Inquiry Operation

During inquiry operation, each inquiry transmission on a given frequency of the inquiry-hopping sequence and each response to it are preceded by an *inquiry access code* (IAC). There are 64 IACs, including the GIAC, associated with 64 reserved LAPs. Excluding the GIAC, the remaining 63 IACs are referred to as *dedicated IACs* (DIACs). The IAC LAPs belong to the set {'0x9E8B00', ..., '0x9E8B3F'}. No device can have an address whose LAP matches any of these reserved LAPs. Note that no matter which IAC is used, the inquiry-hopping sequence over which this IAC is transmitted is generated using the GIAC. This is done to allow devices with multiple receiver correlators, as highlighted in Figure 6.8, to follow a single inquiry-hopping sequence but still be able to listen to multiple classes of inquiries simultaneously.

While the specification does not define how IACs are to be used, they are intended to be used primarily as a filtering mechanism for identifying well-defined subsets of the devices that may receive inquiries. While all devices that execute inquiry scans use the GIAC to generate the inquiry-hopping frequency, only those devices whose receiver correlators are tuned to a particular IAC will receive and respond to inquiries that contain that particular IAC.

Table 6.3 summarizes the various parameters related to the fundamental processes for each of the three operational states of a Bluetooth device.

Table 6.3
The FSM inputs and the access codes used during the various active states of a device.

Device state	Frequency-hop module	Access code
Connected	*clock*: master's; *address*: master's	channel access code (CAC); it coincides with the master's device access code (DAC)
Inquiry	*clock*: own; *address*: GIAC	general or dedicated inquiry access code (GIAC or DIAC)
Page	*clock*: (estimate of) paged device's; *address*: paged device's	paged device's device access code (DAC)

As discussed earlier, the Bluetooth clock and the *BD_ADDR* of the master of a piconet fully identify the frequency-hopping sequence and phase of the channel used in the piconet. Since communication

between a slave and a master can occur only within a piconet, it follows that each slave in a piconet must know the master's clock and *BD_ADDR*. Alternatively, for a potential master to efficiently invite a potential slave, it needs to know the slave's clock and address. The exchange of address and clock information between devices occurs during inquiries, when the master collects operational information about the prospective slaves, and during pages, when the master communicates to a slave its own operational information. The operational information of a device, which includes its *BD_ADDR* and clock value, is sent in a *frequency-hopping sequence* (FHS) BB_PDU described in detail later in this chapter.

Assuming that the fundamental elements (address and clock) of the devices involved are known, the connected state is highlighted first in the next section. The inquiry and page states are presented afterward.

The Connected State

While in the connected state, devices can exchange data under the control of the master that defines which device transmits when. To maintain piconet synchronization, each slave adds an offset to its native clock that accounts for the difference of its own clock from that of the master. Thus, the clock of the master becomes the regulator of timed events in the piconet. In addition, the LAP of the master is used for the access code generation.

With a common clock reference among all devices in a piconet, the transmission time on the piconet is divided into master and slave transmission slots. A master starts its transmissions on even-numbered slots ($c_1 = 0$) exclusively. Likewise, a slave starts its transmissions on odd-numbered slots ($c_1 = 1$) exclusively. A particular slave transmits if and only if the last master transmission was destined exclusively to this slave. Thus, the medium access protocol for Bluetooth communications is a packet-based, *time-division duplex* (TDD)[7] polling scheme. Typically, master and slave transmit slots alternate every 625 μsec, the residence time at a frequency, although as mentioned earlier, multi-slot transmissions are possible. However, multi-slot transmissions are limited to an odd number of slots (one, three or five), which guarantees that master

7. TDD refers to a time-division multiplexing technique where the transmission time on a single communication channel is divided into successive, non-overlapping intervals, every other of which is used for transmissions in one of two opposing directions. The transmission direction alternates between the two directions with each successive interval.

transmissions always start on even slots and slave transmissions on odd slots.

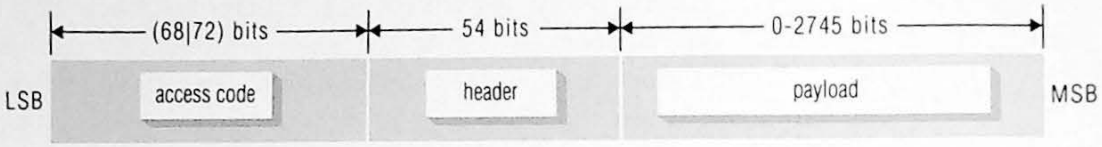

Figure 6.9
The BB_PDU format.

Baseband BB_PDU Types

Figure 6.9 depicts the general format of a BB_PDU transmitted on a piconet. No frequency hops occur during a BB_PDU transmission and at most one BB_PDU is transmitted during the residence time at a frequency. Over-the-air transmissions start with the LSB and proceed to the MSB (*little-endian* transmission ordering). Every BB_PDU transmitted starts with the access code that, for the discussion here, serves as the piconet identifier, thus filtering out transmissions that might be received from other piconets (see Figure 6.8). The access code typically is 72 bits long, which includes a 4-bit trailer field. If the BB_PDU does not contain a header (and hence a payload), the trailer is not used and the BB_PDU consists of the 68-bit access code only. The 68-bit BB_PDUs are used exclusively for transmitting inquiries or pages as described in the corresponding sections later in this chapter.

The header of the BB_PDU contains link control data to aid medium access control. The header contains a mere 18 bits of information for low overhead, but it is encoded with a *forward error-correcting* (FEC) code with rate 1/3 for high transmission reliability. In particular, every bit of the header is transmitted three times in sequence. Table 6.4 summarizes the fields of BB_PDU header.

Table 6.4
The baseband BB_PDU header.

Field name	Size	Comments
AM_ADDR	3 bits	active member address assigned to an active slave when devices exchange paging information, such as during the paging of a device
TYPE	4 bits	defines 16 BB_PDU/payload types
FLOW	1 bit	stop/go flow control switch set by a receiving device in its response to the sender

ARQN	1 bit	used for acknowledging successfully transmitted BB_PDUs; BB_PDUs for which an acknowledgement is not received are retransmitted
SEQN	1 bit	simple (odd/even) sequence number for filtering duplicate transmissions
HEC	8 bits	header error check (HEC) generated by the generator polynomial $G_{\text{HEC}}(x) = x^8 + x^7 + x^5 + x^2 + x + 1$

During the paging process, the master assigns a 3-bit active member address (*AM_ADDR*) to the new slave. *AM_ADDR* takes on the values 1 through 7 and it is unique for each active slave in a piconet. The reserved value of *AM_ADDR* = 'b000' is used to signify broadcast transmissions from the master to all the slaves in a piconet.[8] The *AM_ADDR* is included in the same FHS BB_PDU sent by the master to the slave that also contains the master's address and clock information.

The various BB_PDU types are distinguished by their roles: purely signaling or payload-carrying packets; their link designation: *asynchronous connectionless* (ACL) data or *synchronous connection-oriented* (SCO) data; their size: 1, 3 or 5 slots; and their FEC encoding: no FEC, 2/3 rate FEC encoding, and 1/3 rate FEC encoding. The 2/3 rate FEC is a (15,10) shortened Hamming code generated by the generator polynomial $G_{\text{FEC}}(x) = x^5 + x^4 + x^2 + 1$.

The HEC is implemented through an 8-bit linear feedback shift register (LFSR) circuit whose initial value is UAP[0:7] of the master of the piconet. Hence, even in the case where transmissions from multiple masters with overlapping LAPs (thus generating the identical access code as shown in Figure 6.8) were accepted, the corresponding BB_PDU would be rejected because the BB_PDU header would fail its header error check. In other words, both the LAP and the UAP of the master of a piconet are needed to fully identify and accept a BB_PDU transmission on the piconet.

The link control packets include:

- the ID packet, used in inquiries and pages, that consists of only the access code;
- the NULL packet that is used primarily for slave acknowledgment and flow control information transmission when the slave

8. Due to the TDD polling scheme used for medium access control in a piconet, only the master can directly broadcast data to the other piconet members (its slaves). Slaves in a piconet can only unicast (a point-to-point transmission) to the master of the piconet.

does not have anything else to transmit (NULL packets themselves are not acknowledged);
- the POLL packet that is sent from a master to a slave to poll it for a transmission when the master does not have any payload information to send to the slave (the POLL packet requires a response); and
- the FHS packet used in inquiries and pages.

The ACL packets are designated as D(M|H)(1|3|5), where "DM" stands for medium speed data that use a 2/3 FEC encoding for the payload; "DH" stands for high speed data where no FEC is used for the payload. The size qualifier 1, 3 or 5 refers to the number of slots occupied by the packet. For multi-slot packets, the frequency does not change until the packet finishes its transmission. The next frequency to be used is the one that would have been used if single-slot transmissions were used in the meantime instead. Note that the size of an ACL packet is odd because a master's transmission must always start at even-numbered slots while slave transmissions must start at odd-numbered slots. Using five-slot packets in one direction and one-slot packets in the opposing direction, the maximum achievable rate for ACL traffic is 723.2 Kbps in the first direction and 57.6 Kbps in the opposite direction.

Knowledge of the type of a BB_PDU and hence its size facilitates power conservation in a slave. In particular, as a slave in a piconet inspects an incoming transmission and finds that the transmission is not intended for that slave (that is, its *AM_ADDR* does not match the address in the BB_PDU header), the slave could go to "sleep" for a duration of time dictated by the packet type field.

The SCO packets are one slot long and are designated as HV(1|2|3). All three variants can support 64 Kbps in each direction over periodically reserved slots using different encoding for the payload data. In particular, HV stands for high-quality voice, and the encoding qualifier 1, 2 or 3 refers to the encoding used for the payload data:

- 1 is for 1/3 rate FEC; a device transmits a single-slot packet every 2 slots
- 2 is for 2/3 rate FEC; a device transmits a single-slot packet every 4 slots
- 3 is for no FEC; a device transmits a single-slot packet every 6 slots

In addition to the above packet types there is a DV packet that combines both ACL, with 2/3 FEC, and SCO, with no FEC, data in a single-slot packet. A device could transmit a DV packet every 2 slots.

Figure 6.10 depicts the low-level baseband operations that occur during the transmission and reception of baseband packets. Note that different operations take place for the packet header versus the payload; the operations for the header and the payload occur serially rather than in parallel as the figure might imply. In the figure, *whitening* refers to the process of scrambling the transmitted data to randomize them and to reduce the DC bias in the transmitted data. The data are whitened through the use of the whitening word generated by the polynomial $G_{WHITE}(x) = x^7+x^4+1$. Security features in Bluetooth piconets, including device authentication and link encryption, are discussed in the following LMP section. The CRC operation pertains only to ACL packets, detailed immediately below.

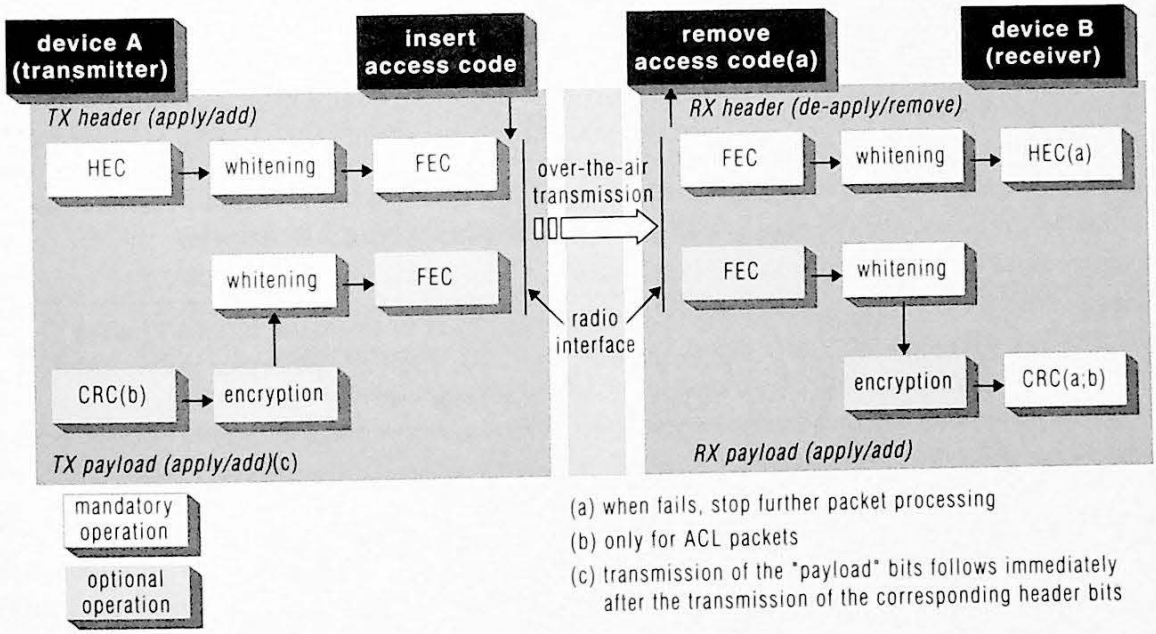

Figure 6.10
Low-level baseband operations during transmission and reception of baseband packets.

Asynchronous Connectionless (ACL) Packets

The payload portion of the DM packets and the ACL portion of a DV packet is further structured with its own ACL packet header and payload along with a two-byte *cyclic redundancy check* (CRC) field. Table 6.5 summarizes the fields of the ACL packets, from the LSB to MSB.

Table 6.5
The ACL packet format.

Field name	Size	Comments
L_CH (logical channel)	2 bits	'b00': not defined
		'b01': continuation segment of an L2CAP PDU
		'b10': start of an L2CAP PDU
		'b11': LMP PDU
FLOW	1 bit	for flow control on the ACL link
LENGTH	(5|9) bits	length of ACL payload in octets (excludes header and CRC) for either single- or multi-slot packets in the case of a 9-bit length field, a 4-bit undefined padding is used to make the header an integer multiple of bytes (2 bytes total)
Payload	0–339 bytes	ACL packet payload spanning 1 to 5 slots
CRC	2 bytes	the *cyclic redundancy check* protects the payload and is generated by the CRC-CCITT generator polynomial $G_{CRC}(x) = x^{16} + x^{12} + x^5 + 1$

There is one extra one-slot ACL packet, AUX1, which does not contain a CRC. The use of this packet is not defined in the specification. Possibly, it could be used for testing and development purposes; however, it is outside the scope of the version 1.0 specification and this discussion.

The Baseband Link Types

As mentioned earlier, the Bluetooth baseband supports two types of links over a piconet. ACL links are used for carrying asynchronous data. The master schedules asynchronous transmissions in slots not already reserved for synchronous (SCO) transmissions. In other words, SCO traffic is of high priority and cannot be preempted for asynchronous transmissions. For each and every slave in its piconet, a master establishes exactly one ACL link that represents a physical pipe between the master-slave pair over which ACL data are exchanged. Point-to-point ACL BB_PDU exchanges are all acknowledged and

retransmitted as appropriate. Broadcast (*AM_ADDR* = 'b000') ACL transmissions from a master to its slaves are not acknowledged but may be repeated several, N_{BC}, times for increased reliability. Note that, since it is impossible for multiple slaves to acknowledge a transmission simultaneously, it is also impossible to acknowledge a broadcast transmission. In particular, there exists no simple and reliable method to dynamically designate a single slave to acknowledge on behalf of all slaves in the piconet that the broadcast transmission has been successfully received by all slaves.

SCO links carry telephony-grade voice audio through *a priori* periodically reserved pairs of slots. In a piconet, following a request from a slave or the master, a master may establish up to three SCO links in total between it and all of its slaves. Each SCO link represents a physical pipe between the master and one of its slaves over which SCO data are exchanged. To maintain the high quality of service expected for SCO traffic, the length of the baseband slot, 625 µsec, is selected to minimize the packetization delay for audio traffic and the effects of noise interference. For example, an HV1 BB_PDU carries the equivalent of 10 bytes of audio information, which at 8,000 samples per second and 8 bits per sample takes 1.25 msec (=2*625 µsec) to accumulate. This is the same as the period of transmissions of HV1 BB_PDUs from a device. Unlike ACL transmissions, SCO BB_PDUs are not acknowledged. Note that prior to establishing an SCO link, an ACL link must already exist to carry, at a minimum, the SCO connection control information.

Bluetooth links between devices can be authenticated, encrypted, operated in low-power mode, and in general, qualified based upon application or user requirements. The operation of the baseband in these cases is highlighted in the following link manager protocol section since it is link manager involvement that initiates these sorts of changes in link behavior.

For communication in a piconet to occur, a slave needs to know its master's clock and address. The following sections discuss the inquiry and page mechanisms by which this information is acquired. As Figure 6.4 shows, this two-step process can be trimmed to one step when connecting to known devices.

Inquiry State

The purpose of device inquiries is to collect information about other Bluetooth devices in proximity. This primarily involves obtaining their fundamental elements: *BD_ADDR* and clock value. The inquiry state is

composed of several substates executed by "potential" masters and slaves. These are the *inquiry* substate executed by a potential master and the *inquiry scan* and *inquiry response* substates executed by a potential slave. In the inquiry substate, a master[9] transmits inquiry packets, which are received by slaves in the inquiry scan substate. The latter devices then enter the inquiry response substate and schedule the transmission of their fundamental elements to the master.

While in the various inquiry substates, devices utilize the inquiry-hopping sequence to transmit and receive packets. The inquiry packet sent by an inquiring master is a BB_PDU that contains only an appropriate IAC, typically the GIAC, as described in the preceding section on access codes. This packet is referred to as the *inquiry ID* packet. The *inquiry ID* packet simply notifies slaves that a master is looking for slaves.

In responding to an inquiry, a slave transmits an FHS packet containing, among other information, the slave's *BD_ADDR* and clock value at the time of transmission of the FHS packet. Table 6.6 depicts the contents of the FHS packet. Since either a master or a slave can send an FHS packet,[10] the fields in the FHS packet are interpreted with respect to the device that sent the packet. The fields and the bits within them are provided in the transmission order from the LSB to the MSB.

Table 6.6
The FHS packet.

Field name	Size	Comments
parity bits	34 bits	used for building the device access code (DAC) for paging the device sending this packet
LAP	24 bits	the lower address part of the *BD_ADDR* of the device sending this packet
Reserved	2 bits	set to 'b00'
Paging interval parameters	4 bits	two 2-bit subfields defining the period of the successive page scans and the duration of the mandatory page scan period for the device sending this packet
UAP	8 bits	the upper address part of the *BD_ADDR* of the device sending this packet

9. For brevity, the qualifier "potential" for masters and slaves will be omitted unless it is required for clarifying the context.
10. A master sends an FHS packet during a page operation as described in the following section.

NAP	16 bits	the non-significant address part of the *BD_ADDR* of the device sending this packet
CoD	24 bits	class of device field containing information regarding the device sending this packet
AM_ADDR	3 bits	for inquiry responses, it is set to 'b000'; for a master response following a page response by a (potential) slave, it is set to the active member address assigned to the new active slave
CLOCK [2:27]	26 bits	the current time (updated with each retransmission of the packet) of the Bluetooth clock of the device sending this packet
Page scan mode	3 bits	the default page scan mode of the device; one mandatory page scan mode and up to three optional ones are supported in version 1.0

The header of the BB_PDU carrying the FHS packet as payload has the *AM_ADDR* field set to 'b000' without meaning that this is a broadcast packet. The FHS packet is protected by a 2-byte CRC and encoded with an FEC with rate 2/3. The size of the FHS packet is 240 bits of payload and 126 bits of packet header and access code.

The operations of a master, the inquiring device, and a slave, the listening device, are highlighted in Figure 6.11. The numbers shown in the figure are typical numbers, which can be changed through intervention from applications as needed.

In summary, a master transmits its inquiry ID packets on different frequencies of the inquiry-hopping sequence, changing the transmission frequency rapidly in the hope of transmitting as soon as possible on one of the frequencies that slaves are listening to. This would ultimately happen, given enough time, as slaves performing inquiry scans change their listening frequency from the inquiry-hopping sequence at a much slower rate. Since a master does not know when a slave listens for inquiries, the master repeats its inquiry transmissions a number of times. When "contact" finally occurs, the slave responds with its FHS packet that includes vital paging information. SCO transmissions scheduled in any of the inquiring or listening devices take precedence over the inquiry procedures. A device will forego an inquiry action to transmit and receive scheduled SCO packets.

The inquiry-hopping rate is 3,200 hops per second, double the nominal frequency-hopping rate. The master will transmit and listen for responses over 16 different frequencies of the inquiry-hopping sequence

(train A in Figure 6.11) over a period of 10 msec (8 transmit and 8 receive slots alternating with each other). If an insufficient number of responses is gathered, the master will repeat the same process over the same set of 16 frequencies at least 256 times, or 2.56 seconds. Following this interval and assuming operation in a 79 channel country, the master may also search through the remaining 16 frequencies of the inquiry-hopping sequence (train B in Figure 6.11) an additional 256 times. The whole inquiry process may then be repeated from the beginning.

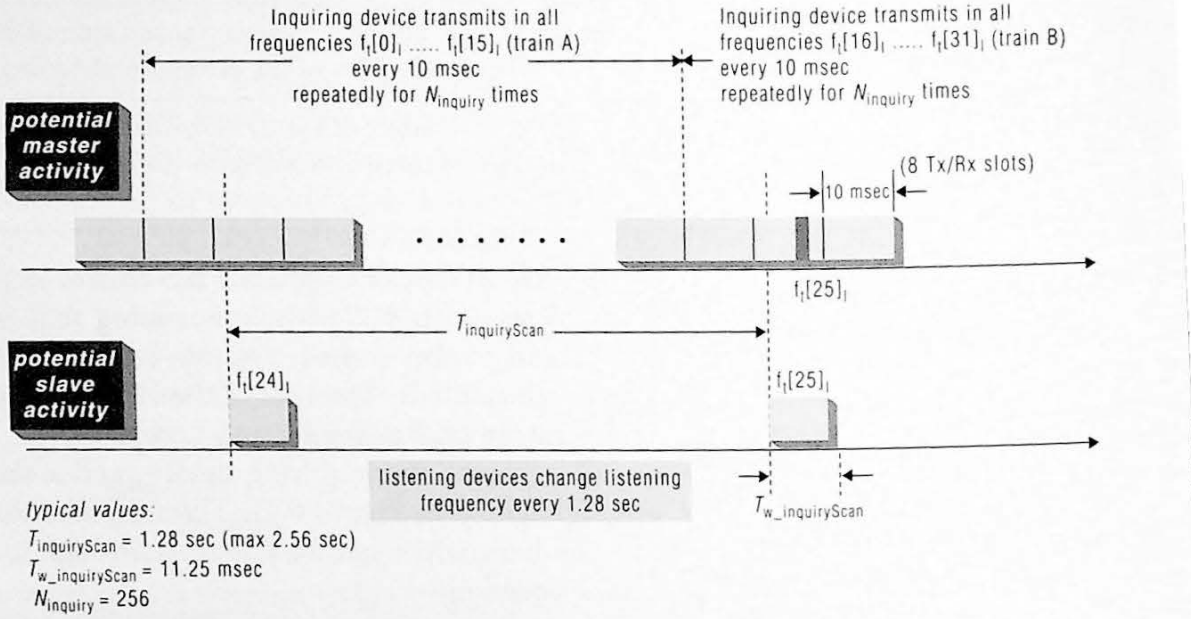

Figure 6.11
Fast frequency scanning for inquiries in a 79 channel country; $f_t[.]_I$ denotes the master transmit frequencies from the inquiry-hopping sequence.

To avoid collisions from multiple slaves responding to the same inquiry ID packet, a rare event in its own right, a back-off mechanism is used and it works as follows. Upon receipt of an inquiry ID packet, the slave enters the inquiry response substate. It then randomly selects a number *RN* (≤ 1,023) and suspends its inquiry response operations for at least that many slots; the slave may move into the standby, connected or page states as necessary. When the slave resumes the inquiry response substate, it will respond with an FHS packet following the first inquiry ID packet it receives again. The detailed description of the inquiry procedures can be found in section 10.7 of the Baseband part of the specification.

Responding immediately after receiving an inquiry ID packet is not prudent, as the master may not have switched to the mode in which

it listens for responses. To guarantee that this does not occur, the slave responds with its FHS packet 625 μsec after the receipt of the inquiry ID packet, as shown in Figure 6.12. The figure shows two slaves, noted as *i* and *j*, responding to inquiry ID packets. Slave *i* heard the inquiry ID packet in the first half of a master transmit slot. Slave *j* heard the inquiry ID packet in the second half of a transmit slot from the master. In either case, even though the slave is unaware exactly when the master will switch to the proper listening frequency, it is guaranteed that the corresponding FHS packet from a slave is sent at the right transmit frequency at the time that the master is listening at that same frequency.

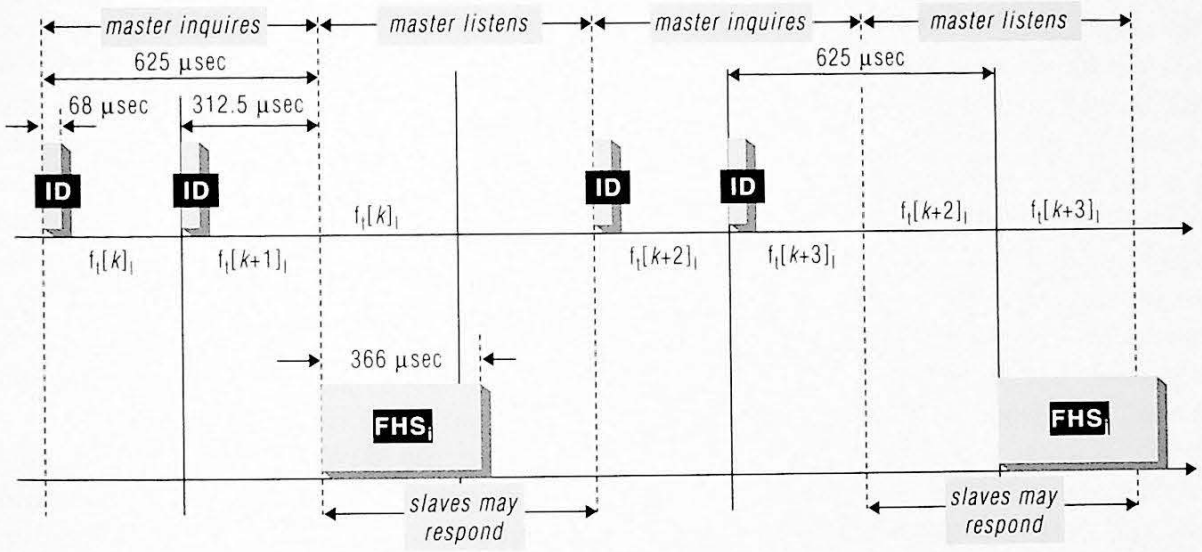

Figure 6.12
Inquiry transmission sequences; $f_r[.]_i$ denotes the master receive frequencies from the inquiry-hopping sequence corresponding to $f_t[.]_i$.

Note that the inquiry ID packet is 68 μsec long and it is transmitted twice over two different frequencies within a 625 μsec time interval. Therefore, closely observing the figure, one may deduce that the transmit frequency synthesizer has 244.5 μsec in which to switch to a new frequency. Actually, accounting for a ±10 μsec tolerance for clock drift, the synthesizer must settle to a new transmit frequency within 224.5 μsec. Admittedly this number is large compared to state-of-the-art synthesizer design. However, it allows the use of less complex and less precise circuitry that results in the desired low cost system design point. For more information on this subject, see [Haartsen00].

Page State

The purpose of a device page is to invite a specific paged device to join a piconet whose master is the paging device. The paging device uses the paged device's *BD_ADDR* and clock estimate to send its pages as discussed earlier. The page state is composed of several substates executed by potential masters and slaves. These are the *page* and *master response* substates executed by a master and the *page scan* and *slave response* substates executed by a slave.

In the page substate, a master transmits page BB_PDUs, which are received by the slave when it is in the page scan substate. The paging transmission contains only the slave's DAC. This BB_PDU is referred to as the *slave ID* packet. In its page response, the slave also sends the slave ID packet that simply notifies the master that the slave has received the page. Finally, the master enters the master response substate during which the master transmits to the slave its fundamental elements and the slave's *AM_ADDR*, which allows the slave to join and participate in communications in the master's piconet. These fundamental elements and AM_ADDR are transmitted within an FHS packet (see Table 6.6). The slave responds with one more slave ID packets, and then enters the connected state and readies itself to start piconet communications.

The operation of the master and the slave in the page and page scan substates, respectively, is quite similar to that of the inquiry and inquiry scan operations discussed earlier. In particular, the illustration of Figure 6.11 for inquiries can also be used for pages as well by replacing the terms related to inquiries with corresponding terms related to pages. The typical value for $T_{pageScan}$ is 1.28 seconds; subsequently, the typical value for N_{page} is 128. These typical values correspond to the so-called R1 paging mode. The paging mode, and hence the corresponding parameters $T_{pageScan}$ and N_{page}, are communicated to the master during the slave FHS transmission in an inquiry, or during link manager information exchange during regular communications.

For a 79-channel country, trains A and B contain 16 frequencies each from the page-hopping sequence, while for 23-channel countries there is only one train containing all 16 frequencies. In the former case, train A contains the 16 frequencies closest to the frequency on which the slave listens for pages as estimated by the master using its knowledge of (the estimate of) the slave's native clock. Train B contains the remaining frequencies from the page-hopping sequence. Yet, since train B is used 1.28 seconds after transmitting pages on frequencies from train A, the

master knows that the frequency on which the slave listens will also move by one. Hence, trains A and B have one common frequency. For example, if train A contains frequencies $f_t[0]_P$ through $f_t[15]_P$, then when train B is used, it will contain frequencies $f_t[17]_P$ through $f_t[0]_P$, in that order. A device that performs page scans in the R1 paging mode can be located within no more than 1.28 seconds most of the time. When the estimate of the slave's native clock deviates more than –8*1.28 seconds or +7*1.28 seconds from the actual value of the slave's clock, the search time can be as much as twice the nominal value, or 2.56 seconds.

The sequence of transmissions during paging operations is summarized in Figure 6.13, where a master pages a slave denoted as *i*. Following the paging operation, the slave shall be able to participate in piconet communications. Hence, the slave not only needs to learn about the master's *BD_ADDR* and clock value but also must have an *AM_ADDR* assigned to it and must know exactly when a master transmit slot starts. The *AM_ADDR* is included in the FHS packet transmission to slave *i*. The time of transmission of this packet is used to identify the start of the master transmit slots in the piconet. The master transmits its FHS packet to slave *i* at the beginning of its transmit slot, regardless of whether the slave sends its previous slave ID packet in the first or second half of the slot during which the master listens for the slave response to its pages. In the example of Figure 6.13, since the master receives the slave response to the master's pages in the second half of the master's receive slot, the master transmits the FHS packet 312.5 μsec after it receives the slave ID packet. Thus, by the time that the slave receives and processes the FHS packet, the slave has all the information needed to participate in the piconet communications, starting with a master transmission 1.25 msec from the start of reception of the FHS packet.

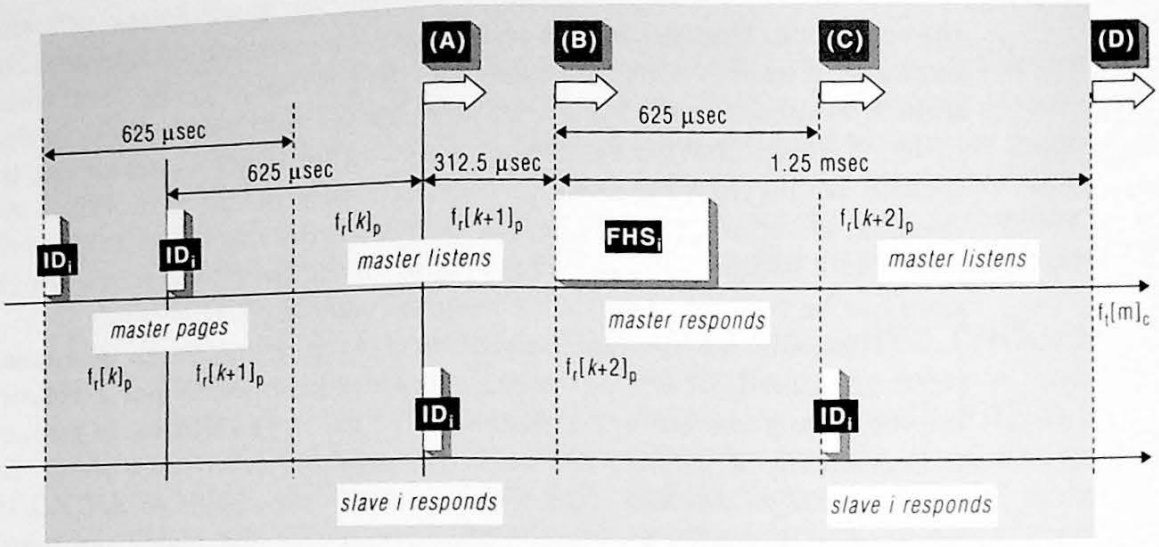

Figure 6.13
Page transmission sequence. $f_t[.]_P$ and $f_r[.]_P$ denote the corresponding master transmit and receive frequencies, respectively, from the page-hopping sequence and $f_t[.]_C$ denotes a master transmission frequency from the channel-hopping sequence.

As with inquiries, SCO transmissions scheduled in any of the paging or paged devices take precedence over the paging procedures. A device will forego a page action to transmit and receive scheduled SCO packets. This case is not elaborated further here. The detailed description of the page procedures can be found in section 10.6 of the Baseband part of the specification.

The Link Manager and Link Manager Protocol

Link manager entities, or simply link managers, in communicating devices exchange messages to control the Bluetooth link between those devices. The communication protocol between link managers is called the *link manager protocol* (LMP) and the messages exchanged between communicating link managers are noted as LMP_PDUs. Figure 6.14 summarizes the functions of a link manager. LMP does not carry application data. Based upon control data it receives from higher layers, LMP either communicates with the link manager in another device using LMP_PDUs or it sends control signals to its own device's baseband and radio layers. It should be noted that, contrary to baseband

processes that are executed in real time, link manager interactions are not. Albeit infrequently, communicating link managers may take up to 30 seconds to react to each other's requests.

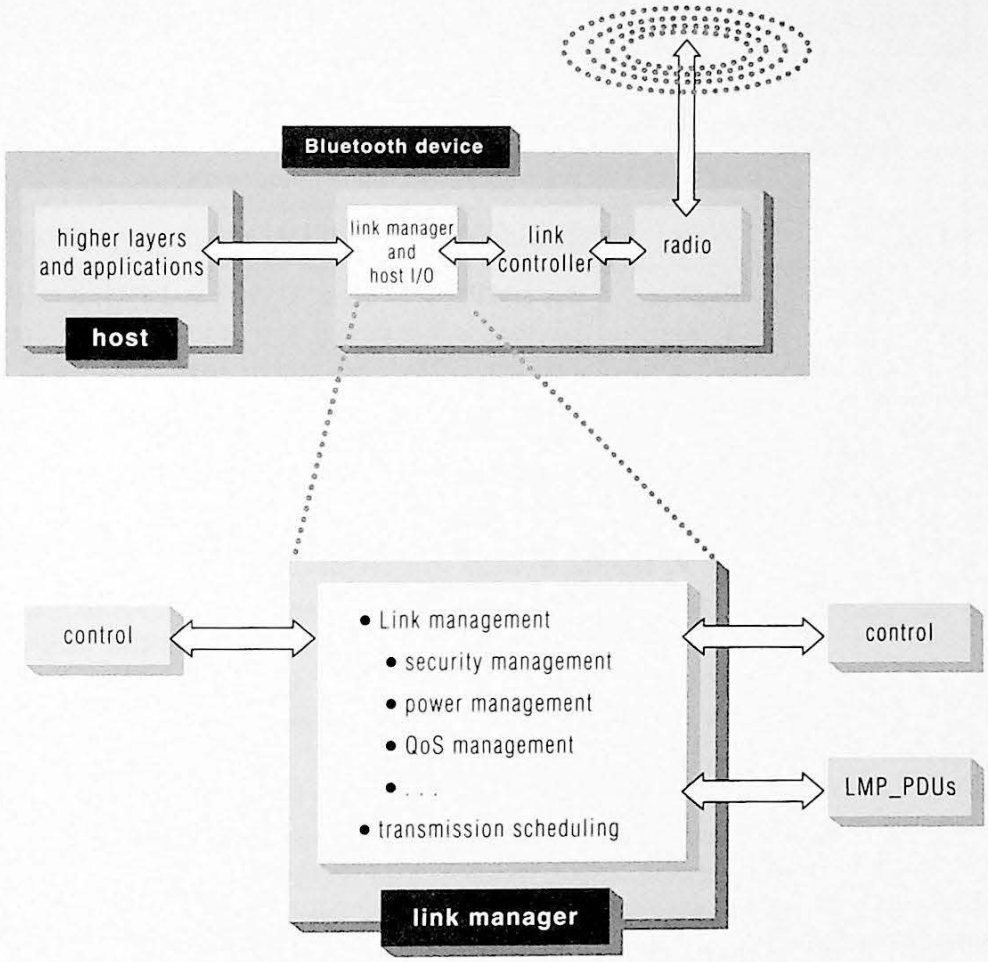

Figure 6.14
The link manager functions.

As discussed earlier (see Table 6.5), LMP_PDUs are carried in the payload of ACL packets whose header has an L_CH field with the value 'b11'. LMP_PDUs are transmitted on single-slot DM1 packets or on DV packets. LMP_PDUs have very high priority and, if needed, they can preempt even an SCO transmission to transmit control information to another device. Table 6.7 summarizes the LMP_PDU packet format; note that LMP_PDU packet format.

Table 6.7
The LMP_PDU format.

Field name	Size	Comments
transactionID	1 bit	'b0': identifies link manager transaction initiated by the master
		'b1': identifies link manager transaction initiated by a slave
OpCode	7 bits	identifies the LMP_PDU and the type of contents it carries
payload	0–17 bytes	the LMP_PDU fits in DM1 BB_PDU; if the LMP_PDU payload is less than 9 bytes then, when supported, DV BB_PDUs may also be used

When a link manager in a device initiates an LMP_PDU transaction with the link manager in another device, the latter link manager will respond with the next LMP_PDU in the transaction sequence (for example, respond with the requested information). Alternatively, the receiving link manager responds with an *LMP_accepted* or *LMP_not_accepted* PDU that signifies whether the link manager request from the transaction initiator is or is not accepted, respectively. When an *LMP_not_accepted* PDU is sent, the reason for not accepting the transaction is provided.

Figure 6.15 shows two typical types of LMP_PDU transactions. In the first, either one of the communicating link managers initiates a transaction with a request. The receiving link manager either will accept the request and act accordingly (perhaps providing the requested information) or will reject the request with an *LMP_not_accepted* PDU. It might also send its own corresponding request LMP_PDU initiating a negotiation phase for the original request. In the second transaction type, the master sends a command to be executed by the slave in an LMP_PDU, without the opportunity for the slave to reject the command or negotiate its parameters. An example of the latter case is forcing a slave to go into a power-saving mode, like the hold mode, or requesting a link detachment, which is the only operation that a slave may also force on a master.

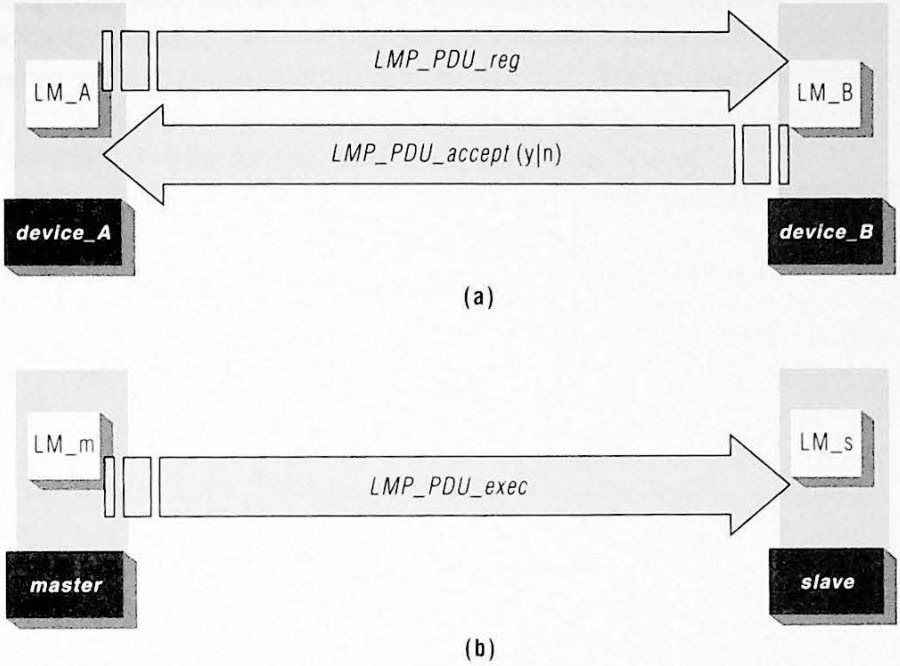

Figure 6.15
Typical LMP_PDU transactions: (a) a request/response transaction and negotiation; (b) master demands link adjustment.

There are many link manager transactions from a simple device name exchange to elaborate authentication and encryption transactions, but not all of them are mandatory. However, the receiving link manager must be able to respond to all link manager transaction requests, even if the response is an *LMP_not_accepted* PDU, with the reason for non-acceptance being "feature not supported." The following sections focus on a few of the more important link manager transactions. The interested reader can find a more detailed presentation in the LMP part of the specification.

Security Management

Security has been part of the specification from the outset of its development. It was recognized early on that in ad hoc, wireless, and especially RF environments security is of paramount importance.

The baseband defines security algorithms and procedures needed to authenticate devices, and if needed to encrypt the data flowing on the link between them. Section 14 of the baseband part of the specification includes algorithms for the generation of authentication and encryption keys and the operations for verifying the authenticity of a device. Even

though security is presented mostly in the baseband part of the specification, it is discussed here in the link manager section, since it ultimately relates to the configuration of a link between devices, the responsibility for which lies with the link managers.

Device authentication is a mandatory feature supported by all Bluetooth devices. Link encryption is optional.

Device Authentication

Authentication of Bluetooth devices is based on a challenge-response transaction. In this transaction, a *verifier* challenges a *claimant* by sending to the latter a 16-byte random number in an *LMP_au_rand* PDU. The claimant operates on the random number and returns the result of the operation to the verifier in an *LMP_sres* PDU. If the result is the one expected by the verifier, the verifier considers the claimant an authenticated device. Optionally, the two devices may then exchange their roles as verifier and claimant and perform authentication in the opposite direction.

The above procedure occurs when the two devices have a common *link key*, which is used to operate on the random number. A device may maintain identical or separate link keys for every other device that it wants to authenticate. If a link key does not exist for a device, when an *LMP_au_rand* PDU is sent, the claimant will respond with an *LMP_not_accepted* PDU, giving the reason that the key is missing.

When a link key is missing, the devices need to be *paired* first. With the pairing process, an *initialization key* is generated and used to authenticate the devices and eventually to create a permanent link key for them. The initialization key is generated by entering a common *personal identification number* (PIN) in both of the pairing devices, which generates a temporary key. Note that neither keys nor the PINs that generate them are ever sent in clear form over the air. Using the temporary key generated by the PIN, the two devices may proceed with the authentication process as if they had a link key. The only difference is that pairing is initiated using an *LMP_in_rand* PDU instead of the *LMP_au_rand* PDU. Certain devices without a user interface may have a fixed, nonchangeable PIN. In this case, the PIN entered in a device with a user interface must match this fixed PIN; otherwise authentication is not possible.

Authentication of Bluetooth devices depends upon a shared secret. Commonly used public key and certificate schemes are not appropriate for ad hoc networks of personal devices. These other schemes require

the support of trusted authentication agencies and other third-party means of communication that cannot be assumed to be available in ad hoc networks. Authentication keys are 128 bits long and are constructed using the PIN (only for the initial authentication), a 128-bit random number, and a device's *BD_ADDR*.

Bluetooth devices may store link keys for future authentication without the need to enter a PIN each time. The two primary types of link keys are the *unit* keys and the *combination* keys. The former are derived from input parameters available in only one device, while the latter are derived by combining input parameters available in each of the two devices. The use of combination keys is preferred, as it provides for a stronger "bonding" between pairs of devices; a device must store a separate combination key for each other device it wants to authenticate using a stored link key. However, devices of limited storage capacity may store and use just a single link key for authentication of other devices.

Link Encryption

To protect the privacy of the data flowing over a Bluetooth link, the link can be encrypted. Encryption in Bluetooth wireless technology is based on a 1-bit stream cipher, whose implementation is included in the specification. The size of the encryption key, which changes with each BB_PDU transmission, is negotiable to match application requirements.

The encryption key is derived from the link key used to authenticate the communicating devices. This implies that prior to using encryption two devices must have authenticated themselves at least once. The maximum key size is 128 bits, but regulatory authorities in various countries may limit the permissible maximum key size. Encryption applies only to the payload of BB_PDUs and it is symmetric, in that both directions of communication are encrypted. Encryption is a link property in that both SCO and ACL packets over this link are encrypted.

A request for encryption starts with the *LMP_encryption_mode_req* PDU with an encryption mode parameter that distinguishes encryption of a link between two devices (point-to-point encryption), or encryption of broadcast packets as well. In the latter case, a *master* key is created that is to be used for encryption by multiple devices in the piconet. If the request for encryption is accepted, the devices then negotiate the size of the encryption key exchanging *LMP_encryption_key_size_req* PDUs. If the negotiation succeeds, the devices can initiate encryption by sending an

LMP_start_encryption PDU. Encryption starts by: (a) the master becoming ready to receive encrypted data; (b) a slave becoming ready to transmit and receive encrypted data; and (c) the master becoming ready to transmit encrypted data.

Power Management and Power-Managed States

Devices in a connected state can regulate their association with the piconet(s) to which they are connected to preserve power or to attend to other business such as participation in a scatternet. There are three low-power modes: sniff, hold, and park; all are optional.

Apart from modifying the property of the link between devices, a device optionally may request its communicating partner to adjust its transmission power depending on the quality of the link, as measured by the received signal strength of incoming transmissions. Power control for this case is discussed in Chapter 2 and is not discussed further here.

Sniff Mode

Typically a slave must listen at the beginning of each even-numbered slot to see whether the master will transmit to the device. In the *sniff* mode, a slave may relax this requirement for its ACL link. The master will transmit to the slave at a reduced duty cycle by starting its transmissions every T_{sniff} slave listening opportunities (master transmit slots). In particular every, say, $T_{sniff}[j]$ listening opportunities, slave j will listen for $N_{sniffAttempt}[j]$ slots for a transmission to it. If data is received, the slave will continue listening until the transmissions stop. The slave will continue listening for an additional $N_{sniffTimeout}[j]$ listening opportunities for additional transmissions to it. While a slave is in the sniff mode, any ongoing SCO transmissions will still occur as scheduled.

A master may force a slave into sniff mode with an *LMP_sniff* PDU. Alternatively, a master or a slave may request that the slave enter into sniff mode with an *LMP_sniff_req* PDU. Both of these LMP_PDUs carry the timing parameters defining the slave operation in sniff mode. These LMP_PDUs also carry an offset parameter D_{sniff} that determines the time of the first sniff instance. Subsequent sniff instances are separated by the period T_{sniff}. In the case of a request to enter the sniff mode, the master and the slave may negotiate the timing parameters for this mode.

Hold Mode

While sniff mode communications are periodic, in *hold* mode ACL communications with a slave are suspended in single installments for a time interval referred to as the *hold time.* While a slave is in hold mode, any ongoing SCO transmissions will still occur as scheduled.

A master may force a slave into hold mode with an *LMP_hold* PDU. Alternatively, a master or a slave may request that the slave enter into hold mode with an *LMP_hold_req* PDU. Both of these LMP_PDUs carry the initial value of the hold time. In the case of a request to enter the hold mode, the master and the slave may negotiate the timing parameters for this mode.

Park Mode

In the sniff and hold modes, a slave is still considered a fully qualified active member of the piconet and as such it maintains its *AM_ADDR* that was assigned to it when it joined the piconet. Note that while in these two modes, SCO transmissions with the slave are not interrupted.

To further reduce power consumption during periods of no activity, a slave may enter the *park* mode during which it disassociates itself from the piconet, while still maintaining timing synchronization with the piconet. By doing so, it can rejoin the piconet relatively quickly without having to perform inquiry and paging procedures.

A slave that enters the park mode gives up its *AM_ADDR.* On the other hand, to manage its fast and orderly readmission to the piconet, the master assigns to the slave two temporary 8-bit addresses, the *parked member address* (*PM_ADDR*) and the *access request address* (*AR_ADDR*). *PM_ADDR* is used to distinguish up to 255 parked devices (actually, slaves); the address '0x00' is a reserved *PM_ADDR.* Parked devices could be recalled using their 48-bit *BD_ADDR* if needed, but the use of the much shorter *PM_ADDR* allows for increased efficiency in recalling multiple parked devices.[11] *AR_ADDR* is used in scheduling the order of readmission of parked devices in a way that minimizes the possibility of collisions.

To maintain synchronization with the piconet and to facilitate the readmission of parked devices in the piconet, the master defines a low-bandwidth *beacon* channel, which consists of the periodic transmission of broadcast packets. Prior to entering park mode, slaves are notified of

11. When the value of *PM_ADDR* = '0x00' is assigned to a device, this device can only be unparked using its *BD_ADDR.* This could be the case when there are at least 255 parked devices in a single piconet, which is a rather exceptional case.

the timing parameters of the beacon transmissions, and thus they know when they can wake to receive possible transmissions from the master. Beacon transmissions destined to the parked stations are broadcast transmissions (BB_PDUs with *AM_ADDR* = 'b000'), for the simple reason that parked devices don't have an *AM_ADDR* and thus they cannot be addressed explicitly.

The timing parameters for beacon intervals are numerous. They include an offset parameter denoting the time of the first beacon transmission and the period of the beacon instances. They also include the broadcast repetition parameter, the interval following a beacon instant before the master gives the opportunity to parked devices to request to rejoin the piconet, the length of time that the master will continue giving this opportunity, and so on.

Just as in the sniff and hold modes, the master may force a slave into park mode with an *LMP_park* PDU. Alternatively, a master or a slave may request that the slave enter into park mode with an *LMP_park_req* PDU. When a slave is to rejoin the piconet, the master broadcasts in the beacon slots an *LMP_unpark_BD_ADDR_req* or an *LMP_unpark_PM_ADDR_req* PDU, depending upon whether the parked device is addressed with its 48-bit *BD_ADDR* or its 8-bit *PM_ADDR*. Multiple parked devices can be invited with a single unpark LMP_PDU. In the unpark LMP_PDUs, the master also includes the new *AM_ADDR* to be assigned to the parked device when it rejoins the piconet as a slave. Note that a parked device cannot be invited to rejoin a piconet when there are already seven active slaves in the piconet. In the latter case, the master may have to park some active devices and free the corresponding *AM_ADDR*s prior to unparking a parked device.

Bandwidth-Conscious Communications

Bluetooth devices have several options available for managing the bandwidth that is allocated between them. As mentioned earlier, devices may use SCO links whose high-priority periodic transmissions provide for telephony-quality voice communications. Transmissions on ACL links can also be provided with bandwidth guarantees through polling interval restrictions. Support for SCO links is optional, while support for ACL polling interval transactions is mandatory.

Furthermore, to increase the efficiency of the transmission and thus increase the bandwidth supported on an ACL link, link managers may negotiate the use of larger BB_PDUs. Support for this link manager transaction is mandatory; the actual use of larger BB_PDUs needs

to be coordinated with additional LMP transactions described below. Finally, devices may dynamically change the encoding schemes of the transmissions to take better advantage of the link quality conditions. Support for this link manager transaction is optional. The latter two transactions are not discussed further.

SCO Links

A piconet supports up to three SCO links for telephony-quality (64 Kbps) voice communications. SCO packets transmitted on the SCO links are of high priority and can preempt any other activity that the device may be involved with, like pages and inquiries, holds, and so on. A device requests the establishment of an SCO link with an *LMP_SCO_link_req* PDU. This LMP_PDU contains the audio parameters like the period T_{SCO}, the SCO BB_PDU type and the air-mode, which represents the voice codec type. Supported codec types are: 64 Kbps μ-law or A-law PCM format, or 64 Kbps CVSD (continuous variable slope delta) modulation format. With such a high resolution of voice traffic, cellular phone conversations carried to, say, a headset that is connected with a cellular phone over a Bluetooth SCO link should not degrade in quality.

When the master responds positively to a slave's *LMP_SCO_link_req* PDU or sends its own *LMP_SCO_link_req* PDU, it supplies the offset D_{SCO} that identifies the time of the first transmission for the new SCO link, and an SCO link unique identifier, or handle. Either the master or the slave can subsequently request to change the parameters of the SCO link, using the *LMP_SCO_link_req* PDU again, or to remove the link altogether, using an *LMP_remove_SCO_link_req* PDU. The latter transactions make use of the SCO link handle to identify the particular SCO link whose parameters or status is to be changed.

Quality of Service (QoS) for ACL Links

To control the minimum bandwidth assignment for ACL traffic between two devices, or equivalently the maximum access delay of ACL BB_PDUs, the maximum *polling interval*[12] for a slave can be adjusted as needed. A master can enforce a new maximum polling interval using an *LMP_quality_of_service* PDU. In this case, the slave cannot reject the adjustment of the polling interval determined by the mas-

12. Recall that a master polls a slave either explicitly through the use of a POLL BB_PDU or implicitly by simply transmitting any payload carrying BB_PDU.

ter. On the other hand, a master or a slave may request a change in the polling interval with an *LMP_quality_of_service_req* PDU. In this case, the request can be either accepted or rejected by the other party.

Both of these LMP_PDUs also carry the number of times, N_{BC}, that each broadcast packet is to be repeated. Since this parameter relates to the operation of the entire piconet, and not just to the ACL link between a slave and the master, N_{BC} is meaningful only when it is sent from the master. When this parameter is included in a request LMP_PDU from a slave, it is ignored.

Link Controller Management

The transactions in this section apply to negotiation of parameters related to the link controller and the baseband protocols, such as negotiation of the paging scheme, timing accuracy, master-slave role switch and so on.

Paging Scheme

When two devices have already communicated with each other, they may subsequently reconnect with each other more rapidly, since they can bypass the inquiry process. However, during an inquiry response, a slave provides paging information to the master in an FHS packet. When bypassing the inquiry process, this FHS packet is also bypassed. Thus, if a device has modified its paging parameters, perhaps to switch to an optional paging scheme or to change its $T_{pageScan}$ interval, the master will not become aware of the change.

With the paging scheme LMP_PDU transaction, devices can announce or even negotiate the paging scheme to be used the next time these devices page each other. With an *LMP_page_mode_req* PDU, the requesting link manager suggests to the other link manager the paging scheme to be used when the requesting device pages the other. Likewise, with an *LMP_page_scan_mode_req* PDU, the requesting link manager suggests to the other link manager the paging scheme to be used when the other device pages the requesting device. Rejecting any of these request LMP_PDUs implies that the current paging scheme is not to be changed. However, a request to change to the mandatory paging scheme cannot be rejected. Support for this transaction is optional.

Master-Slave Role Switch

A paging device becomes the default master of the resulting piconet, although circumstances may necessitate that the roles of the master and the slave be exchanged. For example, such an exchange is needed to implement the LAN access profile using PPP (described in Chapter 15). To initiate the process of master-slave role switch, a device sends an *LMP_switch_req* PDU, which upon acceptance will initialize the role switch. The switch basically comprises the process of migrating from the master/slave transmit timing sequence in the piconet of the old master (the new slave) to the master/slave transmit timing sequence in the piconet of the new master (the old slave). Support for a master-slave role switch is optional.

Clock and Timer Information

A device can request updated clock information from another device to optimize various link controller operations.

An *LMP_clock_offset_req*[13] PDU sent by a master results in a slave returning the most current offset between the native clocks of the slave and the master as registered by the slave. This information can be used to optimize the paging time when the master pages the slave again in the future. Support for this transaction is mandatory.

An *LMP_slot_offset* PDU carries with it the slot offset, in microseconds, between the start of the time of a transmit slot from the master and the corresponding transmit slot from the slave, if the slave were to be a master. This information is used to optimize a master-slave role switch. Support for this LMP_PDU is optional.

An *LMP_timing_accuracy_req* PDU results in returning the long-term *jitter*, in microseconds, and *drift*, in parts per million (ppm), of the receiving device's clock. This information is used to optimize the wake time for a device after a long period of inactivity while still being associated with a piconet, such as waking after hold mode, or waking prior to a master's transmission at a sniff slot or a beacon slot for a parked device. Support for this LMP_PDU is optional; when it is not supported the jitter and accuracy are assumed to be at their maximum value of 10 µsec and 250 ppm, respectively.

An *LMP_supervision_timeout* PDU sent by a master includes the link supervision timeout value used by a link controller to detect the

13. The name of this LMP_PDU is actually *LMP_clkoffset_req*.

loss of a Bluetooth link between the master and a slave. Support for this LMP_PDU is mandatory.

Information Exchange

Link managers typically exchange information about each other to better coordinate their interaction. The *LMP_version_req* PDU contains the LMP version supported by the sender of this PDU. The receiver of this LMP_PDU returns the *LMP_version_res* PDU which contains its own supported LMP version. The version number is provided as a triplet [*versionNo*:*companyID*:*subVersionNo*]. The version number part of the triplet is a version of LMP as defined by the SIG. The subversion number is relative to a company that may have its own particular implementation of the protocol. Support for this LMP_PDU transaction is mandatory.

The *LMP_features_req* PDU contains the optional radio, baseband, and link manager features supported by the sender of this LMP_PDU. The receiver of this LMP_PDU returns the *LMP_features_res* PDU, which contains its own supported features. These features include the supported packet types, other than the mandatory FHS, NULL, POLL, DM1 and DH1; supported power control modes, voice codecs, encryption, role switch, optional paging schemes and so on. Support for this LMP_PDU transaction is mandatory.

With the *LMP_name_req* PDU, the requesting link manager requests the user-friendly name of the device that receives this LMP_PDU. The user-friendly name is the name that a user of a device assigns to it. Within a device a name is encoded in UTF-8 and it can be up to 248 bytes long. Since the name of a device can be longer than a single DM1 packet, when a device requests the user-friendly name of another device it also provides an offset parameter that the responding device can use to transmit the proper segment of its name, using the *LMP_name_res* PDU.

The UTF-8 encoding [IETF96] for device names has been selected for its support of international languages because the Bluetooth wireless technology is targeted for worldwide use. UTF-8 characters are encoded using sequences of 1 to 6 bytes. For compatibility with the widely used ASCII character set, ASCII characters are encoded using a one-byte UTF-8 character, which has the same value as the corresponding ASCII character. Hence, the user-friendly name of a Bluetooth device can be up to 248 ASCII characters long.

Connection Establishment and Link Detachment

LMP is a transport protocol for control information between link managers in devices. LMP does not encapsulate PDUs of any higher communication layer. As such, LMP transactions may occur without involvement of any higher layer, such as L2CAP or the host itself. When a host application in a device wants to communicate with one in another device, the first device sends an *LMP_host_connection_req* PDU which the receiving device will either accept or not. If the connection request is accepted, the two link managers will negotiate the parameters of the link, like authentication and QoS. When link managers finish negotiating, each sends an *LMP_setup_complete* PDU. Only after both link managers have issued the *LMP_setup_complete* PDU can communication that does not involve LMP_PDUs commence between the devices.

When any device wants to terminate its link with another device, it issues an *LMP_detach* PDU with a reason parameter explaining why the link is to be terminated. The *LMP_detach* PDU cannot be contested and the link between the two devices will be terminated immediately.

Support for the LMP_PDUs in this section is mandatory.

Summary

In this chapter we have highlighted the lower Bluetooth transport protocols: radio, baseband, and link manager. These protocols define the operational characteristics of the Bluetooth wireless technology and instantiate the Bluetooth air-interface. Bluetooth devices use relatively high rate frequency-hopping spread-spectrum transceivers operating over 79 (or 23) 1 MHz channels in the 2.4 GHz ISM RF band. Devices find others in their vicinity using inquiries and request connections to those devices by explicitly paging them. Communicating devices are organized in piconets composed of a master and one or more slave devices. Device transmissions are organized over a TDD transmission time axis, with the slaves transmitting only when they have been first addressed by the master. Bluetooth links between devices can support both asynchronous and synchronous transmissions, and they can be authenticated, encrypted, and conditioned based on QoS demands.

The next chapter presents the upper transport protocols, which aim to conceal the details of the lower transport protocols from higher layers and applications. This allows the higher layers to use the Bluetooth links in a manner that is agnostic of the way that connections and

transmissions between devices are organized and managed over the Bluetooth air-interface.

7

The Upper Protocols of the Transport Group

The lower transport protocols are optimized to deal with the hostility of the RF transmission medium, user requirements for low cost and power consumption, security concerns, regulatory issues, and so on. These design points have led to, among other things, selection of the TDD, polling-based medium access protocol at the baseband. While this approach is suitable for operation of Bluetooth systems in the 2.4 GHz ISM band, the small size of a BB_PDU does not fare so well with the much larger packet sizes typically encountered with Internet and other similar traffic.

It was thus recognized early on that an adaptation layer was needed to move larger upper-layer PDUs and smaller lower-layer PDUs back and forth among the protocol stack layers. This adaptation layer was originally called *level 2 medium access control* (MAC-2), at a time when MAC-1 represented the medium access control protocol at the baseband level. But the name MAC-1 never prevailed, and so MAC-2 did not apply either. It was therefore decided late in the summer of 1998 to change the name from MAC-2 to a more descriptive one, and the name *logical link control and adaptation protocol* (L2CAP) came into being. Incidentally, shortly after the first L2CAP draft specification was produced in September 1998, a portion of that specification was split out into its own independent document, spawning a brand-new activity in the SIG. Specifically, the Bluetooth discovery protocol (BDP) section of the MAC-2 and early L2CAP specifications grew in scope and resulted in the service discovery protocol (SDP) specification high-

lighted in the next chapter. This chapter describes L2CAP, which is certainly one of the most important layers in the stack.

In the reference implementation shown in Figure 6.1 in the previous chapter, the L2CAP layer[1] is a software component that resides within a host. To permit L2CAP and higher protocol layers and applications to transfer and receive control and application data to and from any vendor's Bluetooth module in a standardized way, the SIG has developed a protocol that allows the host to communicate to the module in an interoperable manner. This protocol is the host controller interface (HCI), supplemented with a series of HCI transport protocols, which are mechanisms used to transfer HCI data across various physical connectors, like USB ports, RS-232 ports, and so on. This chapter also discusses HCI and its transport protocols.

Figure 7.1 is a refinement of Figure 6.1 and depicts the placement of the transport protocols in a reference implementation of a Bluetooth device. Note that the link manager and host I/O block in Figure 6.1 has been separated into two logical components, which may nevertheless run on the same firmware platform. The host controller interacts with both the host and the Bluetooth module hardware and firmware components to transfer data between them. On the host side, the *HCI layer* executes the HCI communications protocol to carry data to and from the module (through the host controller) and the host itself.

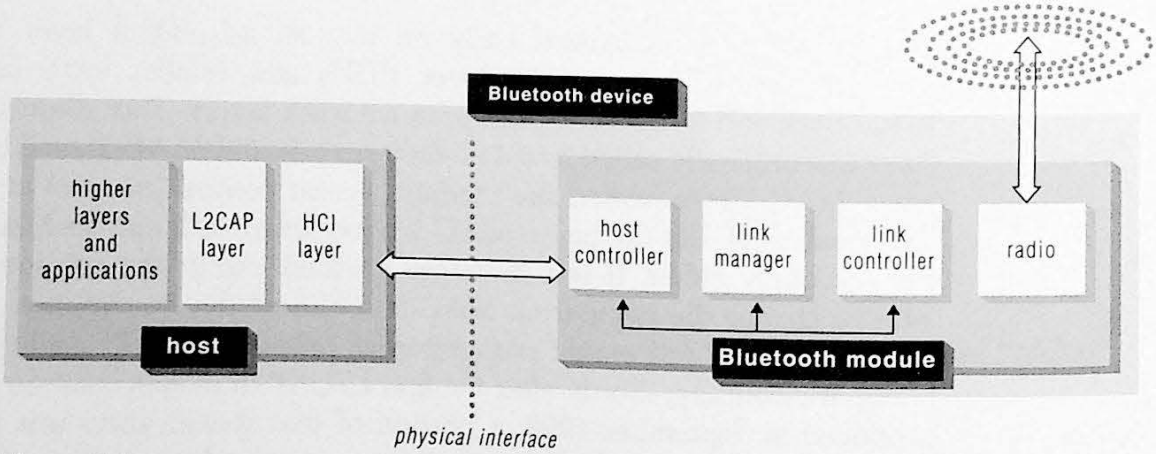

Figure 7.1
The transport protocol group placement in a Bluetooth reference implementation.

1. This layer is more appropriately called "L2CA layer" or "L2CAL," but since L2CAP is phonetically more pleasing than either L2CA or L2CAL, the technically inappropriate usage of the term "L2CAP layer" or even just "L2CAP" has tacitly prevailed. Here, the use of the term "L2CAP" as a noun is reserved primarily for the protocol used to communicate between L2CAP layers in different devices.

The L2CAP Layer

The primary role of the L2CAP layer is to hide the peculiarities of the lower-layer transport protocols from the upper layers. By doing so, a large number of already developed higher-layer transport protocols and applications can be made to run over Bluetooth links with little, if any, modification.

The L2CAP layer concerns itself only with asynchronous information (ACL packet) transmission. Its packets, referred to as L2CAP_PDUs, are carried on ACL BB_PDUs whose *L_CH* field in the payload header has the value 'b10', denoting the start of an L2CAP_PDU, or 'b01', denoting the continuation of an L2CAP_PDU (see Table 6.5 in the previous chapter). Even though L2CAP_PDUs are closely associated with ACL BB_PDUs, the lower transport protocol concepts of master, slave, polling, frequency hopping sequences, native clocks, and so on are meaningless at the L2CAP layer. The lower transport layers provide the equivalent of a packet interface to L2CAP over which L2CAP sends and receives its data and control messages, but L2CAP and higher layers are otherwise insulated from the lower transport protocols.

The L2CAP layer supports higher-layer protocol multiplexing, compensating for the lack of such support at the lower transport layers. Furthermore, it facilitates the segmentation and reassembly of larger-size, higher-layer packets to and from the smaller baseband packets. Since no knowledge of BB_PDUs exists at the L2CAP layer, not to mention knowledge of their transmission size, the L2CAP layer itself does not perform segmentation and reassembly of lower-layer PDUs. However, it facilitates these operations by providing L2CAP_PDU length information in its PDUs, allowing a reassembly engine to verify that it has reconstructed the PDU correctly. Moreover, the L2CAP layer exports maximum-packet-size information to higher layers that informs them of the largest packet size that L2CAP layers in other devices can handle. It is the responsibility of the higher layer to fragment its information into packets that do not violate the L2CAP maximum packet size.

In addition, the L2CAP layer supports the exchange of quality-of-service (QoS) information, which aids in controlling the transmission resources in a way that supports the expected QoS. Finally, the L2CAP layer also provides a group abstraction to higher layers. This allows mapping groups of higher-layer protocol addresses into piconets without exposing the concept of a piconet to the higher layers.

The L2CAP layer assumes that the underlying transmission facilities (that is, the lower transport protocols) provide a full-duplex communication channel that delivers L2CAP_PDUs in an orderly manner. L2CAP itself does not provide any mechanisms for securing the reliable transmission of its PDUs. Instead, it relies upon the retransmission process at the baseband layer to support a sufficiently reliable communication channel for higher layers.

L2CAP Channels

Communication between L2CAP layers is based on logical links, called *channels*, through which L2CAP traffic flows between *endpoints* within each device. Each endpoint of a channel is assigned a unique *channel identifier* (CID). A CID is a 16-bit identifier that is locally administered. An L2CAP layer assigns a different CID to every channel endpoint used therein.[2]

2. Strictly speaking, "local" CIDs assigned by an L2CAP layer need to be unique only within the set of channels that this L2CAP layer establishes with a single "remote" L2CAP layer.

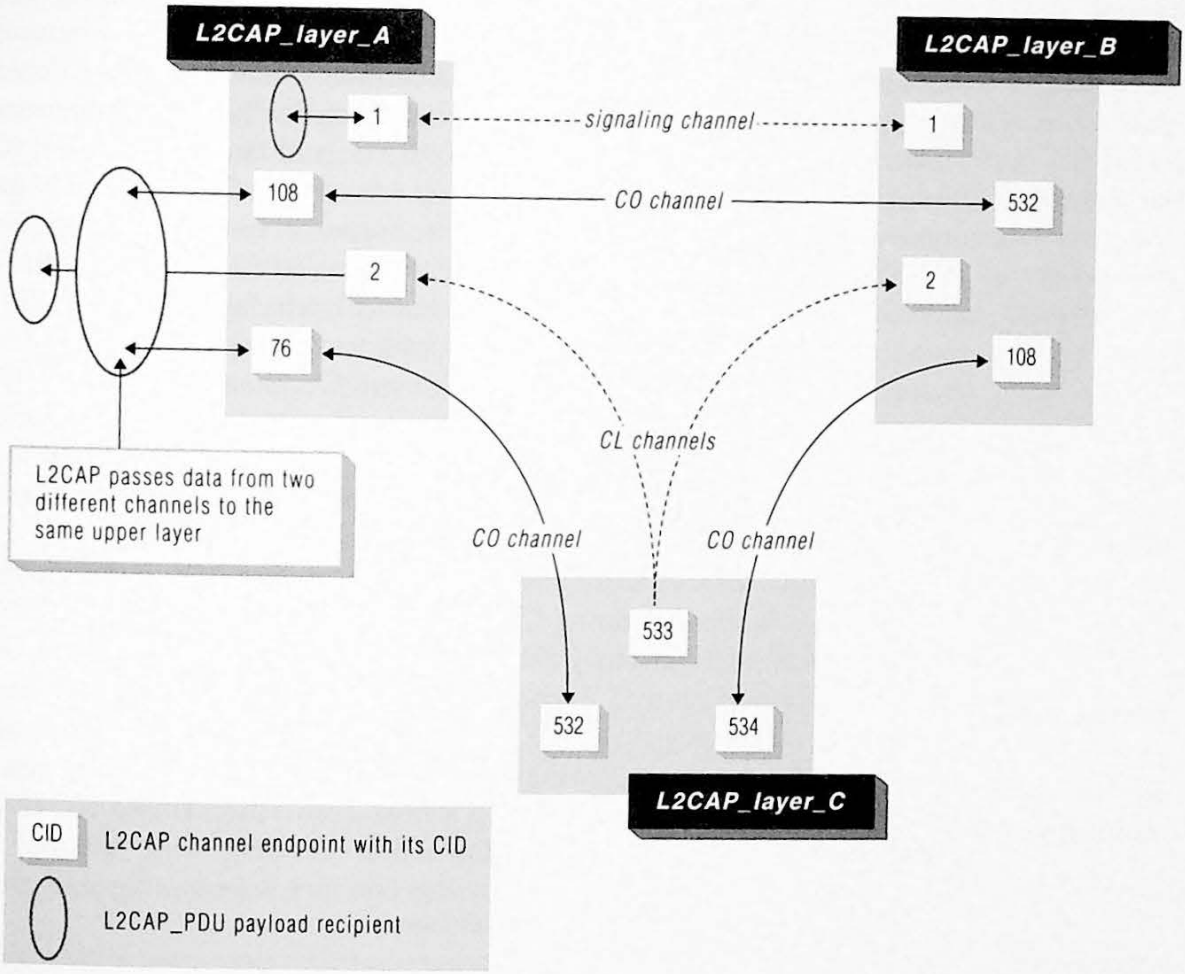

Figure 7.2
L2CAP channels; although not shown, every endpoint is associated with a payload recipient entity.

Figure 7.2 shows the L2CAP layers in three devices exchanging information using L2CAP channels. Each channel terminates at an endpoint within the L2CAP layer. Each L2CAP endpoint is uniquely associated with a *payload recipient entity* to which the payload of the L2CAP_PDU is directed for additional processing. The payload recipient entity may reside within the L2CAP layer itself and be used for signaling purposes between communicating L2CAP layers. The payload recipient entity may also reside above the L2CAP layer, in which case it represents the higher layer whose PDUs are transported across Bluetooth links by the L2CAP layer.

The figure identifies the various types of L2CAP channels currently defined. There are persistent *connection-oriented* (CO) channels that are used for bidirectional communications; these require a connection signaling exchange prior to being established. There are ephemeral *connectionless* (CL) channels that are unidirectional and can be used for broadcast transmissions to groups of devices. Since these channels are unidirectional, a device that needs to respond to a connectionless L2CAP transmission must use another channel to do so. Finally, there are signaling channels that are used primarily to exchange control information that is used to establish and configure CO channels. Signaling channels borrow features from both CO and CL channels. Just like CL channels, they do not require an explicit connection establishment prior to commencing communications over them, but they are persistent and bidirectional in that information flows in both directions over the same signaling channel.

A number of CIDs are reserved for well-defined, globally known channels, while the remaining CIDs are administered as needed. Table 7.1 summarizes the various CID types. Between two devices there can be many CO and many CL channels, but only one signaling channel, since there exists only one CID that can be assigned to both endpoints of the signaling channel. Owing to this restriction, each signaling channel can be viewed as a CO channel whose connection phase has been eliminated because the CID information defining that channel is already known to the devices; thus the channel is preconfigured with fixed parameters.

Table 7.1
The CID assignment types.

CID	Comments
'0x0000'	null identifier, not to be assigned
'0x0001'	CID for both endpoints of an L2CAP signaling channel
'0x0002'	CID for the destination endpoint of a CL L2CAP channel
'0x0003'–'0x003F'	range of reserved CIDs
'0x0040'–'0xFFFF'	range of CIDs allocated on demand by a device to its local endpoints for either CL or CO L2CAP channels

At the outset the MAC-2 protocol (L2CAP's predecessor) was to provide only connectionless services to the upper layers. Over time it was recognized that for this layer to be extensible and to remain relevant and amenable to future enhancements (such as support for quality of service), it needed to provide configurable traffic services. The latter need led to the development and inclusion of configurable connection-oriented services in the L2CAP layer. Today this connection-oriented service is the primary one provided by the L2CAP layer. In version 1.0, only the TCS-based telephony profiles (described in Chapter 13) require the use of CL L2CAP channels.

L2CAP_PDU Types

There are two types of L2CAP_PDUs: the first is used with CO channels and the second with CL channels. Signaling L2CAP_PDUs are formed according to the former type.

The Connectionless (CL) L2CAP_PDU Type

Table 7.2 summarizes the fields of a CL L2CAP_PDU in the order of transmission. Note that each header field uses a little-endian byte order with the least significant byte transmitted first; all fields in the L2CAP_PDU headers follow this little-endian transmission convention.

Table 7.2
The format of a connectionless L2CAP_PDU.

field	size	comments
L2CAP_PDU_CL_Header		
Length	2 bytes	total length in bytes of this L2CAP_PDU excluding the *Length* and CID fields (≥ 2)
Destination_CID	2 bytes	indicates the CID of the destination endpoint of the L2CAP channel used for this transmission: has fixed value '0x0002'
PSM	≥ 2 bytes	protocol and service multiplexer
L2CAP_PDU_CL_Payload		
Payload	*(Length – PSM_field_size) bytes*	CL L2CAP_PDU payload data; the maximum possible size is 65,535 bytes minus the size of the *PSM* field, which typically is 2 bytes

The PSM field provides the means for identifying the higher-layer recipient of the payload of this connectionless L2CAP_PDU. It is discussed in more detail below along with the signaling L2CAP_PDUs for CO channels. The minimum supported value for *maximum transmission unit* (MTU) for payloads in CL L2CAP_PDUs is $MTU_{CL} = 670$ bytes, unless explicitly stated otherwise, say, in a profile specification.[3]

The Connection-Oriented (CO) L2CAP_PDU Type

Table 7.3 summarizes the fields of a CO L2CAP_PDU in the order of transmission. As mentioned earlier, each header field uses a little-endian byte order with the least significant byte transmitted first.

Table 7.3
The format of a connection-oriented L2CAP_PDU.

field	size	comments
L2CAP_PDU_CO_Header		
Length	2 bytes	total length of this L2CAP_PDU excluding the *L2CAP_header* (≥ 0)
Destination_CID	2 bytes	indicates the CID of the destination endpoint of the L2CAP channel used for this transmission
L2CAP_PDU_CO_Payload		
Payload	*Length* bytes	CO L2CAP_PDU payload data; the maximum possible size is 65,535 bytes

Comparing the headers of the CL and CO L2CAP_PDUs, one may notice a subtle difference in the definition of the corresponding *Length* fields. The PSM field in a CL L2CAP_PDU is treated as if it were part of the payload, hence its size is accounted for when calculating the *Length* field. This difference is intended to maximize code reuse when processing L2CAP_PDUs of either type.

The MTU for payloads in CO L2CAP_PDUs, MTU_{CO}, as well as other parameters of a CO channel, are negotiated during the connec-

3. The various L2CAP MTUs discussed in this chapter apply only to the payload section of the corresponding L2CAP_PDUs. Thus, the MTU information is used by the payload recipient entities at the endpoints of an L2CAP channel, see Figure 7.2, to adjust the maximum size of PDUs that they can send or receive over this L2CAP channel.

tion establishment phase using the L2CAP signaling discussed in the following section.

L2CAP Channel Management: The L2CAP Signaling

Connection-oriented, point-to-point L2CAP channels use signaling to become established, to be configured, and to be terminated. During the connection establishment phase, the payload recipient entity in an upper layer of L2CAP_PDUs is identified and the endpoint CIDs for the newly formed channel are exchanged between the two communicating L2CAP layers. During the connection establishment phase, the properties for each direction of transmission on the channel are negotiated and agreed upon. These properties include the payload MTUs, the reliability level of transmissions between the involved devices, and QoS.

L2CAP signaling consists of request and response PDU transactions carried over the signaling channel whose endpoints both have the reserved CID value '0x0001'. A device, referred to as the *local* device, sends an L2CAP signaling request PDU (for example, to create or configure a channel) to a *remote* device, and the remote device responds to the local device's request with a corresponding L2CAP signaling response PDU. Each transaction is identified by a *transactionID* that matches a request from a local device with the subsequent response from the remote device. Contrary to other L2CAP_PDU transmissions, L2CAP signaling transactions are reliable in that a local device may retransmit a request if a response is not received within a timeout period. The decision about whether or not to retransmit a request, as well as the value of the timeout period, are implementation dependent. The timeout period is at least 1 second and can be up to 60 seconds. A local device may wait up to an additional 300 seconds to receive the final response if the remote device has responded indicating that it has received an initial request but needs additional time to complete its processing. For example, a request for connection may require a device authentication to occur first, which in turn, may require the device to enter a PIN as discussed in Chapter 6.

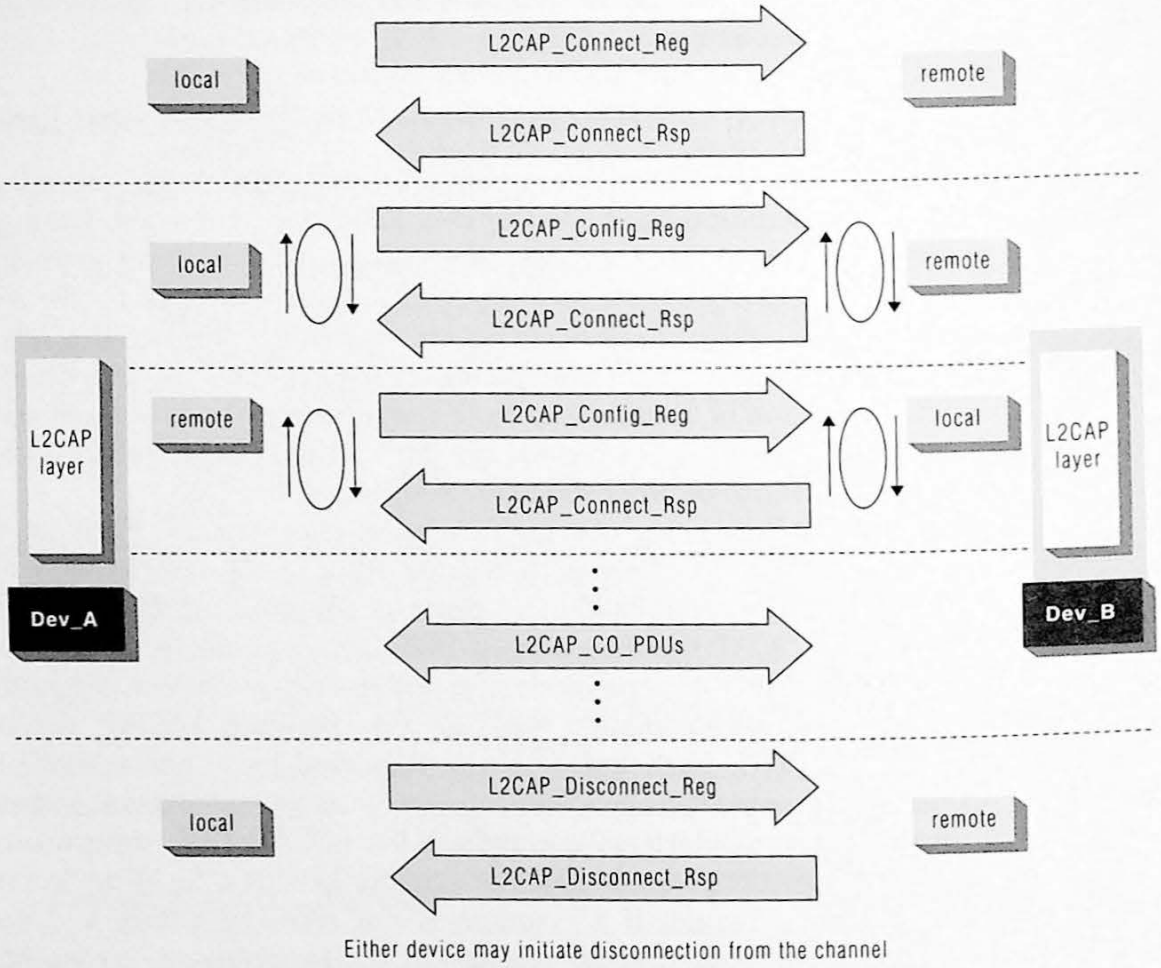

Figure 7.3
The management and data-exchange phases of a CO channel and the local/remote device roles.

Figure 7.3 shows a typical sequence of L2CAP signaling transactions between a local and a remote device. This sequence starts with the L2CAP layer of the local device transmitting a request for the creation of a channel between the devices. The remote device responds by either accepting or rejecting the request. Upon acceptance of the request and establishment of the channel, the local device initiates configuration of the channel for one direction of communication. The configuration parameters are dictated by the implementation limitations of the L2CAP layer, including the L2CAP MTU that can be used on this channel and the QoS requirements of the applications that will be using this channel. The remote device responds positively or negatively to the local device's configuration parameters. A negative response causes a

configuration negotiation phase between the devices until they both agree upon the configuration parameters. Following the termination of configuration in one direction, the devices exchange their local and remote roles and the configuration process continues likewise in the opposite direction. Upon termination of the configuration phase, the channel is ready to receive and transport PDUs from higher layers over the newly established channel. Subsequent channel reconfiguration could happen at any time that the channel is active and could be initiated by either device. The L2CAP channel terminates when either device initiates the termination phase.

The header of the L2CAP signaling PDUs resembles that of a CO L2CAP_PDU, with the destination endpoint CID field having the reserved CID value '0x0001'. The payload of the signaling PDUs consists of a collection of *signaling commands*. All L2CAP implementations must be able to accept signaling PDUs with an MTU_{SIG} of 48 bytes. Table 7.4 summarizes the fields of a signaling command.

Table 7.4
The format of an L2CAP signaling command.

field	size	comments
L2CAP_Signaling_Command_Header		
Code	1 byte	identifies the command type
Identifier	1 byte	identifies the signaling transaction, transactionID, for matching responses to corresponding requests; retransmitted commands use the same identifier
Length	2 bytes	total length of the payload portion of this command (≥ 0)
L2CAP_Signaling_Command_Payload		
Payload	*Length* bytes	signaling command payload data (collection of signaling commands)

Signaling commands with an unsupported code result in an *L2CAP_Command_Reject* command (*Code* = '0x01') being transmitted in the reverse direction.

The *L2CAP_Connection_Request* Signaling Command

When a local device wants to establish a CO L2CAP channel with a remote device, it forms and sends an *L2CAP_Connection_Request* signaling command (Code = '0x02') to the remote device. The significant fields of the command are summarized in Table 7.5; note that the table does not show all the fields of the command and must be interpreted with respect to the signaling-command format in Table 7.4.

Table 7.5
The *L2CAP_Connection_Request* signaling command.

field	size	comments
L2CAP_Connection_Request_Command_Payload		
PSM	≥ 2 bytes	identifies the destination payload processing entity for the L2CAP_PDUs transmitted to the remote device over the requested channel
Source_CID	2 bytes	the CID for the requested channel endpoint in the local device

If and when the channel is established, the remote device will use the value in the *Source_CID* field as the *Destination_CID* (see Table 7.3) to send L2CAP_PDUs to the local device over this CO channel.

The *PSM* (protocol and service multiplexer) field is a variable-size field with a minimum (and typical) size of 2 bytes. It is used for multiplexing several protocols over the L2CAP layer. In particular, the local device uses the *PSM* field to inform the remote device about where to direct the payload of the L2CAP_PDU transmissions over this channel.

Although the *PSM* field is used for multiplexing middleware protocols that use L2CAP's transport services, like SDP, RFCOMM, TCS, and so on, it goes a step further. PSM could also identify one of many implementations of a protocol layer, or even an entire protocol stack, that reside above L2CAP. For example, a manufacturer could implement two independent RFCOMM layers within a single device. In this case, multiplexing simply on the name of the RFCOMM protocol would not be very helpful. The *PSM* field aids in distinguishing the two RFCOMM implementations by assigning a different *PSM* value to each of them.

The range of values for the *PSM* field is divided into two regions. The first region covers the *PSM* values up to '0x1000' and consists of reserved values that represent established, well-known protocols that directly use L2CAP, such as SDP (*PSM* = '0x0001') or RFCOMM (*PSM* = '0x0003').[4] The region of *PSM* values above '0x1000' is used to identify multiple implementations of a given protocol above L2CAP (as described above), protocols not yet standardized, experimental protocols under development, and so on. The *PSM* value for a given connection can be retrieved from service discovery records using SDP as described in the next chapter.

Originally, the *PSM* field was just a *Protocol* field, used to identify a specific protocol that could be multiplexed over the L2CAP layer. The L2CAP working group in the SIG realized that enhanced system security requires the ability to identify not just the protocol used but also the application associated with the connection request. In this respect, the L2CAP layer was recognized as the natural choice in which to add any security safeguards. But an L2CAP connection request using just the *Protocol* field could not identify the application that would ultimately use the connection. Thus, the PSM field was introduced to identify a protocol stack from the L2CAP layer all the way up to the service provided by the application that would use the newly formed channel. Eventually, the group consensus about security at the L2CAP layer took a more general path and is highlighted in [Muller99]. However, the *PSM* field did not revert back to the *Protocol* field. It was recognized that the added multiplexing flexibility that the *PSM* field introduced was too powerful to ignore. Thus, the use of the *PSM* field has persisted in the specification, although its name is now somewhat outdated, and it can be used to multiplex not only protocols, but multiple stack implementations as well. This feature is truly unique to the L2CAP layer.

The *L2CAP_Connection_Response* Signaling Command

Following the receipt of an *L2CAP_Connection_Request* command, the remote device returns an *L2CAP_Connection_Response* command (Code = '0x03') to inform the local device whether or not it accepts the connection request. The remote device could also return a *connection pending* response to inform the local device that the connection request has been received but no decision has been made yet; the remote L2CAP

4. The values of the *PSM* field are always odd, thus providing a simple means for extending the *PSM* field beyond its typical size of 2 bytes.

layer could, for example, await the result of an authentication procedure before deciding to accept or reject the connection. If a connection is refused, the remote device provides the reason(s) for doing so, which could include security issues, resources not available, or *PSM* value not supported. Finally, the remote device also returns the *Destination_CID* that identifies the CID of the channel endpoint located in the remote device. This CID is meaningful only if the connection is accepted, and it is used as the *Destination_CID* field of the CO L2CAP_PDUs that the local device subsequently sends to the remote device over this CO channel.

The *L2CAP_Configuration_Request* Signaling Command

Following channel establishment, the channel needs to be configured. The channel configuration transaction is mandatory; however, empty configuration commands may be sent when nothing needs to be configured. Configuration transactions for a channel may occur again at a later time after normal communications have commenced for the channel. The key fields of the *L2CAP_Configuration_Request* command (*Code* = '0x04') are summarized in Table 7.6.

Table 7.6
The *L2CAP_Configuration_Request* signaling command.

field	size	comments
L2CAP_Configuration_Request_Command_Payload		
Destination_CID	2 bytes	for the channel to be configured, identifies the CID of the channel endpoint in the device that receives this command (the remote device)
Flags	2 bytes	in version 1.0, only the LSB is used to signify that additional configuration options are to follow in a subsequent configuration command from the local device
Config_Options	variable	configuration options for the channel to be configured

The LSB of the *Flags* field, referred to as the *C* bit, is used as a continuation flag for additional configuration options to follow in subsequent configuration commands. Segmentation of the configuration

options in multiple configuration commands may be necessitated by the small MTU_{SIG} value.

Before examining the configuration options, we discuss the corresponding response command from the remote device, which also contains a similar set of configuration options.

In the *L2CAP_Configuration_Response* command (*Code* = '0x05') that corresponds to an *L2CAP_Configuration_Request* command sent from a local device,[5] the responding (remote) device states whether or not it agrees with the values of the various configurable parameters in the *L2CAP_Configuration_Request* command. If a configuration option in an *L2CAP_Configuration_Request* command is not supported by the remote device, it rejects the request and includes in its response a copy of the unsupported option(s).

When an *L2CAP_Configuration_Request* command is rejected due to disagreement with the values of any of the configurable parameters, the remote device may either terminate the configuration process or provide the values of the parameters that could have been acceptable to it. In this case, the local device may submit a new *L2CAP_Configuration_Request* command with the configuration parameter values readjusted; the number of successive resubmissions of L2CAP_Configuration_Request commands is implementation dependent. If no agreement is reached between the local and remote devices, the current configuration parameters remain in effect, or the channel is terminated. The duration of configuration negotiations is implementation dependent, but the maximum time is 120 seconds.

Note that any response from the remote device regarding a configuration option is exclusively for the direction of traffic flow implied by the *L2CAP_Configuration_Request* command. For example, if the configurable parameter in an *L2CAP_Configuration_Request* command refers to traffic leaving the local device, an outgoing traffic parameter, any reference to this parameter by the remote device will refer to the same traffic parameter arriving at the remote device, an incoming traffic parameter. Following termination of configuration transactions in one direction, the role of the local and remote devices is reversed once and configuration of the channel parameters in the opposite direction may commence. This gives the other device (the former remote device) the opportunity to configure its traffic parameters as well.

5. Recall that we have defined a local device to be the device that originates an L2CAP signaling transaction, which starts with the transmission of a request command. The responding device is the remote device.

The Configuration Options

Table 7.7 lists the communications parameters that can be adjusted through configuration. The interpretation of these parameters is relative to the local device–that is, the device that originates the *L2CAP_Configuration_Request* command and contains the configuration option related to these parameters.

Table 7.7
The configuration options.

parameter	comments
MTU_{CO}	identifies the maximum L2CAP_PDU payload, in bytes, that the local device can accept over this channel; minimum $MTU_{CO} = 48$ bytes, default $MTU_{CO} = 672$ bytes[1]
Flush_Timeout	identifies the amount of time, in multiples of 1.25 milliseconds, during which the link manager of the local device will continue attempting to transmit the baseband segments from an L2CAP_PDU, prior to discarding it; by default the *Flush_Timeout* value indicates a reliable channel where PDUs are retransmitted for as long as required, or until the link is declared lost
QoS	identifies the traffic-flow specification for the local device's traffic (outgoing direction) over this channel

1. While the minimum MTU_{CO} was chosen to accommodate buffering capabilities of "simple" devices, such as headsets, the default value was chosen so that it can conveniently fit within two DH5 BB_PDUs.

The *QoS* option includes a *Service_Type* parameter that determines whether or not traffic will be sent in the outgoing direction by the local device. If yes, then the *Service_Type* further determines if the outgoing traffic will be treated by the local device as *best effort* or *guaranteed.* Best-effort traffic is the default and mandatory service type supported by any L2CAP-layer implementation. With the guaranteed service type, the local device provides the QoS parameters associated with its outgoing traffic flow. These QoS parameters consist of a subset of the ones specified in [IETF92] and include: *Token_Rate*, *Token_Bucket_Size*, *Peak_Bandwidth*, *Latency*, and *Delay_Variation.* All of these parameters have default values of "not specified." These values are ultimately communicated to the link manager of the local device, which then negotiates with the link manager of the remote device to determine an appropriate polling interval that supports the desired QoS.

The QoS parameters must be passed to the L2CAP layer from the upper layers. Following any configuration negotiation for these values, the L2CAP layer uses the resulting parameter values to control the admission of new connection requests initiated by upper layers of the stack. Based on their QoS requirements, the L2CAP layer may decide to reject any new connection requests if it concludes that establishing a new L2CAP channel with a remote device could compromise the QoS of any existing channel.

With the exception of MTU_{CO}, the definition and use of the QoS parameters was a source of much debate within the SIG's L2CAP task force. The QoS discussions continued to the very day that the specification was finalized for publication in the summer of 1999. Debate centered around the meaning of the QoS parameters, their applicability, implementation effort required, the way they should be negotiated, and so on. No version 1.0 profile supports anything but best effort traffic; hence, the QoS parameters easily could have been omitted from the version 1.0 specification. While proposals to do just that were circulated, it was finally decided that including QoS parameters in the specification was preferable. Having a standardized set of QoS parameters enables experimentation with them; this could help software developers gain QoS experience such that whenever the need arises in any future profiles for the use of QoS guarantees, a solid foundation of knowledge and implementation experience will exist to efficiently address the problem. Inclusion of the QoS parameters is an example of the SIG addressing future flexibility of the specification rather than an immediate requirement of a version 1.0 usage model.

Other Configuration Commands

The most commonly used signaling transactions pertain to the *L2CAP_Connection_Request* and *L2CAP_Configuration_Request* signaling commands, together with their corresponding response commands, and a channel termination request and response transaction using the *L2CAP_Disconnection_Request/_Response* signaling commands. L2CAP signaling also provides two additional exchanges used for testing and information gathering.

The *L2CAP_Echo_Request* and *L2CAP_Echo_Response* command transaction is used to test the connection to a remote device, in a manner similar to the *ping* command in IP networks. The echo commands contain an optional payload portion, which could be used to pass infor-

mation between the devices in a nonstandard, proprietary (vendor-specific) manner.

The *L2CAP_Information_Request* and *L2CAP_Information_Re*-sponse command transaction is used to request implementation-specific information. Currently, the only piece of information that can be requested in a standard manner is the value of the connectionless MTU, MTU_{CL}, that the remote device supports. Nonstandard, vendor-specific information could also be requested with this transaction.

The Host Controller Interface (HCI)

The Bluetooth transport protocols can be implemented in an integrated fashion, entirely on the same host (motherboard or processor) that runs the applications that use the transport protocols. On the other hand, they may be implemented independently of the host on a separate Bluetooth module that is then attached to the host as an add-on accessory attachment or a plug-in card, through some physical interface on the host (such as a USB port or an RS-232 serial port). When implemented separately, the module also contains a *host controller* unit whose responsibility is to interpret the information received from the host and direct it to the appropriate components of the module, like the link manger or the link controller. Likewise, the host controller collects data and hardware/firmware status from the module and passes it to the host as needed. In the reference implementation of a Bluetooth module considered by the SIG, the module contains the radio, the link controller, the link manager and host interfaces for attaching the module to a host, as depicted in Figure 7.1. This reference implementation does not preclude the possibility of other implementations built with different components of the stack.

To permit the interoperable use of nonintegrated modules from different manufacturers, the SIG has defined a standard interface to communicate with the module's host controller in a manner that is independent of the physical interface and transport mechanism used between the host and the host controller. The SIG has also defined a transaction-style communication protocol to carry information between the host and the host controller. This standardized interface between the host controller and the host, together with the corresponding communication protocol between them, is collectively referred to as the *host controller interface* (HCI).

The HCI portion of the specification is by far the largest one–nearly 300 pages including the HCI transport sections. The SIG's HCI e-mail list is also the most populated and busy one. This should come as no surprise; in a sense, the capabilities of the HCI define what can be accomplished with the Bluetooth technology. Since HCI defines the set of functions of a Bluetooth module that are accessible to the host and its applications, HCI is the gatekeeper of the services that this module can provide to its users. Any feature of the module that is not exposed (or that is not exposed properly) by the HCI limits the functionality of the module and ultimately the potential of Bluetooth wireless communications.

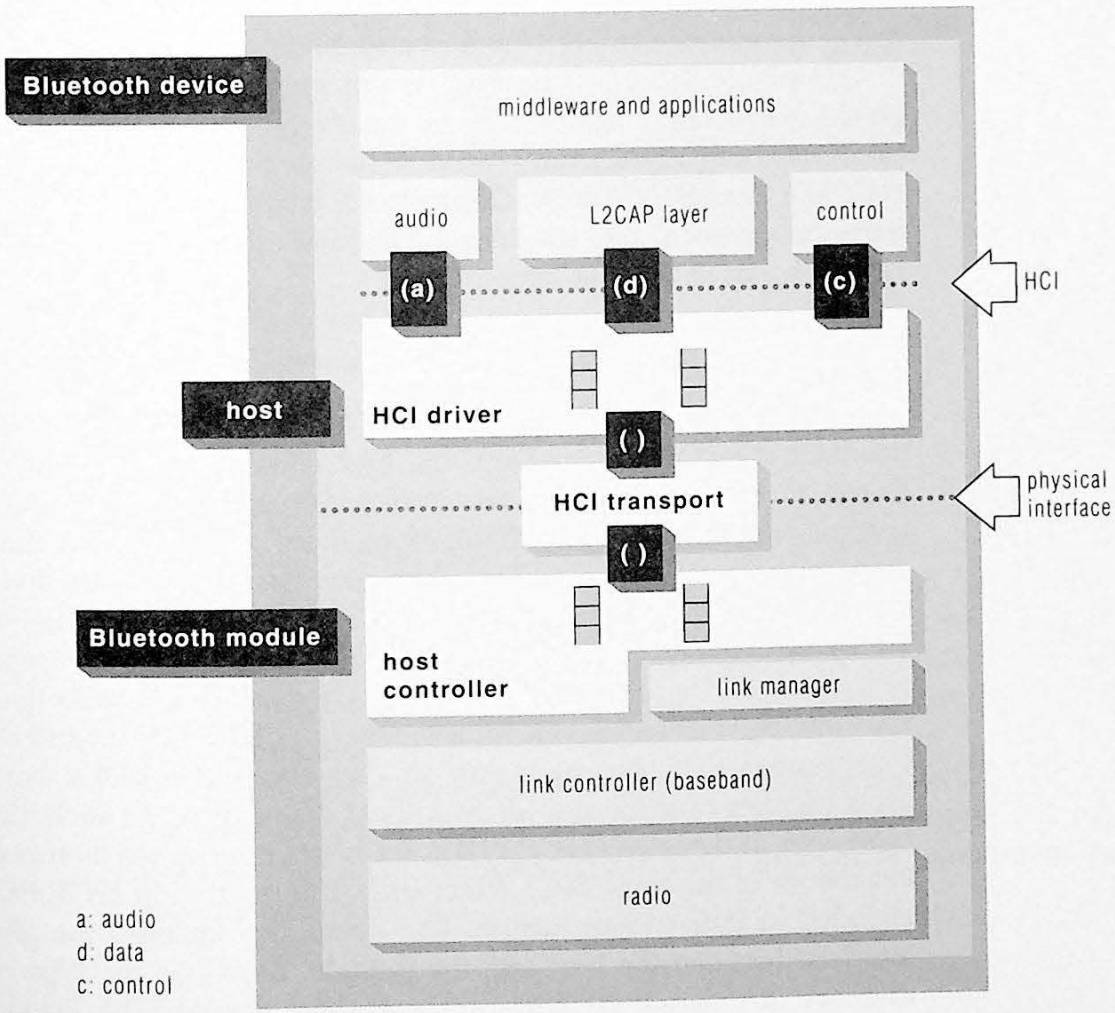

Figure 7.4
System architecture for a device with a host interface.

Figure 7.4 shows a protocol stack with an HCI. The HCI part contains a set of interfaces to the higher layers, which the specification

defines through a long list of HCI_PDUs.[6] There is also a host driver, or HCI driver, which executes the communication protocol for exchanging the HCI_PDUs with the host controller. Finally, there is the transport protocol that carries the HCI_PDUs across the physical interface, or HCI transport. For the HCI transport, the SIG has not developed any new protocols; instead it has reused three existing ones: the Universal Serial Bus (USB) protocol, the RS-232 serial port protocol, and the universal asynchronous receiver and transmitter (UART) protocol, which is a proper subset of the RS-232 protocol and can be used if the module attaches directly to the UART of a serial port without the need for serial cables. The HCI transport has its own complementary driver components in both the host and module.

Implementation of the HCI is not mandatory and in fully integrated systems may not even be necessary. However, manufacturers of products, both OEMs and component integrators, must provide an HCI-like interface to test their product for compliance with the lower transport protocol test specification as described in the Test Control Interface part of the specification.

The HCI_PDU Packet Classes

Three classes of HCI_PDUs are used to exchange information between the HCI layer[7] in the host and the host controller in the module, as shown in Figure 7.5. There are *command*-class HCI_PDUs that carry control and management information; these are sent from the HCI layer to the host controller. There are *event*-class HCI_PDUs that carry control and management information from the host controller to the HCI layer. Finally, there are *data*-class HCI_PDUs that carry fragments of L2CAP_PDUs and SCO data. The HCI specification splits the latter class into two categories, one for asynchronous L2CAP data and one for synchronous data, but in reality all data-class HCI_PDUs have the same number of fields and identical field delineation. As such, the two categories of data-class HCI_PDUs are better interpreted as two different modes of the same HCI_PDU class. Information in HCI_PDUs is encoded in little-endian format. The figure also includes the physical interface between the host and the module. Based on the type of this interface, a separate HCI transport protocol is used. The figure indi-

6. Once again, this is not the terminology used in the specification, but it is used here for consistency with the rest of the book.
7. We collectively refer to the HCI and the HCI protocol driver as the HCI layer.

cates the three HCI transports defined in the specification: RS-232, UART and USB.

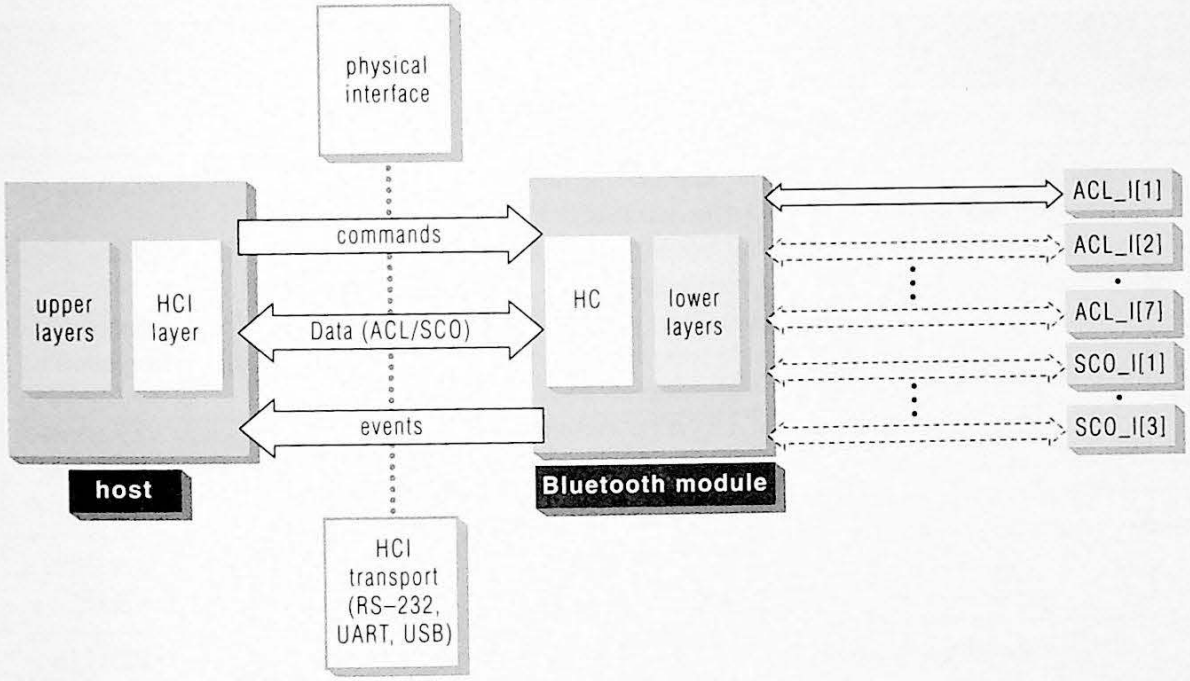

Figure 7.5
The HCI_PDU classes; HC stands for host controller.

For completeness, Figure 7.5 shows the possible active links that a device might have. Two devices may have at most one ACL link between them. A master may have up to seven active ACL links with its slaves (one per active slave). Also, there may be up to three SCO links between the master and all its slaves in a piconet. Note that an ACL link between two devices must exist prior to establishing any SCO links between those devices.

Table 7.8 shows the structure of a command HCI_PDU. It includes an *OpCode* field used to identify the command type. The payload section of the command consists of a number of parameters that vary by command type.

Table 7.8
The command HCI_PDU.

field	size	comments
HCI_Command_Header		
OpCode	2 bytes	OpCode group subfield (OGF) (6 bits) identifies the group that the OpCode belongs to: • 'b111110' identifies a reserved OGF used for Bluetooth logo testing • 'b111111' identifies a reserved OGF for vendor-specific commands used during module manufacture, such as module debugging operations • Other values indicate other groups such as link control, link policy, baseband and others as described below and detailed in the specification
		OpCode command subfield (OCF) (10 bits) identifies a specific HCI command within the particular OGF
Payload_Length	1 byte	length of the payload of the command HCI_PDU in bytes
HCI_Command_Payload		
Payload	*Payload_Length* bytes	the payload of a command HCI_PDU is structured as a sequence of variable-size fields for the various parameters related to this command

Table 7.9 shows the structure of an event HCI_PDU. Similarly to a command HCI_PDU, it consists of an *Event_Code* field used to identify the event and a payload section with a number of parameters which are relative to this event HCI_PDU.