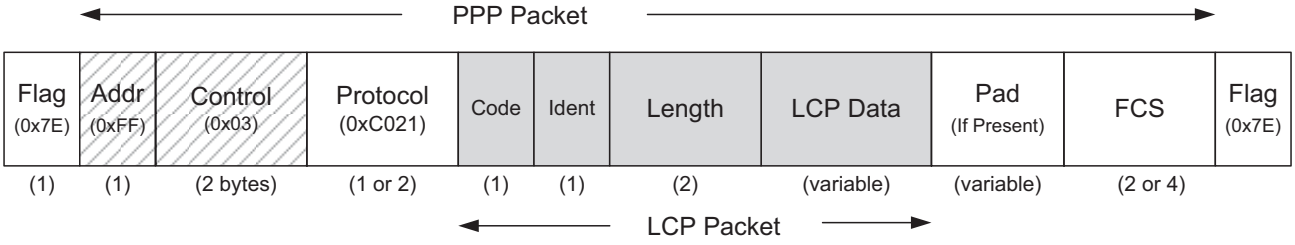

Figure 3-23 The LCP packet is a fairly general format capable of identifying the type of encapsulated data and its length. LCP frames are used primarily in establishing a PPP link, but this basic format also forms the basis of many of the various network control protocols.

The PPP *Protocol* field value for LCP is always 0xC021, which is not eliminated using PFC, so as to minimize ambiguity. The *Ident* field is a sequence number provided by the sender of LCP request frames and is incremented for each subsequent message. When forming a reply (ACK, NACK, or REJECT response), this field is constructed by copying the value included in the request to the response packet. In this fashion, the requesting side can identify replies to the appropriate request by matching identifiers. The *Code* field gives the type of operation being either requested or responded to: configure-request (0x01), configure-ACK (0x02), configure-NACK (0x03), configure-REJECT (0x04), terminate-request (0x05), terminate-ACK (0x06), code-REJECT (0x07), protocol-REJECT (0x08), echo-request (0x09), echo-reply (0x0A), discard-request (0x0B), identification (0x0C), and time-remaining (0x0D). Generally, ACK messages indicate acceptance of a set of options, and NACK messages indicate a partial rejection with suggested alternatives. A REJECT message rejects one or more options entirely. A rejected code indicates that one of the field values contained in a previous packet is unknown. The *Length*

field gives the length of the LCP packet in bytes and is not permitted to exceed the link's *maximum received unit* (MRU), a form of maximum advised frame limit we shall discuss later. Note that the *Length* field is part of the LCP protocol; the PPP protocol in general does not provide such a field.

The main job of LCP is to bring up a point-to-point link to a minimal level. *Configure* messages cause each end of the link to start the basic configuration procedure and establish agreed-upon options. *Termination* messages are used to clear a link when complete. LCP also provides some additional features mentioned previously. *Echo Request/Reply* messages may be exchanged anytime a link is active by LCP in order to verify operation of the peer. The *Discard Request* message can be used for performance measurement; it instructs the peer to discard the packet without responding. The *Identification* and *Time-Remaining* messages are used for administrative purposes: to know the type of the peer system and to indicate the amount of time allowed for the link to remain established (e.g., for administrative or security reasons).

Historically, one common problem with point-to-point links occurs if a remote station is in *loopback mode* or is said to be "looped." Telephone company wide area data circuits are sometimes put into loopback mode for testing—data sent at one side is simply returned from the other. Although this may be useful for line testing, it is not at all helpful for data communication, so LCP includes ways to send a *magic number* (an arbitrary number selected by the sender) to see if it is immediately returned in the same message type. If so, the line is detected as being looped, and maintenance is likely required.

To get a better feeling for how PPP links are established and options are negotiated, Figure 3-24 illustrates a simplified packet exchange timeline as well as a simplified state machine (implemented at both ends of the link).

The link is considered to be established once the underlying protocol layer has indicated that an association has become active (e.g., carrier detected for modems). Link quality testing, which involves an exchange of link quality reports and acknowledgments (see Section 3.6.1.2), may also be accomplished during this period. If the link requires authentication, which is common, for example, when dialing in to an ISP, a number of additional exchanges may be required to establish the authenticity of one or both parties attached to the link. The link is terminated once the underlying protocol or hardware has indicated that the association has stopped (e.g., carrier lost) or after having sent a link termination request and received a termination ACK from the peer.

3.6.1.2 LCP Options

Several *options* can be negotiated by LCP as it establishes a link for use by one or more NCPs. We shall discuss two of the more common ones. The *Asynchronous Control Character Map* (ACCM) or simply "asyncmap" option defines which control characters (i.e., ASCII characters in the range 0x00–0x1F) need to be "escaped" as PPP operates. Escaping a character means that the true value of the character is

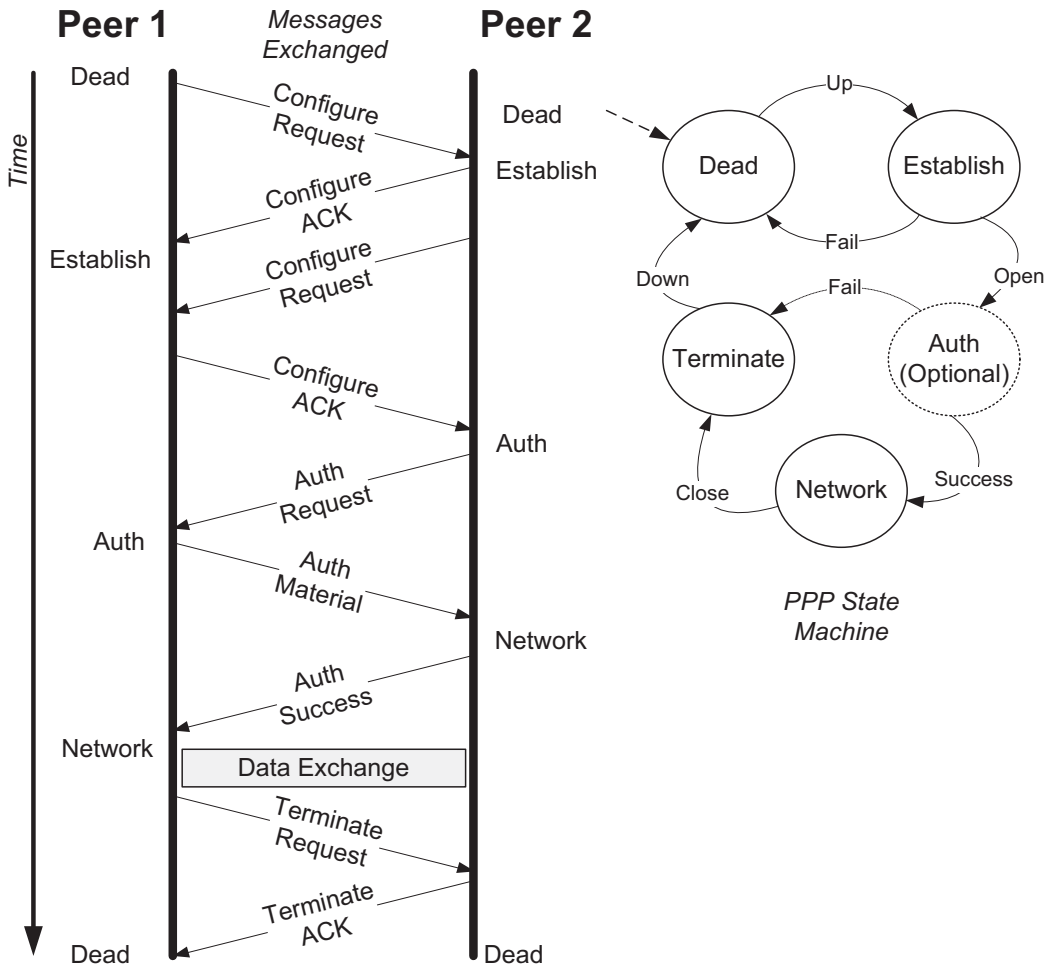

Figure 3-24 LCP is used to establish a PPP link and agree upon options by each peer. The typical exchange involves a pair of configure requests and ACKs that contain the option list, an authentication exchange, data exchange (not pictured), and a termination exchange. Because PPP is such a general-purpose protocol with many parts, many other types of operations may occur between the establishment of a link and its termination.

not sent, but instead the PPP escape character (0x7D) is stuffed in front of a value formed by XORing the original control character with the value 0x20. For example, the XOFF character (0x13) would be sent as (0x7D33). ACCM is used in cases where control characters may affect the operation of the underlying hardware. For example, if software flow control using XON/XOFF characters is enabled and the XOFF character is passed through the link unescaped, the data transfer ceases until the hardware observes an XON character. The asyncmap option is generally specified as a 32-bit hexadecimal number where a 1 bit in the *n*th least significant bit position indicates that the control character with value *n* should be escaped. Thus, the asyncmap 0xffffffff would escape all control characters, 0x00000000 would escape none of them, and 0x000A0000 would escape XON (value 0x11) and XOFF (value 0x13). Although the value 0xffffffff is the specified default, many links today can operate safely with the asyncmap set to 0x00000000.

Because PPP lacks a *Length* field and serial lines do not typically provide framing, no immediate hard limit is set on the length of a PPP frame, in theory. In practice, some maximum frame size is typically given by specifying the MRU. When a host specifies an MRU option (type 0x01), the peer is requested to never send frames longer than the value provided in the MRU option. The MRU value is the length of the data field in bytes; it does not count the various other PPP overhead fields (i.e., *Protocol*, *FCS*, *Flag* fields). Typical values are 1500 or 1492 but may be as large as 65,535. A minimum of 1280 is required for IPv6 operations. The standard requires PPP implementations to accept frames as large as 1500 bytes, so the MRU serves more as advice to the peer in choosing the packet size than as a hard limit on the size. When small packets are interleaved with larger packets on the same PPP link, the larger packets may use most of the bandwidth of a low-bandwidth link, to the detriment of the small packets. This can lead to jitter (delay variance), negatively affecting interactive applications such as remote login and VoIP. Configuring a smaller MRU (or MTU) can help mitigate this issue at the cost of higher overhead.

PPP supports a mechanism to exchange link quality reporting information. During option negotiation, a configuration message including a request for a particular quality protocol may be included. Sixteen bits of the option are reserved to specify the particular protocol, but the most common is a PPP standard involving *Link Quality Reports* (LQRs) [RFC1989], using the value 0xC025 in the PPP *Protocol* field. If this is enabled, the peer is asked to provide LQRs at some periodic rate. The maximum time between LQRs requested is encoded as a 32-bit number present in the configuration option and expressed in 1/100s units. Peers may generate LQRs more frequently than requested. LQRs include the following information: a magic number, the number of packets and bytes sent and received, the number of incoming packets with errors and the number of discarded packets, and the total number of LQRs exchanged. A typical implementation allows the user to configure how often LQRs are requested from the peer. Some also provide a way to terminate the link if the quality history fails to meet some configured threshold. LQRs may be requested after the PPP link has reached the Establish state. Each LQR is given a sequence number, so it is possible to determine trends over time, even in the face of reordering of LQRs.

Many PPP implementations support a *callback* capability. In a typical callback setup, a PPP dial-up callback client calls in to a PPP callback server, authentication information is provided, and the server disconnects and calls the client back. This may be useful in situations where call toll charges are asymmetric or for some level of security. The protocol used to negotiate callback is an LCP option with value 0x0D [RFC1570]. If agreed upon, the *Callback Control Protocol* (CBCP) completes the negotiation.

Some compression and encryption algorithms used with PPP require a certain minimum number of bytes, called the *block size*, when operating. When data is not otherwise long enough, padding may be added to cause the length to become an even multiple of the block size. If present, padding is included beyond the data

area and prior to the PPP *FCS* field. A padding method known as *self-describing padding* [RFC1570] alters the value of padding to be nonzero. Instead, each byte gets the value of its offset in the pad area. Thus, the first byte of pad would have the value 0x01, and the final byte contains the number of pad bytes that were added. At most, 255 bytes of padding are supported. The self-describing padding option (type 10) indicates to a peer the ability to understand this form of padding and includes the *maximum pad value* (MPV), which is the largest pad value allowed for this association. Recall that the basic PPP frame lacks an explicit *Length* field, so a receiver can use self-describing padding to determine how many pad bytes should be trimmed from the received data area.

To lessen the impact of the fixed costs of sending a header on every frame, a method has been introduced to multiplex multiple distinct payloads of potentially different protocols into the same PPP frame, an approach called *PPPMux* [RFC3153]. The primary PPP header *Protocol* field is set to *multiplexed frame* (0x0059), and then each payload block is inserted into the frame. This is accomplished by introducing a 1- to 4-byte subframe header in front of each payload block. It includes 1 bit (called PFF) indicating whether a *Protocol* field is included in the subframe header and another 1-bit field (called *LXT*) indicating whether the following *Length* field is 1 or 2 bytes. Beyond this, if present, is the 1- or 2-byte *Protocol ID* using the same values and same compression approach as with the outer PPP header. A 0 value for PFF (meaning no *PID* field is present) is possible when the subframe matches the default PID established when the configuration state is set up using the *PPPMux Control Protocol* (PPPMuxCP).

The PPP frame format in Figure 3-19 indicates that the ordinary PPP/HDLC FCS can be either 16 or 32 bits. While the default is 16, 32-bit FCS values can be enabled with the 32-bit FCS option. Other LCP options include the use of PFC and ACFC, and selection of an authentication algorithm.

Internationalization [RFC2484] provides a way to convey the language and character set to be used. The character set is one of the standard values from the "charset registry" [IANA-CHARSET], and the language value is chosen from the list in [RFC5646][RFC4647].

3.6.2 Multilink PPP (MP)

A special option to PPP called *multilink PPP* (MP) [RFC1990] can be used to aggregate multiple point-to-point links to act as one. This idea is similar to link aggregation, discussed earlier, and has been used for aggregating multiple circuit-switched channels together (e.g., ISDN B channels). MP includes a special LCP option to indicate multilink support as well as a negotiation protocol to fragment and recombine fragmented PPP frames across multiple links. An aggregated link, called a *bundle*, operates as a complete virtual link and can contain its own configuration information. The bundle comprises a number of *member links*. Each member link may also have its own set of options.

The obvious method to implement MP would be to simply alternate packets across the member links. This approach, called the *bank teller's algorithm*, may lead to reordering of packets, which can have undesirable performance impacts on other protocols. (Although TCP/IP, for example, can function properly with reordered packets, it may not function as well as it could without reordering.) Instead, MP places a 2- or 4-byte *sequencing header* in each packet, and the remote MP receiver is tasked with reconstructing the proper order. The data frame appears as shown in Figure 3-25.

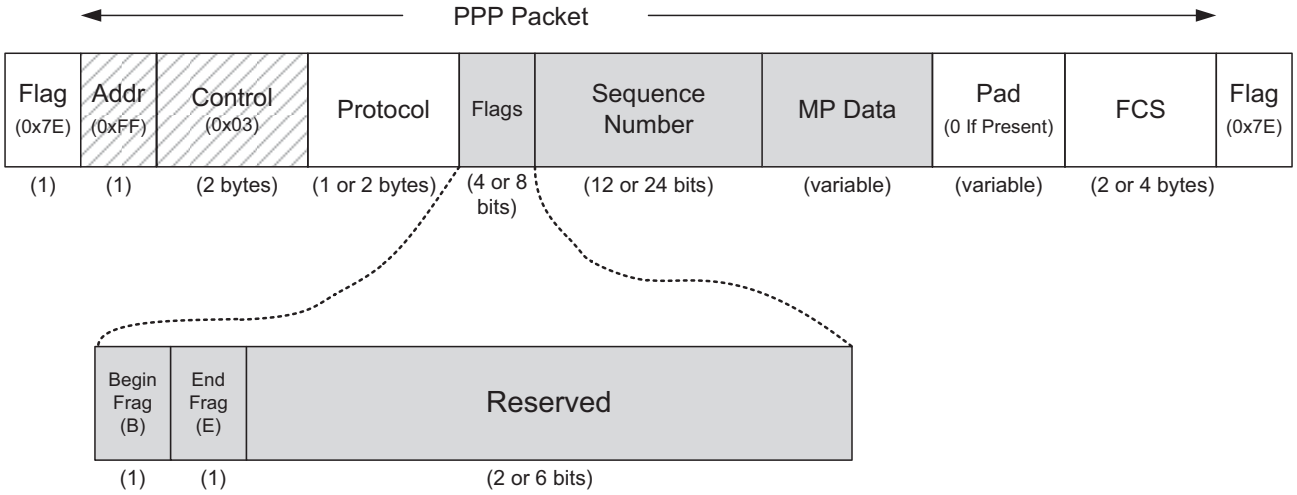

Figure 3-25 An MP fragment contains a sequencing header that allows the remote end of a multilink bundle to reorder fragments. Two formats of this header are supported: a short header (2 bytes) and a long header (4 bytes).

In Figure 3-25 we see an MP fragment with the begin (*B*) and end (*E*) fragment bit fields and *Sequence Number* field. Note that there is both a long format, in which 4 bytes are used for the fragmentation information, and a short format, in which only 2 bytes are used. The format being used is selected during option negotiation using the LCP *short sequence number* option (type 18). If a frame is not fragmented but is carried in this format, both the *B* and *E* bits are set, indicating that the fragment is the first and last (i.e., it is the whole frame). Otherwise, the first fragment has the *BE* bit combination set to 10 and the final fragment has the *BE* bits set to 01, and all fragments in between have them set to 00. The sequence number then gives the packet number offset relative to the first fragment.

Use of MP is requested by including an LCP option called the *multilink maximum received reconstructed unit* (MRRU, type 18) that can act as a sort of larger MRU applying to the bundle. Frames larger than any of the member link MRUs may still be permitted across the MP link, up to the limit advertised in this value.

Because an MP bundle may span multiple member links, a method is needed to identify member links as belonging to the same bundle. Member links in the same bundle are identified by the LCP *endpoint discriminator* option (type 19). The

endpoint discriminator could be a phone number, a number derived from an IP or MAC address, or some administrative string. Other than being common to each member link, there are few restrictions on the form of this option.

The basic method of establishing MP as defined in [RFC1990] expects that member links are going to be used symmetrically—about the same number of fragments will be allocated to each of a fixed number of links. In order to achieve more sophisticated allocations than this, the *Bandwidth Allocation Protocol* (BAP) and *Bandwidth Allocation Control Protocol* (BACP) are specified in [RFC2125]. BAP can be used to dynamically add or remove links from a bundle, and BACP can be used to exchange information regarding how links should be added or removed using BAP. This capability can be used to help implement *bandwidth on demand* (BOD). In networks where some fixed resource needs to be allocated in order to meet an application's need for bandwidth (e.g., by dialing some number of telephone connections), BOD typically involves monitoring traffic and creating new connections when usage is high and shutting down connections when usage is low. This is useful, for example, in cases where some monetary charge is associated with the number of connections being used.

BAP/BACP makes use of a new *link discriminator* LCP option (LCP option type 23). This option contains a 16-bit numeric value that is required to be different for each member link of a bundle. It is used by BAP to identify which links are to be added or removed. BACP is negotiated once per bundle during the network phase of a PPP link. Its main purpose is to identify a *favored peer*. That is, if more than one bundle is being set up simultaneously among multiple peers, the favored peer is preferentially allocated member links.

BAP includes three packet types: request, response, and indication. Requests are to add a link to a bundle or to request the peer to delete a link from a bundle. Indications convey the results of attempted additions back to the original requester and are acknowledged. Responses are either ACKs or NACKs for these requests. More details can be found in [RFC2125].

3.6.3 Compression Control Protocol (CCP)

Historically, PPP has been the protocol of choice when using relatively slow dial-up modems. As a consequence, a number of methods have been developed to compress data sent over PPP links. This type of compression is distinct both from the types of compression supported in modem hardware (e.g., V.42bis, V.44) and also from protocol *header compression*, which we discuss later. Today, several compression options are available. To choose among them for each direction on a PPP link, LCP can negotiate an option to enable the *Compression Control Protocol* (CCP) [RFC1962]. CCP acts like an NCP (see Section 3.6.5) but handles the details of configuring compression once the compression option is indicated in the LCP link establishment exchange.

In behaving like an NCP, CCP can be negotiated only once the link has entered the Network state. It uses the same packet exchange procedures and formats as

LCP, except the *Protocol* field is set to 0x80FD, there are some special options, and in addition to the common *Code* field values (1–7) two new operations are defined: reset-request (0x0e) and reset-ACK (0x0f). If an error is detected in a compressed frame, a reset request can be used to cause the peer to reset compression state (e.g., dictionaries, state variables, state machines, etc.). After resetting, the peer responds with a reset-ACK.

One or more compressed packets may be carried within the information portion of a PPP frame (i.e., the portion including the LCP data and possibly pad portions). Compressed frames carry the *Protocol* field value of 0x00FD, but the mechanism used to indicate the presence of multiple compressed datagrams is dependent on the particular compression algorithm used (see Section 3.6.6). When used in conjunction with MP, CCP may be used either on the bundle or on some combination of the member links. If used only on member links, the *Protocol* field is set to 0x00FB (individual link compressed datagram).

CCP can enable one of about a dozen compression algorithms [PPPn]. Most of the algorithms are not official standards-track IETF documents, although they may be described in informational RFCs (e.g., [RFC1977] describes the BSD compression scheme, and [RFC2118] describes the *Microsoft Point-to-Point Compression Protocol* (MPPC)). If compression is being used, PPP frames are reconstructed before further processing, so higher-layer PPP operations are not generally concerned with the details of the compressed frames.

3.6.4 PPP Authentication

Before a PPP link becomes operational in the Network state, it is often necessary to establish the identity of the peer(s) of the link using some *authentication* (identity verification) mechanism. The basic PPP specification has a default of no authentication, so the authentication exchange of Figure 3-24 would not be used in such cases. More often, however, some form of authentication is required, and a number of protocols have evolved over the years to deal with this situation. In this chapter we discuss them only from a high-level point of view and leave the details for the chapter on security (Chapter 18). Other than no authentication, the simplest and least secure authentication scheme is called the *Password Authentication Protocol* (PAP). This protocol is very simple—one peer requests the other to send a password, and the password is so provided. As the password is sent unencrypted over the PPP link, any eavesdropper on the line can simply capture the password and use it later. Because of this significant vulnerability, PAP is not recommended for authentication. PAP packets are encoded as LCP packets with the *Protocol* field value set to 0xC023.

A somewhat more secure approach to authentication is provided by the *Challenge-Handshake Authentication Protocol* (CHAP) [RFC1994]. Using CHAP, a random value is sent from one peer (called the authenticator) to the other. A response is formed by using a special *one-way* (i.e., not easily invertible) function to combine the random value with a shared secret key (usually derived from a password)

to produce a number that is sent in response. Upon receiving this response, the authenticator can determine with a very high degree of confidence that its peer possesses the correct secret key. This protocol never sends the key or password over the link in a clear (unencrypted) form, so any eavesdropper is unable to learn the secret. Because a different random value is used each time, the result of the function changes for each challenge/response, so the values an eavesdropper may be able to capture cannot be reused (played back) to impersonate the peer. However, CHAP is vulnerable to a "man in the middle" form of attack (see Chapter 18).

EAP [RFC3748] is an authentication framework available for many different network types. It also supports many (about 40) different authentication methods, ranging from simple passwords such as PAP and CHAP to more elaborate types of authentication (e.g., smart cards, biometrics). EAP defines a message format for carrying a variety of specific types of authentication formats, but additional specifications are needed to define how EAP messages are carried over particular types of links.

When EAP is used with PPP, the basic authentication method discussed so far is altered. Instead of negotiating a specific authentication method early in the link establishment (at LCP link establishment), the authentication operation may be postponed until the Auth state (just before the Network state). This allows for a greater richness in the types of information that can be used to influence access control decisions by *remote access servers* (RASs). When there is a standard protocol for carrying a variety of authentication mechanisms, a network access server may not need to process the contents of EAP messages at all but can instead depend on some other infrastructure authentication server (e.g., a RADIUS server [RFC2865]) to determine access control decisions. This is currently the design of choice for enterprise networks and ISPs.

3.6.5 Network Control Protocols (NCPs)

Although many different NCPs can be used on a PPP link (even simultaneously), we shall focus on the NCPs supporting IPv4 and IPv6. For IPv4, the NCP is called the *IP Control Protocol* (IPCP) [RFC1332]. For IPv6, the NCP is *IPV6CP* [RFC5072]. Once LCP has completed its link establishment and authentication, each end of the link is in the Network state and may proceed to negotiate a network-layer association using zero or more NCPs (one, such as IPCP, is typical).

IPCP, the standard NCP for IPv4, can be used to establish IPv4 connectivity over a link and configure *Van Jacobson header compression* (VJ compression) [RFC1144]. IPCP packets may be exchanged after the PPP state machine has reached the Network state. IPCP packets use the same packet exchange mechanism and packet format as LCP, except the *Protocol* field is set to 0x8021, and the *Code* field is limited to the range 0–7. These values of the *Code* field correspond to the message types: vendor-specific (see [RFC2153]), configure-request, configure-ACK, configure-REJECT, terminate-request, terminate-ACK, and code-REJECT. IPCP can negotiate a number of options, including an IP compression protocol (2), the IPv4 address

(3), and Mobile IPv4 [RFC2290] (4). Other options are available for learning the location of primary and secondary domain name servers (see Chapter 11).

IPV6CP uses the same packet exchange and format as LCP, except it has two different options: interface-identifier and IPv6-compression-protocol. The interface identifier option is used to convey a 64-bit IID value (see Chapter 2) used as the basis for forming a link-local IPv6 address. Because it is used only on the local link, it does not require global uniqueness. This is accomplished using a standard link-local prefix for the higher-order bits of the IPv6 address and allowing the lower-order bits to be a function of the interface identifier. This mimics IPv6 autoconfiguration (see Chapter 6).

3.6.6 Header Compression

PPP dial-up lines have historically been comparatively slow (54,000 bits/s or less), and many small packets are often used with TCP/IP (e.g., for TCP's acknowledgments; see Chapter 15). Most of these packets contain a TCP and IP header that changes little from one packet to another on the same TCP connection. Other higher-layer protocols behave similarly. Thus, it is useful to have a way of compressing the headers of these higher-layer protocols (or eliminating them) so that fewer bytes need to be carried over relatively slow point-to-point links. The methods employed to compress or eliminate headers have evolved over time. We discuss them in chronological order, beginning with VJ compression, mentioned earlier.

In VJ compression, portions of the higher-layer (TCP and IP) headers are replaced with a small, 1-byte connection identifier. [RFC1144] discusses the origin of this approach, using an older point-to-point protocol called CSLIP (Compressed Serial Line IP). A typical IPv4 header is 20 bytes, and a TCP header without options is another 20. Together, a common combined TCP/IPv4 header is thus 40 bytes, and many of the fields do not change from packet to packet. Furthermore, many of the fields that do change from packet to packet change only slightly or in a limited way. When the nonchanging values are sent over a link once (or a small number of times) and kept in a table, a small index can be used as a replacement for the constants in subsequent packets. The limited changing values are then encoded differentially (i.e., only the amount of change is sent). As a result, the entire 40-byte header can usually be compressed to an effective 3 or 4 bytes. This can significantly improve TCP/IP performance over slow links.

The next step in the evolution of header compression is simply called *IP header compression* [RFC2507][RFC3544]. It provides a way to compress the headers of multiple packets using both TCP or UDP transport-layer protocols and either IPv4 or IPv6 network-layer protocols. The techniques are a logical extension and generalization of the VJ compression technique that applies to more protocols, and to links other than PPP links. [RFC2507] points out the necessity of some strong error detection mechanism in the underlying link layer because erroneous packets can be constructed at the egress of a link if compressed header values are damaged in transit. This is important to recognize when header compression is used on links that may not have as strong an FCS computation as PPP.

The most recent step in the evolution of header compression is known as *Robust Header Compression* (ROHC) [RFC5225]. It further generalizes IP header compression to cover more transport protocols and allows more than one form of header compression to operate simultaneously. Like the IP header compression mentioned previously, it can be used over various types of links, including PPP.

3.6.7 Example

We now look at the debugging output of a PPP server interacting with a client over a dial-in modem. The dialing-in client is an IPv6-capable Microsoft Windows Vista machine, and the server is Linux. The Vista machine is configured to negotiate multilink capability even on single links (Properties | Options | PPP Settings), for demonstration purposes, and the server is configured to require an encryption protocol negotiated using CCP (see `MPPE` in the following listing):

```
data dev=ttyS0, pid=28280, caller='none', conn='38400',
     name='',cmd='/usr/sbin/pppd', user='/AutoPPP/'
pppd 2.4.4 started by a_ppp, uid 0
using channel 54
Using interface ppp0
ppp0 <--> /dev/ttyS0
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth eap>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfNak id=0x1 <auth chap MS-v2>]
sent [LCP ConfReq id=0x2 <asyncmap 0x0> <auth chap MS-v2>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfAck id=0x2 <asyncmap 0x0> <auth chap MS-v2>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfReq id=0x2 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp><callback CBCP> <mrru 1614>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84.f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [LCP ConfRej id=0x2 <callback CBCP> <mrru 1614>]
rcvd [LCP ConfReq id=0x3 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84.f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [LCP ConfAck id=0x3 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84.f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [CHAP Challenge id=0x1a <4d53c52b8e7dcfe7a9ea438b2b4daf55>,
      name = "dialer"]
rcvd [LCP Ident id=0x4 magic=0xa531e06 "MSRASV5.20"]
rcvd [LCP Ident id=0x5 magic=0xa531e06 "MSRAS-0-VISTA"]
rcvd [CHAP Response id=0x1a
     <4b5dc95ed4e1788b959025de0233d4fc0000000
      00000000033a555d2a77bd1fa692f2a0af707cd 4f0c0072c379c82e0f00>,
     name = "dialer"]
sent [CHAP Success id=0x1a
     "S=7E0B6B513215C87520BEF6725EF8A9945C28E918M=Access granted"]
```

```
sent [CCP ConfReq id=0x1 <mppe +H -M +S +L -D -C>]
rcvd [IPV6CP ConfReq id=0x6 <addr fe80::0000:0000:dead:beef>]
sent [IPV6CP TermAck id=0x6]
rcvd [CCP ConfReq id=0x7 <mppe -H -M -S -L -D +C>]
sent [CCP ConfNak id=0x7 <mppe +H -M +S +L -D -C>]
rcvd [IPCP ConfReq id=0x8 <compress VJ 0f 01> <addr 0.0.0.0>
      <ms-dns1 0.0.0.0> <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0>
      <ms-wins 0.0.0.0>]
sent [IPCP TermAck id=0x8]
rcvd [CCP ConfNak id=0x1 <mppe -H -M +S -L -D -C>]
sent [CCP ConfReq id=0x2 <mppe -H -M +S -L -D -C>]
rcvd [CCP ConfReq id=0x9 <mppe -H -M +S -L -D -C>]
sent [CCP ConfAck id=0x9 <mppe -H -M +S -L -D -C>]
rcvd [CCP ConfAck id=0x2 <mppe -H -M +S -L -D -C>]
MPPE 128-bit stateful compression enabled
sent [IPCP ConfReq id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
sent [IPV6CP ConfReq id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
rcvd [IPCP ConfAck id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
rcvd [IPCP ConfReq id=0xa <compress VJ 0f 01>
      <addr 0.0.0.0> <ms-dns1 0.0.0.0>
      <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
sent [IPCP ConfRej id=0xa <ms-wins 0.0.0.0> <ms-wins 0.0.0.0>]
rcvd [IPV6CP ConfReq id=0xb <addr fe80::0000:0000:dead:beef>]
sent [IPV6CP ConfAck id=0xb <addr fe80::0000:0000:dead:beef>]
rcvd [IPCP ConfAck id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
local LL address fe80::0206:5bff:fedd:c5c3
remote LL address fe80::0000:0000:dead:beef
rcvd [IPCP ConfReq id=0xc <compress VJ 0f 01>
     <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns3 0.0.0.0>]
sent [IPCP ConfNak id=0xc <addr 192.168.0.2> <ms-dns1 192.168.0.1>
      <ms-dns3 192.168.0.1>]
sent [IPCP ConfAck id=0xd <compress VJ 0f 01> <addr 192.168.0.2>
      <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
local IP address 192.168.0.1
remote IP address 192.168.0.2
... data ...
```

Here we can see a somewhat involved PPP exchange, as viewed from the server. The PPP server process creates a (virtual) network interface called `ppp0`, which is awaiting an incoming connection on the dial-up modem attached to serial port `ttyS0`. Once the incoming connection arrives, the server requests an asyncmap of 0x0, EAP authentication, PFC, and ACFC. The client refuses EAP authentication and instead suggests MS-CHAP-v2 (`ConfNak`) [RFC2759]. The server then tries again, this time using MS-CHAP-v2, which is then accepted and acknowledged (`ConfAck`). Next, the incoming request includes CBCP; an MRRU of 1614 bytes, which is associated with MP support; and an endpoint ID. The server rejects the request for CBCP and multilink operation (`ConfRej`). The endpoint discriminator is once again sent by the client, this time without the MRRU, and is

accepted and acknowledged. Next, the server sends a CHAP challenge with the name `dialer`. Before a response to the challenge arrives, two incoming identity messages arrive, indicating that the peer is identified by the strings `MSRASV5.20` and `MSRAS-0-VISTA`. Finally, the CHAP response arrives and is validated as correct, and an acknowledgment indicates that access is granted. PPP then moves on to the Network state.

Once in the Network state, the CCP, IPCP, and IPV6CP NCPs are exchanged. CCP attempts to negotiate *Microsoft Point-to-Point Encryption* (MPPE) [RFC3078]. MPPE is somewhat of an anomaly, as it is really an encryption protocol, and rather than compressing the packet it actually expands it by 4 bytes. It does, however, provide a relatively simple means of establishing encryption early in the negotiation process. The options `+H -M +S +L -D -C` indicate whether MPPE stateless operation is desired (`H`), what cryptographic key strength is available (secure, `S`; medium, `M`; or low, `L`), an obsolete `D` bit, and whether a separate, proprietary compression protocol called MPPC [RFC2118] is desired (`C`). Eventually the two peers agree on stateful mode using strong 128-bit keying (`-H`, `+S`). Note that during the middle of this negotiation, the client attempts to send an IPCP request, but the server responds with an unsolicited `TermAck` (a message defined within LCP that ICPC adopts). This is used to indicate to the peer that the server is "in need of renegotiation" [RFC1661].

After the successful negotiation of MPPE, the server requests the use of VJ header compression and provides its IPv4 and IPv6 addresses, `192.168.0.1` and `fe80::0206:5bff:fedd:c5c3`. This IPv6 address is derived from the server's Ethernet MAC address 00:06:5B:DD:C5:C3. The client initially suggests its IPv4 address and name servers to be `0.0.0.0` using IPCP, but this is rejected. The client then requests to use `fe80::0000:0000:dead:beef` as its IPv6 address, which is accepted and acknowledged. Finally, the client ACKs both the IPv4 and IPv6 addresses of the server, and the IPv6 addresses have been established. Next, the client again requests IPv4 and server addresses of `0.0.0.0`, which is rejected in favor of `192.168.0.1`. These are accepted and acknowledged.

As we can see from this exchange, the PPP negotiation is both flexible and tedious. There are many options that can be attempted, rejected, and renegotiated. While this may not be a big problem on a link with low delay, imagine how long this exchange could take if each message took a few seconds (or longer) to reach its destination, as might occur over a satellite link, for example. Link establishment would be a visibly long procedure for the user.

3.7 Loopback

Although it may seem surprising, in many cases clients may wish to communicate with servers on the same computer using Internet protocols such as TCP/IP. To enable this, most implementations support a network-layer *loopback* capability that typically takes the form of a virtual loopback network interface. It acts like a real

network interface but is really a special piece of software provided by the operating system to enable TCP/IP and other communications on the same host computer. IPv4 addresses starting with 127 are reserved for this, as is the IPv6 address ::1 (see Chapter 2 for IPv4 and IPv6 addressing conventions). Traditionally, UNIX-like systems including Linux assign the IPv4 address of 127.0.0.1 (::1 for IPv6) to the loopback interface and assign it the name `localhost`. An IP datagram sent to the loopback interface must not appear on any network. Although we could imagine the transport layer detecting that the other end is a loopback address and short-circuiting some of the transport-layer logic and all of the network-layer logic, most implementations perform complete processing of the data in the transport layer and network layer and loop the IP datagram back up in the network stack only when the datagram leaves the bottom of the network layer. This can be useful for performance measurement, for example, because the amount of time required to execute the stack software can be measured without any hardware overheads. In Linux, the loopback interface is called `lo`.

```
Linux% ifconfig lo
lo Link encap:Local Loopback
           inet addr:127.0.0.1 Mask:255.0.0.0
           inet6 addr: ::1/128 Scope:Host
           UP LOOPBACK RUNNING MTU:16436 Metric:1
           RX packets:458511 errors:0 dropped:0 overruns:0 frame:0
           TX packets:458511 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:266049199 (253.7 MiB)
           TX bytes:266049199 (253.7 MiB)
```

Here we see that the local loopback interface has the IPv4 address `127.0.0.1` and a subnet mask of `255.0.0.0` (corresponding to class A network number 127 in classful addressing). The IPv6 address `::1` has a 128-bit-long prefix, so it represents only a single address. The interface has an MTU of 16KB (this can be configured to a much larger size, up to 2GB). A significant amount of traffic, nearly half a million packets, has passed through the interface without error since the machine was initialized two months earlier. We would not expect to see errors on the local loopback device, given that it never really sends packets on any network.

In Windows, the Microsoft Loopback Adapter is not installed by default, even though IP loopback is still supported. This adapter can be used for testing various network configurations even when a physical network interface is not available. To install it under Windows XP, select Start | Control Panel | Add Hardware | Select Network Adapters from list | Select Microsoft as manufacturer | Select Microsoft Loopback Adapter. For Windows Vista or Windows 7, run the program `hdwwiz` from the command prompt and add the Microsoft Loopback Adapter manually. Once this is performed, the `ipconfig` command reveals the following (this example is from Windows Vista):

```
C:\> ipconfig /all
...
Ethernet adapter Local Area Connection 2:
   Connection-specific DNS Suffix . :
   Description . . . . . . . . . . . : Microsoft Loopback Adapter
   Physical Address. . . . . . . . . : 02-00-4C-4F-4F-50
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . :
           fe80::9c0d:77a:52b8:39f0%18(Preferred)
   Autoconfiguration IPv4 Address. . : 169.254.57.240(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 302121036
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
           fec0:0:0:ffff::2%1
           fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Here we can see that the interface has been created, has been assigned both IPv4 and IPv6 addresses, and appears as a sort of virtual Ethernet device. Now the machine has several loopback addresses:

```
C:\> ping 127.1.2.3
Pinging 127.1.2.3 with 32 bytes of data:
Reply from 127.1.2.3: bytes=32 time<1ms TTL=128
Reply from 127.1.2.3: bytes=32 time<1ms TTL=128
Reply from 127.1.2.3: bytes=32 time<1ms TTL=128
Reply from 127.1.2.3: bytes=32 time<1ms TTL=128
Ping statistics for 127.1.2.3:
       Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
       Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\> ping ::1
Pinging ::1 from ::1 with 32 bytes of data:
Reply from ::1: time<1ms
Reply from ::1: time<1ms
Reply from ::1: time<1ms
Reply from ::1: time<1ms
Ping statistics for ::1:
      Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:

      Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\> ping 169.254.57.240
Pinging 169.254.57.240127.1.2.3 with 32 bytes of data:
Reply from 169.254.57.240: bytes=32 time<1ms TTL=128
Reply from 169.254.57.240: bytes=32 time<1ms TTL=128
Reply from 169.254.57.240: bytes=32 time<1ms TTL=128
```

```
Reply from 169.254.57.240: bytes=32 time<1ms TTL=128
Ping statistics for 169.254.57.240:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
      Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Here we can see that in IPv4, any destination address starting with `127` is looped back. For IPv6, however, only the single address `::1` is defined for loopback operation. We can also see how the loopback adapter with address `169.254.57.240` returned data immediately. One subtlety to which we will return in Chapter 9 is whether multicast or broadcast datagrams should be copied back to the sending computer (over the loopback interface). This choice can be made by each individual application.

3.8 MTU and Path MTU

As we can see from Figure 3-3, there is a limit on the size of the frame available for carrying the PDUs of higher-layer protocols in many link-layer networks such as Ethernet. This usually limits the number of payload bytes to about 1500 for Ethernet and often the same amount for PPP in order to maintain compatibility with Ethernet. This characteristic of the link layer is called the *maximum transmission unit* (MTU). Most packet networks (like Ethernet) have a fixed upper limit. Most stream-type networks (serial links) have a configurable limit that is then used by framing protocols such as PPP. If IP has a datagram to send, and the datagram is larger than the link layer's MTU, IP performs *fragmentation*, breaking the datagram up into smaller pieces (fragments), so that each fragment is smaller than the MTU. We discuss IP fragmentation in Chapters 5 and 10.

When two hosts on the same network are communicating with each other, it is the MTU of the local link interconnecting them that has a direct effect on the size of datagrams that are used during the conversation. When two hosts communicate across multiple networks, each link can have a different MTU. The minimum MTU across the network path comprising all of the links is called the *path MTU*.

The path MTU between any two hosts need not be constant over time. It depends on the path being used at any time, which can change if the routers or links in the network fail. Also, paths are often not *symmetric* (i.e., the path from host A to B may not be the reverse of the path from B to A); hence the path MTU need not be the same in the two directions.

[RFC1191] specifies the *path MTU discovery* (PMTUD) mechanism for IPv4, and [RFC1981] describes it for IPv6. A complementary approach that avoids some of the issues with these mechanisms is described in [RFC4821]. PMTU discovery is used to determine the path MTU at a point in time and is required of IPv6 implementations. In later chapters we shall see how this mechanism operates after we have described ICMP and IP fragmentation. We shall also see what effect it can have on transport performance when we discuss TCP and UDP.

3.9 Tunneling Basics

In some cases it is useful to establish a virtual link between one computer and another across the Internet or other network. VPNs, for example, offer this type of service. The method most commonly used to implement these types of services is called *tunneling*. Tunneling, generally speaking, is the idea of carrying lower-layer traffic in higher-layer (or equal-layer) packets. For example, IPv4 can be carried in an IPv4 or IPv6 packet; Ethernet can be carried in a UDP or IPv4 or IPv6 packet, and so on. Tunneling turns the idea of strict layering of protocols on its head and allows for the formation of *overlay networks* (i.e., networks where the "links" are really virtual links implemented in some other protocol instead of physical connections). It is a very powerful and useful technique. Here we discuss the basics of some of the tunneling options.

There is a great variety of methods for tunneling packets of one protocol and/or layer over another. Three of the more common protocols used to establish tunnels include *Generic Routing Encapsulation* (GRE) [RFC2784], the Microsoft proprietary *Point-to-Point Tunneling Protocol* (PPTP) [RFC2637], and the *Layer 2 Tunneling Protocol* (L2TP) [RFC3931]. Others include the earlier nonstandard IP-in-IP tunneling protocol [RFC1853]. GRE and LT2P were developed to standardize and replace IP-in-IP and PPTP, respectively, but all of these approaches are still in use. We shall focus on GRE and PPTP, with more emphasis on PPTP, as it is more visible to individual users even though it is not an IETF standard. L2TP is often used with security at the IP layer (IPsec; see Chapter 18) because L2TP by itself does not provide security. Because GRE and PPTP are closely related, we now look at the GRE header in Figure 3-26, in both its original standard and revised standard forms.

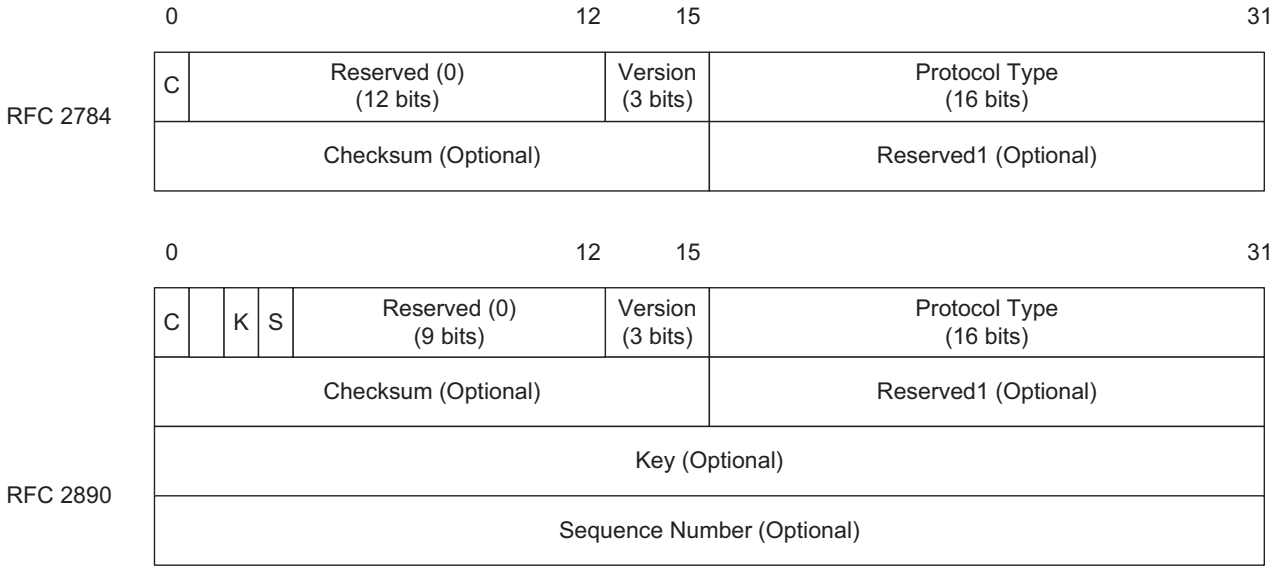

Figure 3-26 The basic GRE header is only 4 bytes but includes the option of a 16-bit checksum (of a type common to many Internet protocols). The header was later extended to include an identifier (*Key* field) common to multiple packets in a flow, and a *Sequence Number*, to help in resequencing packets that get out of order.

As can be seen from the headers in Figure 3-26, the baseline GRE specification [RFC2784] is rather simple and provides only a minimal encapsulation for other packets. The first bit field (*C*) indicates whether a checksum is present. If it is, the *Checksum* field contains the same type of checksum found in many Internet-related protocols (see Section 5.2.2). If the *Checksum* field is present, the *Reserved1* field is also present and is set to 0. [RFC2890] extends the basic format to include optional *Key* and *Sequence Number* fields, present if the *K* and *S* bit fields from Figure 3-26 are set to 1, respectively. If present, the *Key* field is arranged to be a common value in multiple packets, indicating that they belong to the same flow of packets. The *Sequence Number* field is used in order to reorder packets if they should become out of sequence (e.g., by going through different links).

Although GRE forms the basis for and is used by PPTP, the two protocols serve somewhat different purposes. GRE tunnels are typically used within the network infrastructure to carry traffic between ISPs or within an enterprise intranet to serve branch offices and are not necessarily encrypted, although GRE tunnels can be combined with IPsec. PPTP, conversely, is most often used between users and their ISPs or corporate intranets and is encrypted (e.g., using MPPE). PPTP essentially combines GRE with PPP, so GRE can provide the virtual point-to-point link upon which PPP operates. GRE carries its traffic using IPv4 or IPv6 and as such is a layer 3 tunneling technology. PPTP is more often used to carry layer 2 frames (such as Ethernet) so as to emulate a direct LAN (link-layer) connection. This can be used for remote access to corporate networks, for example. PPTP uses a non-standard variation on the standard GRE header (see Figure 3-27).

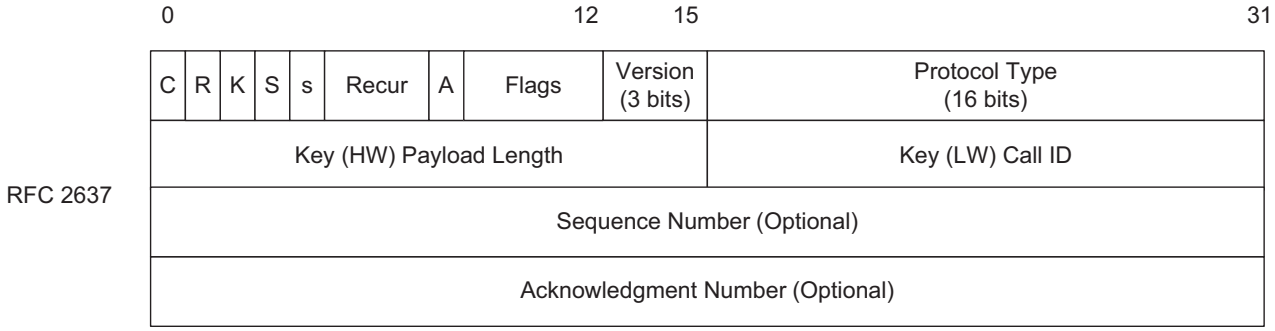

Figure 3-27 The PPTP header is based on an older, nonstandard GRE header. It includes a sequence number, a cumulative packet acknowledgment number, and some identification information. Most of the fields in the first word are set to 0.

We can see a number of differences in Figure 3-27 from the standard GRE header, including the extra *R*, *s*, and *A* bit fields, additional *Flags* field, and *Recur* field. Most of these are simply set to 0 and not used (their assignment is based on an older, nonstandard version of GRE). The *K*, *S*, and *A* bit fields indicate that the *Key*, *Sequence Number*, and *Acknowledgment Number* fields are present. If present, the value of the *Sequence Number* field holds the largest packet number seen by the peer.

We now turn to the establishment of a PPTP session. We shall conclude later with a brief discussion of some of PPTP's other capabilities. The following example is similar to the PPP link establishment example given earlier, except now instead of using a dial-up link, PPTP is providing the "raw" link to PPP. Once again, the client is Windows Vista, and the server is Linux. This output comes from the `/var/log/messages` file when the debug option is enabled:

```
pptpd: MGR: Manager process started
pptpd: MGR: Maximum of 100 connections available
pptpd: MGR: Launching /usr/sbin/pptpctrl to handle client
pptpd: CTRL: local address = 192.168.0.1
pptpd: CTRL: remote address = 192.168.1.1
pptpd: CTRL: pppd options file = /etc/ppp/options.pptpd
pptpd: CTRL: Client 71.141.227.30 control connection started
pptpd: CTRL: Received PPTP Control Message (type: 1)
pptpd: CTRL: Made a START CTRL CONN RPLY packet
pptpd: CTRL: I wrote 156 bytes to the client.
pptpd: CTRL: Sent packet to client
pptpd: CTRL: Received PPTP Control Message (type: 7)

pptpd: CTRL: Set parameters to 100000000 maxbps, 64 window size
pptpd: CTRL: Made a OUT CALL RPLY packet
pptpd: CTRL: Starting call (launching pppd, opening GRE)
pptpd: CTRL: pty_fd = 6
pptpd: CTRL: tty_fd = 7
pptpd: CTRL (PPPD Launcher): program binary = /usr/sbin/pppd
pptpd: CTRL (PPPD Launcher): local address = 192.168.0.1
pptpd: CTRL (PPPD Launcher): remote address = 192.168.1.1
pppd: pppd 2.4.4 started by root, uid 0
pppd: using channel 60
pptpd: CTRL: I wrote 32 bytes to the client.
pptpd: CTRL: Sent packet to client
pppd: Using interface ppp0
pppd: Connect: ppp0 <--> /dev/pts/1
pppd: sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth chap MS-v2>
            <magic 0x4e2ca200> <pcomp> <accomp>]
pptpd: CTRL: Received PPTP Control Message (type: 15)
pptpd: CTRL: Got a SET LINK INFO packet with standard ACCMs
pptpd: GRE: accepting packet #0
pppd: rcvd [LCP ConfReq id=0x0 <mru 1400> <magic 0x5e565505>
            <pcomp> <accomp>]
pppd: sent [LCP ConfAck id=0x0 <mru 1400> <magic 0x5e565505>
            <pcomp> <accomp>]
pppd: sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth chap MS-v2>
            <magic 0x4e2ca200> <pcomp> <accomp>]
pptpd: GRE: accepting packet #1
pppd: rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <auth chap MS-v2>
            <magic 0x4e2ca200> <pcomp> <accomp>]
pppd: sent [CHAP Challenge id=0x3
            <eb88bfff67d1c239ef73e98ca32646a5>, name = "dialer"]
pptpd: CTRL: Received PPTP Control Message (type: 15)
pptpd: CTRL: Ignored a SET LINK INFO packet with real ACCMs!
```

```
pptpd: GRE: accepting packet #2
pppd: rcvd [CHAP Response id=0x3<276f3678f0f03fa57f64b3c367529565000000
            00000000000fa2b2ae0ad8db9d986f8e222a0217a620638a24
            3179160900>, name = "dialer"]
pppd: sent [CHAP Success id=0x3
            "S=C551119E0E1AAB68E86DED09A32D0346D7002E05
            M=Accessgranted"]
pppd: sent [CCP ConfReq id=0x1 <mppe +H -M +S +L -D -C>]
pptpd: GRE: accepting packet #3
pppd: rcvd [IPV6CP ConfReq id=0x1 <addr fe80::1cfc:fddd:8e2c:e118>]
pppd: sent [IPV6CP TermAck id=0x1]
pptpd: GRE: accepting packet #4
pppd: rcvd [CCP ConfReq id=0x2 <mppe +H -M -S -L -D -C>]
pppd: sent [CCP ConfNak id=0x2 <mppe +H -M +S +L -D -C>]
pptpd: GRE: accepting packet #5
pptpd: GRE: accepting packet #6
pppd: rcvd [IPCP ConfReq id=0x3 <addr 0.0.0.0> <ms-dns1 0.0.0.0>
            <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
pptpd: GRE: accepting packet #7
pppd: sent [IPCP TermAck id=0x3]
pppd: rcvd [CCP ConfNak id=0x1 <mppe +H -M +S -L -D -C>]
pppd: sent [CCP ConfReq id=0x2 <mppe +H -M +S -L -D -C>]
pppd: rcvd [CCP ConfReq id=0x4 <mppe +H -M +S -L -D -C>]
pppd: sent [CCP ConfAck id=0x4 <mppe +H -M +S -L -D -C>]
pptpd: GRE: accepting packet #8
pppd: rcvd [CCP ConfAck id=0x2 <mppe +H -M +S -L -D -C>]
pppd: MPPE 128-bit stateless compression enabled
pppd: sent [IPCP ConfReq id=0x1 <addr 192.168.0.1>]
pppd: sent [IPV6CP ConfReq id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
pptpd: GRE: accepting packet #9
pppd: rcvd [IPCP ConfAck id=0x1 <addr 192.168.0.1>]
pptpd: GRE: accepting packet #10
pppd: rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
pptpd: GRE: accepting packet #11
pppd: rcvd [IPCP ConfReq id=0x5 <addr 0.0.0.0>
            <ms-dns1 0.0.0.0> <ms-wins 0.0.0.0>
            <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
pppd: sent [IPCP ConfRej id=0x5 <ms-wins 0.0.0.0> <ms-wins 0.0.0.0>]
pptpd: GRE: accepting packet #12
pppd: rcvd [IPV6CP ConfReq id=0x6 <addr fe80::1cfc:fddd:8e2c:e118>]
pppd: sent [IPV6CP ConfAck id=0x6 <addr fe80::1cfc:fddd:8e2c:e118>]
pppd: local LL address fe80::0206:5bff:fedd:c5c3
pppd: remote LL address fe80::1cfc:fddd:8e2c:e118
pptpd: GRE: accepting packet #13
pppd: rcvd [IPCP ConfReq id=0x7 <addr 0.0.0.0>
            <ms-dns1 0.0.0.0> <ms-dns3 0.0.0.0>]
pppd: sent [IPCP ConfNak id=0x7 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
pptpd: GRE: accepting packet #14
pppd: rcvd [IPCP ConfReq id=0x8 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
pppd: sent [IPCP ConfAck id=0x8 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
```

```
pppd: local IP address 192.168.0.1
pppd: remote IP address 192.168.1.1
pptpd: GRE: accepting packet #15
pptpd: CTRL: Sending ECHO REQ id 1
pptpd: CTRL: Made a ECHO REQ packet
pptpd: CTRL: I wrote 16 bytes to the client.
pptpd: CTRL: Sent packet to client
```

This output looks similar to the PPP example we examined earlier, except this one has output from both the `pppd` process as well as a `pptpd` process. These processes work together to establish PPTP sessions at the server. The setup begins with `pptpd` receiving a type 1 control message, indicating that the client wishes to establish a control connection. PPTP uses a separate control and data stream, so first the control stream is set up. After responding to this request, the server receives a type 7 control message indicating an outgoing call request from the peer. The maximum speed (in bits per second) is set to a large value of 100,000,000, which effectively means it is unbounded. The *window* is set to 64, a concept we typically encounter in transport protocols such as TCP (see Chapter 15). Here the window is used for flow control. That is, PPTP uses its sequence numbers and acknowledgment numbers to determine how many frames reach the destination successfully. If too few frames are successfully delivered, the sender slows down. To determine the amount of time to wait for an acknowledgment for frames it sends, PPTP uses an adaptive timeout mechanism based on estimating the round-trip time of the link. We shall see this type of calculation again when we study TCP.

Soon after the window is set, the `pppd` application begins to run and process the PPP data as we saw before in the dial-up example. The only real difference between the two is that `pptpd` relays packets to the `pppd` process as they arrive and depart, and a few special PPTP messages (such as `set link info` and `echo request`) are processed by `pptpd` itself. This example illustrates how the PPTP protocol really acts as a GRE tunneling agent for PPP packets. This is convenient because an existing PPP implementation (here, `pppd`) can be used as is to process the encapsulated PPP packets. Note that while GRE is itself ordinarily encapsulated in IPv4 packets, similar functionality is available using IPv6 to tunnel packets [RFC2473].

3.9.1 Unidirectional Links

An interesting issue arises when the link to be used operates in only one direction. Such links are called *unidirectional links* (UDLs), and many of the protocols described so far do not operate properly in such circumstances because they require exchanges of information (e.g., PPP's configuration messages). To deal with this situation, a standard has been created whereby tunneling over a second Internet interface can be combined with operation of the UDL [RFC3077]. The typical situation where this arises is an Internet connection that uses a satellite for downstream traffic (headed to the user) and a dial-up modem link for upstream

traffic. This setup can be useful in cases where the satellite-connected user's usage is dominated by downloading as opposed to uploading and was commonly used in early satellite Internet installations. It operates by encapsulating link-layer upstream traffic in IP packets using a GRE encapsulation.

To establish and maintain tunnels automatically at the receiver, [RFC3077] specifies a *Dynamic Tunnel Configuration Protocol* (DTCP). DTCP involves sending multicast *Hello messages* on the downlink so that any interested receiver can learn about the existence of the UDL and its MAC and IP addresses. In addition, Hello messages indicate a list of tunnel endpoints within the network that can be reached by the user's secondary interface. After the user selects which tunnel endpoint to use, DTCP arranges for return traffic to be encapsulated with the same MAC type as the UDL in GRE tunnels. The service provider arranges to receive these GRE-encapsulated layer 2 frames (frequently Ethernet), extract them from the tunnel, and forward them appropriately. Thus, although the upstream side of the UDLs (provider's side) requires manual tunnel configuration, the downstream side, which includes many more users, has automatically configured tunnels. Note that this approach to handling UDLs essentially "hides" the link asymmetry from the upper-layer protocols. As a consequence, the performance (latency, bandwidth) of the "two" directions of the link may be highly asymmetric and may adversely affect higher-layer protocols [RFC3449].

As the satellite example helps to illustrate, one significant issue with tunnels is the amount of effort required to configure them, which has traditionally been done by hand. Typically, tunnel configuration involves selecting the endpoints of a tunnel and configuring the devices located at the tunnel endpoints with an IP address of the peer, and perhaps also providing protocol selection and authentication information. A number of techniques have arisen to help in configuring or using tunnels automatically. One such approach specified for transitioning from IPv4 to IPv6 is called *6to4* [RFC3056]. In 6to4, IPv6 packets are tunneled over an IPv4 network using the encapsulation specified in [RFC3056]. A problem with this approach occurs when corresponding hosts are located behind network address translators (see Chapter 7). This is common today, especially for home users. Dealing with the IPv6 transition using automatically configured tunnels is specified in an approach called *Teredo* [RFC4380]. Teredo tunnels IPv6 packets over UDP/IPv4 packets. Because this approach requires some background in IPv4 and IPv6, as well as UDP, we postpone any detailed discussion of such tunnel autoconfiguration options to Chapter 10.

3.10 Attacks on the Link Layer

Attacking layers below TCP/IP in order to affect the operations of TCP/IP networks has been a popular approach because much of the link-layer information is not shared by the higher layers and can therefore be somewhat difficult to detect and mitigate. Nevertheless, many such attacks are now well understood, and we

mention a few of them here to better understand how problems at the link layer can affect higher-layer operations.

In conventional wired Ethernet, interfaces can be placed in *promiscuous mode*, which allows them to receive traffic even if it is not destined for them. In the early days of Ethernet, when the medium was literally a shared cable, this capability allowed anyone with a computer attached to the Ethernet cable to "sniff" anybody else's frames and inspect their contents. As many higher-layer protocols at the time included sensitive information such as passwords, it was nearly trivial to intercept a person's password by merely looking at the ASCII decode of a packet trace. Two factors have affected this approach substantially: the deployment of switches and the deployment of encryption in higher-layer protocols. With switches, the only traffic that is provided on a switch port to which an end station is attached is traffic destined for the station itself (or others for which it is bridging) and broadcast/multicast traffic. As this type of traffic rarely contains information such as passwords, the attack is largely thwarted. Much more effective, however, is simply the use of encryption at higher layers, which is now common. In this case, sniffing packets leads to little benefit as the contents are essentially impossible to read.

Another type of attack targets the operation of switches. Recall that switches hold tables of stations on a per-port basis. If these tables are able to be filled quickly (e.g., by quickly masquerading as a large number of stations), it is conceivable that the switch might be forced into discarding legitimate entries, leading to service interruption for legitimate stations. A related but probably worse attack can be mounted using the STP. In this case, an attacking station can masquerade as a switch with a low-cost path to the root bridge and cause traffic to be directed toward it.

With Wi-Fi networks, some of the eavesdropping and masquerading issues present in wired Ethernet networks are exacerbated, as any station can enter a monitoring mode and sniff packets from the air (although placing an 802.11 interface into monitoring mode tends to be more challenging than placing an Ethernet interface into promiscuous mode, as doing so depends on an appropriate device driver). Some of the earliest "attacks" (which may not really have been attacks, depending on the relevant legal framework) involved simply roaming about while scanning, looking for access points providing Internet connectivity (i.e., *war driving*). Although many access points use encryption to limit access to authorized users, others are either open or use so-called *capturing portals* that direct a would-be user to a registration Web page and then filter access based on MAC address. Capturing portal systems have been subverted by observing a station as it registers and "hijacking" the connection as it is formed by impersonating the legitimate registering user.

A more sophisticated set of attacks on Wi-Fi involves attacking the cryptographic protection, especially the WEP encryption used on many early access points. Attacks on WEP [BHL06] were sufficiently devastating so as to prod the IEEE into revising the standard. The more recent WPA2 encryption framework (and WPA, to a lesser extent) is known to be significantly stronger, and WEP is no longer recommended for use.

PPP links can be attacked in a number of ways if the attacker can gain access to the channel between the two peers. For very simple authentication mechanisms (e.g., PAP), sniffing can be used to capture the password in order to facilitate illegitimate subsequent use. Depending on the type of higher-layer traffic being carried over the PPP link (e.g., routing traffic), additional unwanted behaviors can be induced.

In terms of attacks, tunneling can play the role of both target and tool. In terms of a target, tunnels pass through a network (often the Internet) and thus are subject to being intercepted and analyzed. The configured tunnel endpoints can also be attacked, either by attempting to establish more tunnels than the endpoint can support (a DoS attack) or by attacking the configuration itself. If the configuration is compromised, it may be possible to open an unauthorized tunnel to an endpoint. At this point the tunnel becomes a tool rather than a target, and protocols such as L2TP can provide a convenient protocol-independent method of gaining access to private internal networks at the link layer. In one GRE-related attack, for example, traffic is simply inserted in a nonencrypted tunnel, where it appears at the tunnel endpoint and is injected to the attached "private" network as though it were sent locally.

3.11 Summary

In this chapter we examined the lowest layer in the Internet protocol suite with which we are concerned—the link layer. We looked at the evolution of Ethernet, in terms of both its increases in speed from 10Mb/s to 10Gb/s and beyond, as well as its evolution of capabilities, including VLANs, priorities, link aggregation, and frame formats. We saw how switches provide improved performance over bridges by implementing a direct electrical path between multiple independent sets of stations, and how full-duplex operation has largely replaced the earlier half-duplex operation. We also looked at the IEEE 802.11 wireless LAN "Wi-Fi" standard in some detail, noting its similarities and differences with respect to Ethernet. It has become one of the most popular IEEE standards and provides license-free network access across the two primary bands of 2.4GHz and 5GHz. We also looked at the evolution of the security methods for Wi-Fi, with the evolution from the relatively weak WEP to the more formidable WPA and WPA2 frameworks. Moving beyond IEEE standards, we discussed point-to-point links and the PPP protocol. PPP can encapsulate essentially any kind of packets used for TCP/IP and non-TCP/IP networks using an HDLC-like frame format, and it is used on links ranging from low-speed dial-up modems to high-speed fiber-optic lines. It is a whole suite of protocols itself, including methods for compression, encryption, authentication, and link aggregation. Because it supports only two parties, it does not have to deal with controlling access to a shared medium like the MAC protocols of Ethernet or Wi-Fi.

The loopback interface is provided by most implementations. Access to this interface is either through the special loopback address, normally 127.0.0.1 (::1 for IPv6), or by sending IP datagrams to one of a host's own IP addresses. Loopback data has been completely processed by the transport layer and by IP when it loops around to go up the protocol stack. We described an important feature of many link layers, the MTU, and the related concept of a path MTU.

We also discussed the use of tunneling, which involves carrying lower-layer protocols in higher-layer (or equal-layer) packets. This technique allows for the formation of overlay networks, using tunnels over the Internet as links in another level of network infrastructure. This technique has become very popular, both for experimentation with new capabilities (e.g., running an IPv6 network overlay on an IPv4 internet) and for operational use (e.g., with VPNs).

We concluded the chapter with a brief discussion of the types of attacks involving the link layer—as either target or tool. Many attacks simply involve intercepting traffic for analysis (e.g., looking for passwords), but more sophisticated attacks involve masquerading as endpoints or modifying traffic in transit. Other attacks involve compromising control information such as tunnel endpoints or the STP to direct traffic to otherwise unintended locations. Access to the link layer also provides an attacker with a general way to perform DoS attacks. Perhaps the best-known variant of this is jamming communication signals, an endeavor undertaken by certain parties since nearly the advent of radio.

This chapter has covered only some of the common link technologies used with TCP/IP today. One reason for the success of TCP/IP is its ability to work on top of almost any link technology. In essence, IP requires only that there exists some path between sender and receiver(s) across a cascade of intermediate links. Although this is a relatively modest requirement, some research is aimed at stretching this even farther—to cases where there may never be an end-to-end path between sender and receiver(s) at any single point in time [RFC4838].

3.12 References

[802.11-2007] "IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," June 2007.

[802.11n-2009] "IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," Oct. 2009.

[802.11y-2008] "IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: 3650-3700 MHz Operation in USA," Nov. 2009.

[802.16-2009] "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems," May 2009.

[802.16h-2010] "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 2: Improved Coexistence Mechanisms for License-Exempt Operation," July 2010.

[802.16j-2009] "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 1: Multihop Relay Specification," June 2009.

[802.16k-2007] "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 5: Bridging of IEEE 802.16," Aug. 2010.

[802.1AK-2007] "IEEE Standard for Local and Metropolitan Area Networks, Virtual Bridged Local Area Networks Amendment 7: Multiple Registration Protocol," June 2007.

[802.1AE-2006] "IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Security," Aug. 2006.

[802.1ak-2007] "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks—Amendment 7: Multiple Registration Protocol," June 2007.

[802.1AX-2008] "IEEE Standard for Local and Metropolitan Area Networks—Link Aggregation," Nov. 2008.

[802.1D-2004] "IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Bridges," June 2004.

[802.1Q-2005] IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks," May 2006.

[802.1X-2010] "IEEE Standard for Local and Metropolitan Area Networks Port-Based Network Access Control," Feb. 2010.

[802.2-1998] "IEEE Standard for Local and Metropolitan Area Networks Logical Link Control" (also ISO/IEC 8802-2:1998), May 1998.

[802.21-2008] "IEEE Standard for Local and Metropolitan Area Networks, Part 21: Media Independent Handover Services," Jan. 2009.

[802.3-2008] "IEEE Standard for Local and Metropolitan Area Networks, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," Dec. 2008.

[802.3at-2009] "IEEE Standard for Local and Metropolitan Area Networks—Specific Requirements, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment 3: Date Terminal Equipment (DTE) Power via the Media Dependent Interface (MDI) Enhancements," Oct. 2009.

[802.3ba-2010] "IEEE Standard for Local and Metropolitan Area Networks, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Amendment 4: Media Access Control Parameters, Physical Layers, and Management Parameters for 40Gb/s and 100Gb/s Operation," June 2010.

[802.11n-2009] "IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 5: Enhancements for Higher Throughput," Oct. 2009.

[AES01] U.S. National Institute of Standards and Technology, FIPS PUB 197, "Advanced Encryption Standard," Nov. 2001.

[BHL06] A. Bittau, M. Handley, and J. Lackey, "The Final Nail in WEP's Coffin," *Proc. IEEE Symposium on Security and Privacy*, May 2006.

[BOND] http://bonding.sourceforge.net

[ETHERTYPES] http://www.iana.org/assignments/ethernet-numbers

[ETX] D. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," *Proc. Mobicom*, Sep. 2003.

[G704] ITU, "General Aspects of Digital Transmission Systems: Synchronous Frame Structures Used at 1544, 6312, 2048k, 8488, and 44736 kbit/s Hierarchical Levels," ITU-T Recommendation G.704, July 1995.

[IANA-CHARSET] "Character Sets," http://www.iana.org/assignments/character-sets

[ISO3309] International Organization for Standardization, "Information Processing Systems—Data Communication High-Level Data Link Control Procedure—Frame Structure," IS 3309, 1984.

[ISO4335] International Organization for Standardization, "Information Processing Systems—Data Communication High-Level Data Link Control Procedure—Elements of Procedure," IS 4335, 1987.

[JF] M. Mathis, "Raising the Internet MTU," http://www.psc.edu/~mathis/MTU

[MWLD] "Long Distance Links with MadWiFi," http://madwifi-project.org/wiki/UserDocs/LongDistance

[PPPn] http://www.iana.org/assignments/ppp-numbers

[RFC0894] C. Hornig, "A Standard for the Transmission of IP Datagrams over Ethernet Networks," Internet RFC 0894/STD 0041, Apr. 1984.

[RFC1042] J. Postel and J. Reynolds, "Standard for the Transmission of IP Datagrams over IEEE 802 Networks," Internet RFC 1042/STD 0043, Feb. 1988.

[RFC1144] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," Internet RFC 1144, Feb. 1990.

[RFC1191] J. Mogul and S. Deering, "Path MTU Discovery," Internet RFC 1191, Nov. 1990.

[RFC1332] G. McGregor, "The PPP Internet Protocol Control Protocol," Internet RFC 1332, May 1992.

[RFC1570] W. Simpson, ed., "PPP LCP Extensions," Internet RFC 1570, Jan. 1994.

[RFC1661] W. Simpson, "The Point-to-Point Protocol (PPP)," Internet RFC 1661/STD 0051, July 1994.

[RFC1662] W. Simpson, ed., "PPP in HDLC-like Framing," Internet RFC 1662/STD 0051, July 1994.

[RFC1663] D. Rand, "PPP Reliable Transmission," Internet RFC 1663, July 1994.

[RFC1853] W. Simpson, "IP in IP Tunneling," Internet RFC 1853 (informational), Oct. 1995.

[RFC1962] D. Rand, "The PPP Compression Protocol (CCP)," Internet RFC 1962, June 1996.

[RFC1977] V. Schryver, "PPP BSD Compression Protocol," Internet RFC 1977 (informational), Aug. 1996.

[RFC1981] J. McCann and S. Deering, "Path MTU Discovery for IP Version 6," Internet RFC 1981, Aug. 1996.

[RFC1989] W. Simpson, "PPP Link Quality Monitoring," Internet RFC 1989, Aug. 1996.

[RFC1990] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP Multilink Protocol (MP)," Internet RFC 1990, Aug. 1996.

[RFC1994] W. Simpson, "PPP Challenge Handshake Authentication Protocol (CHAP)," Internet RFC 1994, Aug. 1996.

[RFC2118] G. Pall, "Microsoft Point-to-Point (MPPC) Protocol," Internet RFC 2118 (informational), Mar. 1997.

[RFC2125] C. Richards and K. Smith, "The PPP Bandwidth Allocation Protocol (BAP)/The PPP Bandwidth Allocation Control Protocol (BACP)," Internet RFC 2125, Mar. 1997.

[RFC2153] W. Simpson, "PPP Vendor Extensions," Internet RFC 2153 (informational), May 1997.

[RFC2290] J. Solomon and S. Glass, "Mobile-IPv4 Configuration Option for PPP IPCP," Internet RFC 2290, Feb. 1998.

[RFC2464] M. Crawford, "Transmission of IPv6 Packets over Ethernet Networks," Internet RFC 2464, Dec. 1988.

[RFC2473] A. Conta and S. Deering, "Generic Packet Tuneling in IPv6 Specification," Internet RFC 2473, Dec. 1998.

[RFC2484] G. Zorn, "PPP LCP Internationalization Configuration Option," Internet RFC 2484, Jan. 1999.

[RFC2507] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression," Internet RFC 2507, Feb. 1999.

[RFC2615] A. Malis and W. Simpson, "PPP over SONET/SDH," Internet RFC 2615, June 1999.

[RFC2637] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)," Internet RFC 2637 (informational), July 1999.

[RFC2759] G. Zorn, "Microsoft PPP CHAP Extensions, Version 2," Internet RFC 2759 (informational), Jan. 2000.

[RFC2784] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)," Internet RFC 2784, Mar. 2000.

[RFC2865] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," Internet RFC 2865, June 2000.

[RFC2890] G. Dommety, "Key and Sequence Number Extensions to GRE," Internet RFC 2890, Sept. 2000.

[RFC3056] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," Internet RFC 3056, Feb. 2001.

[RFC3077] E. Duros, W. Dabbous, H. Izumiyama, N. Fujii, and Y. Zhang, "A Link-Layer Tunneling Mechanism for Unidirectional Links," Internet RFC 3077, Mar. 2001.

[RFC3078] G. Pall and G. Zorn, "Microsoft Point-to-Point Encryption (MPPE) Protocol," Internet RFC 3078 (informational), Mar. 2001.

[RFC3153] R. Pazhyannur, I. Ali, and C. Fox, "PPP Multiplexing," Internet RFC 3153, Aug. 2001.

[RFC3366] G. Fairhurst and L. Wood, "Advice to Link Designers on Link Automatic Repeat reQuest (ARQ)," Internet RFC 3366/BCP 0062, Aug. 2002.

[RFC3449] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry," Internet RFC 3449/BCP 0069, Dec. 2002.

[RFC3544] T. Koren, S. Casner, and C. Bormann, "IP Header Compression over PPP," Internet RFC 3544, July 2003.

[RFC3561] C. Perkins, E. Belding-Royer, and S. Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing," Internet RFC 3561 (experimental), July 2003.

[RFC3610] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," Internet RFC 3610 (informational), Sept. 2003.

[RFC3626] T. Clausen and P. Jacquet, eds., "Optimized Link State Routing Protocol (OLSR)," Internet RFC 3626 (experimental), Oct. 2003.

[RFC3748] B. Aboba et al., "Extensible Authentication Protocol (EAP)," Internet RFC 3748, June 2004.

[RFC3931] J. Lau, M. Townsley, and I. Goyret, eds., "Layer Two Tunneling Protocol—Version 3 (L2TPv3)," Internet RFC 3931, Mar. 2005.

[RFC4017] D. Stanley, J. Walker, and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs," Internet RFC 4017 (informational), Mar. 2005.

[RFC4380] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," Internet RFC 4380, Feb. 2006.

[RFC4647] A. Phillips and M. Davis, "Matching of Language Tags," Internet RFC 4647/BCP 0047, Sept. 2006.

[RFC4821] M. Mathis and J. Heffner, "Packetization Layer Path MTU Discovery," Internet RFC 4821, Mar. 2007.

[RFC4838] V. Cerf et al., "Delay-Tolerant Networking Architecture," Internet RFC 4838 (informational), Apr. 2007.

[RFC4840] B. Aboba, ed., E. Davies, and D. Thaler, "Multiple Encapsulation Methods Considered Harmful," Internet RFC 4840 (informational), Apr. 2007.

[RFC5072] S. Varada, ed., D. Haskins, and E. Allen, "IP Version 6 over PPP," Internet RFC 5072, Sept. 2007.

[RFC5225] G. Pelletier and K. Sandlund, "RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP, and UDP-Lite," Internet RFC 5225, Apr. 2008.

[RFC5646] A. Phillips and M. Davis, eds., "Tags for Identifying Languages," Internet RFC 5646/BCP 0047, Sept. 2009.

[S08] D. Skordoulis et al., "IEEE 802.11n MAC Frame Aggregation Mechanisms for Next-Generation High-Throughput WLANs," *IEEE Wireless Communications*, Feb. 2008.

[S96] B. Schneier, *Applied Cryptography, Second Edition* (John Wiley & Sons, 1996).

[SAE] D. Harkins, "Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks," *Proc. SENSORCOMM*, Aug. 2008.

[SC05] S. Shalunov and R. Carlson, "Detecting Duplex Mismatch on Ethernet," *Proc. Passive and Active Measurement Workshop*, Mar. 2005.

[SHK07] C. Sengul, A. Harris, and R. Kravets, "Reconsidering Power Management," Invited Paper, *Proc. IEEE Broadnets*, 2007.

[WOL] http://wake-on-lan.sourceforge.net