acknowledged by the Slave. If the Slave does not have any information to send, it responds to the Master with a NULL packet.

The FHS (Frequency Hop Synchronization) is a special control packet containing real time information. It contains the BD_ADDR and clock information of the sender. The clock information is updated before each retransmission (since it is real time and the clock would have changed since the last transmission) for this particular packet type. This packet is used for frequency hop synchronization before the piconet channel has been established or when existing piconet changes to a new piconet (by means of a role switch).

The DM1 (Data Medium Rate 1-slot) packet is used by the Link Manger and Link Controller to exchange control packets. Besides control packets, this packet can also be used to carry regular data. The details for this packet type are provided in the ACL Packet Types section.

The scenarios in which these packet types are used are illustrated later in Figure 3.10.

3.3.7.2 ACL Packet Types

The different types of ACL packets are explained in Table 3.3. Different packets types are suitable for different scenarios. For example when high throughput is desired in only one direction, 3-DH5 packets are the most suitable. This is the case when transferring files, downloading data from the internet etc. In such cases the majority of the data is transmitted in one direction and there are only a few bytes that need to be transmitted in the reverse direction.

The information about supported ACL packet types is provided at the time of ACL connection creation. The link managers running on the two devices negotiate the packet types that will be used subsequently based on this information. Only the packet types that are successfully negotiated are used for subsequent transmissions.

Table 3.3 ACL Packet Types

Type	*User Payload (bytes)*	*Symmetric Max Rate (kb/s)*	*Asymmetric Max Rate (kb/s)*		*Remarks*
			Forward	*Reverse*	
DM1	0-17	108.8	108.8	108.8	Data – Medium Rate, 1 Slot
DH1	0-27	172.8	172.8	172.8	Data – High Rate, 1 Slot
DM3	0-121	258.1	387.2	54.4	
DH3	0-183	390.4	585.6	86.4	
DM5	0-224	286.7	477.8	36.3	
DH5	0-339	433.9	723.2	57.6	This packet is used for maximum throughput in one direction for Basic Rate (BR) links.
AUX1	0-29	185.6	185.6	185.6	
2-DH1	0-54	345.6	345.6	345.6	
2-DH3	0-367	782.9	1174.4	172.8	
2-DH5	0-679	869.1	1448.5	115.2	
3-DH1	0-83	531.2	531.2	531.2	
3-DH3	0-552	1177.6	1766.4	235.6	
3-DH5	0-1021	1306.9	2178.1	177.1	This packet is used for maximum throughput in one direction for EDR links.

As shown in Table 3.3, the maximum throughput is achieved when using asymmetric data transfer using DH5 packets for BR and 3-DH5 packets for EDR.

3.3.7.3 Synchronous Packet Types

HV (High Quality Voice) and DV (Data-Voice) packets are used for SCO transmissions. The HV packets do not include a CRC and thus these are not retransmitted. The DV is a combined data-voice packet and it has separate sections for voice and data. These packets do not include a CRC on the voice section, but include a CRC on the data section. So in case the data section is not received properly at the remote side, it is retransmitted. The voice section however is not retransmitted and contains the next speech information.

eSCO uses EV packets. These packets include a CRC and retransmission is done if an acknowledgment is not received from the remote side within the retransmission window. EV3, EV4 and EV5 packets are used for Basic Rate (BR) operations. 2-EV3, 2-EV5, 3-EV3, 3-EV5 packets are used for Enhanced Data Rate (EDR) transmissions. As mentioned previously eSCO packets can be used for both 64kb/s speech as well as data at the rates mentioned below.

3.3.8 Link Controller States

There are three major states used in the link controller:

- *STANDBY:* This is the default state of the device. In this state, the device may also enter low power mode to save power. The link controller may leave this state to scan for inquiry or page messages from other devices or to itself inquire or page.
- *CONNECTION:* Once a device knows the address of the device to connect to (using the inquiry procedure), it can create a connection to it. This procedure is known as paging. The device that initiates paging becomes the Master

Table 3.4 Synchronous Packet Types

Type	*User Payload (bytes)*	*Symmetric Max Rate (kb/s)*	*Remarks*
HV1	10	64	
HV2	20	64	
HV3	30	64	
DV	10 + (0–9) D	64.0+57.6 D	Data Voice. This packet type can support both Data and Voice simultaneously. D indicates the Data Payload.
EV3	1-30	96	
EV4	1-120	192	
EV5	1-180	288	
2-EV3	1-60	192	
2-EV5	1-360	576	
3-EV3	1-90	288	
3-EV5	1-540	864	

after the connection is established. In this state, packets can be exchanged between the Master and the Slave(s). This state is left through a detach or reset command.

- *PARK:* A Slave can enter the park state if it does not need to participate in the piconet but still needs to remain synchronized with it. This mode helps conserve power. The Slave wakes up periodically to resynchronize and listen to the channel to decide whether it should come back to active state or not.

Besides this, there are seven substates. These are interim states that are used for discovering devices in the vicinity and establishing a connection. The seven substates are divided into two categories:

Device Discovery Substates

1. *Inquiry scan:* In this substate, the device listens for incoming inquiry requests.
2. *Inquiry:* This substate is used to discover devices in the vicinity.
3. *Inquiry response:* After receiving the inquiry message, the Slave enters this state and transmits an inquiry response message.

Connection Establishment Substates

4. *Page scan:* In this substate, the device listens for incoming connection requests.
5. *Page:* This substate is used by a device to connect to another device. The device that initiates the paging ends up being a Master after the connection is established and the device that was in page scan mode ends up being Slave.
6. *Slave response:* After receiving the page message, the Slave enters this state and transmits a Slave response message.
7. *Master response:* The Master enters this state after it receives the Slave response message.

The different states along with the permitted transitions from one state to another are illustrated in Figure 3.8. As an example, the transition of states and substates during inquiry, connection and disconnection is illustrated in Figure 3.9 and Figure 3.10.

3.3.8.1 Connection State

During the connection state, the packets can be exchanged between the Master and the Slave. The first few packets contain control messages that are exchanged between the link managers of the devices to setup the link and negotiate link parameters. If the connection is no longer needed, the connection state may be left by either a detach or reset command.

During the connection state, the device may be in one of the following three modes:

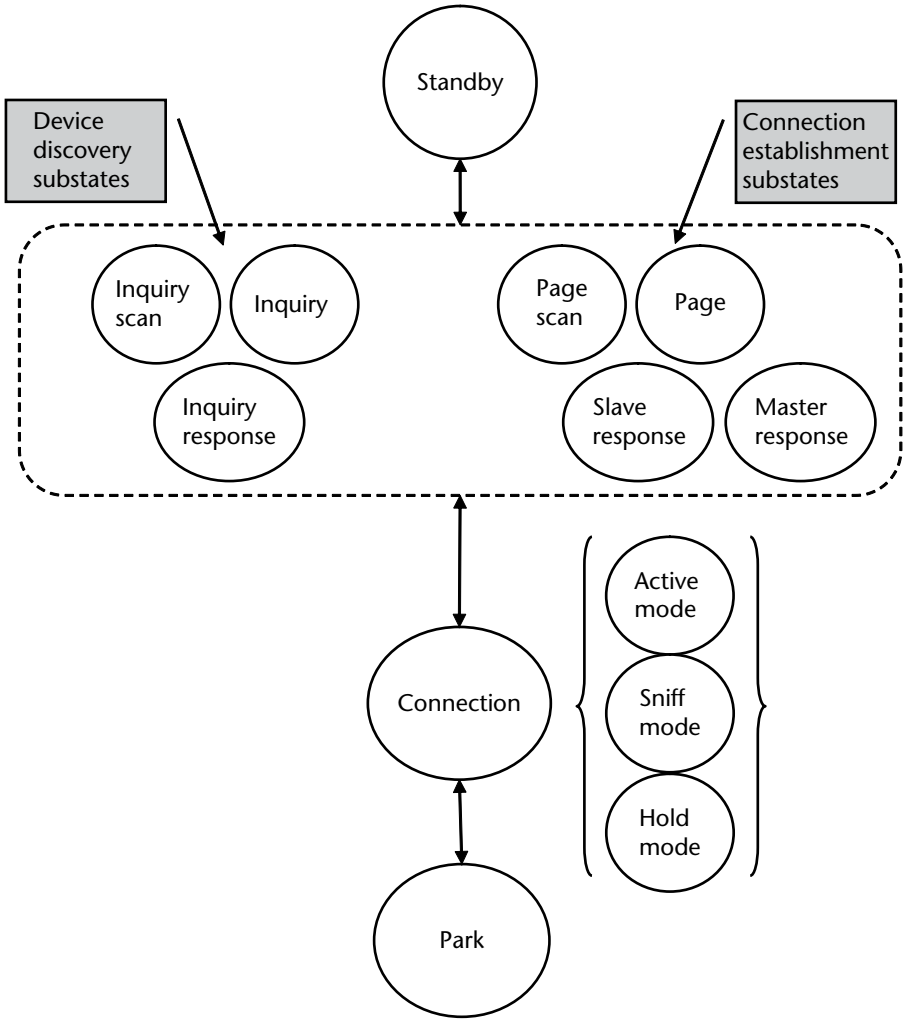

Figure 3.8 Link controller states.

- Active Mode.
- Hold Mode.
- Sniff Mode.

Active Mode

In the active mode, one Master and up to seven Slaves can be active at any time in the piconet. The Master schedules the transmissions in different slots based on the SCO/eSCO connections that are established and the traffic requirements from different Slaves. Even if the Master does not have any data to send, it keeps on regularly transmitting POLL packets to the Slaves to keep them synchronized to the channel. If the Slave has data to send, it sends the appropriate data packet. Else it responds with a NULL packet to acknowledge the POLL packet.

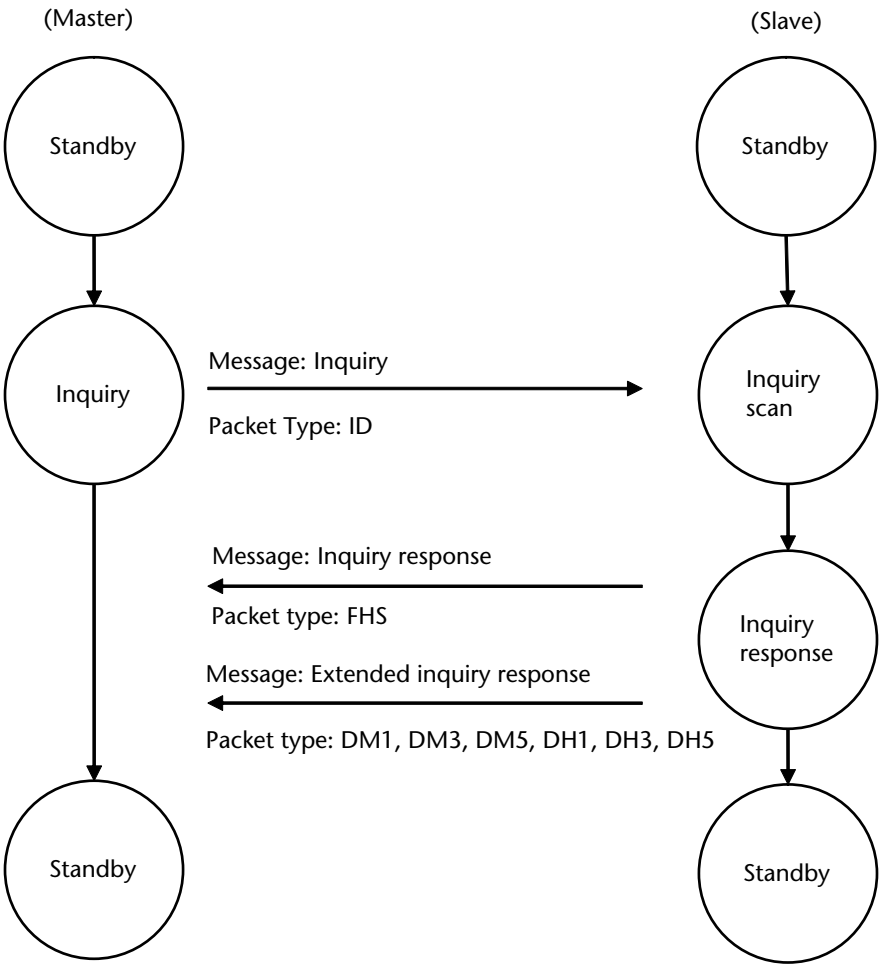

Figure 3.9 Link controller messages and states during inquiry.

Each Slave is assigned a 3-bit address called Active Member Address (AM_ADDR). The Master uses this address in the packets to identify the Slave for which those packets are meant.

Hold Mode

During the connection state, the ACL logical transport to the Slave can be set to hold mode. In this mode, the ACL packets are not exchanged. The voice links (SCO, eSCO) are still supported.

Before entering the hold mode, the Master and the Slave agree on the time duration for which the Slave will remain in hold mode. During this time duration, the Slave can either go to low power mode or do other activities like scanning, paging, inquiring or participating in another piconet.

On expiration of the time duration of the hold mode, the Slave wakes up and listens for transmissions from the Master.

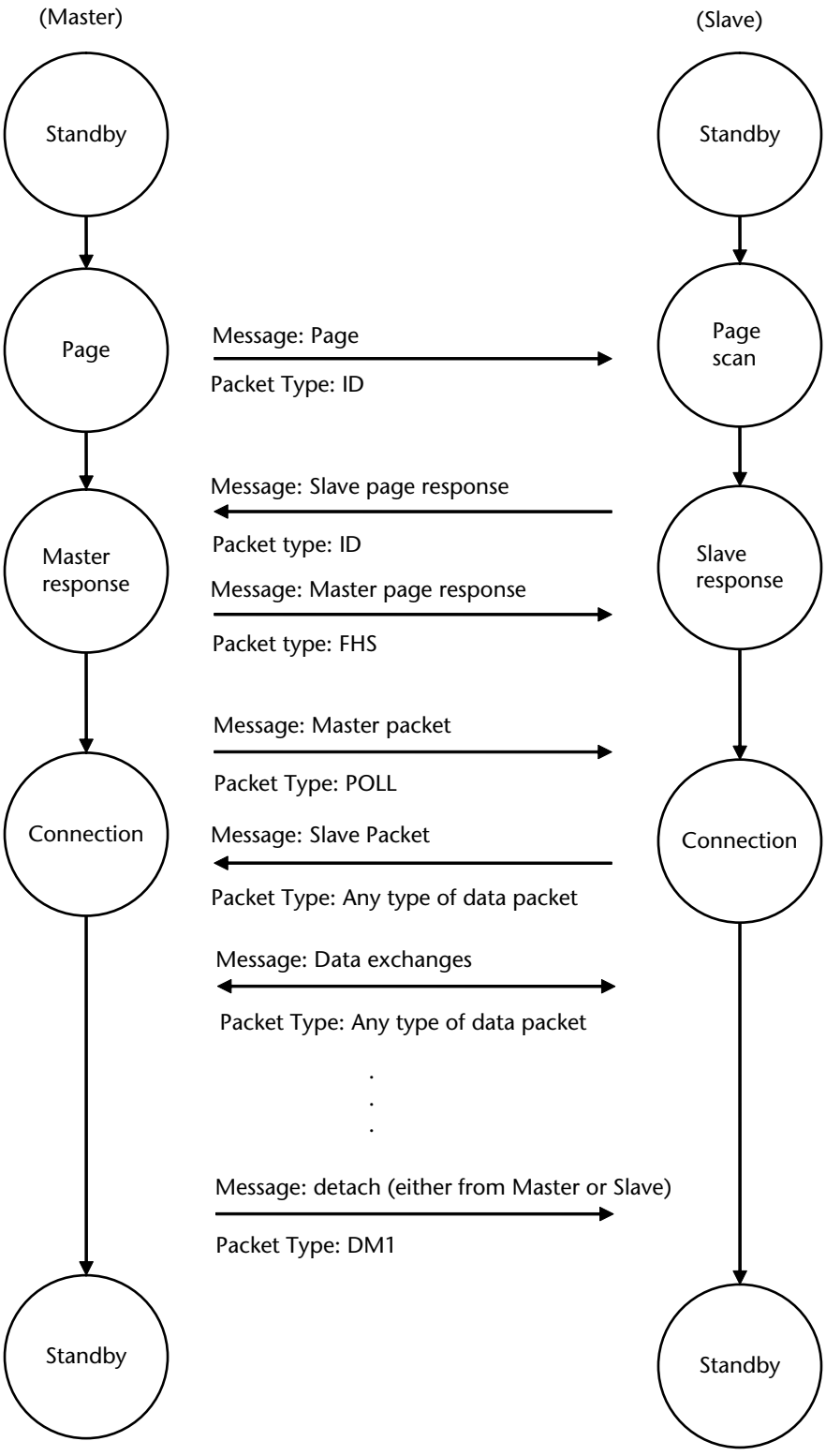

Figure 3.10 Link controller messages and states during connection and disconnection.

Sniff Mode

The sniff mode is the most commonly used low power mode. It affects only the ACL logical transport and does not affect the SCO or eSCO logical transports. This means that when this mode is activated, the transmissions on ACL logical transport are reduced, while the transmissions on SCO or eSCO logical transport continue as usual.

In the sniff mode, the device can become absent from the piconet for a certain period of time. So a duty cycle is defined consisting of the duration for which the device will be present and active in the piconet versus the time it will be absent.

A practical use of the sniff mode is when a connection is made between a mobile phone and a mono headset. The two devices can continue to be connected for several days even when there is no active voice call going on. This ensures that as soon as there is an incoming call, time is not wasted in first creating an ACL connection and then creating a SCO connection to route the call. So, to conserve power, the ACL connection between the mobile phone and the headset is put in sniff mode. Most Bluetooth headsets implement the sniff mode.

Another use of the sniff mode is when a Bluetooth keyboard is attached to a PC or laptop. Since the user may not be actively typing all the time, the ACL link is put into sniff mode after an inactivity timer. As soon as the user types a key, the link is brought back into the active mode.

Besides saving power, the sniff mode may also be useful when the device needs to enter into a scatternet scenario. In this scenario, the device may put the ACL link of the current piconet into sniff mode and then become active on another ACL link in the second piconet. So by tuning the sniff parameters it may have periods in which it is absent in first piconet and present in the second piconet and vice versa.

This mode requires a good bit of attention at the link level, especially when there is a connection in sniff mode for a long duration. Any clock drifts on either side while in sniff mode for a long time, could lead to the loss of packets or even loss of the link. Generally controllers wake up time to time in between to synchronize with the Master and go back to sniff mode again to avoid such problems.

3.3.8.2 Park State

A Slave can enter park state if it does not need to participate in the piconet channel but still needs to be synchronized with it. It's a very low power mode with very little activity. In this mode both ACL and SCO or eSCO traffic is stopped. The Slave still remains synchronized to the channel.

The parked Slave wakes up at regular intervals to listen to the channel to check whether there are any messages for it. The Master informs the parked Slaves about any messages for them through broadcast messages.

When a device is put into park state, a different address known as Parked Member Address (PM_ADDR) is assigned to it by the Master during the parking procedure. The Active Member Address (AM_ADDR) that was earlier used by the Slave in active mode is freed up so that it can be re-used by the Master to make a connection with some other device.

Park state is also used when more than seven Slaves need to be connected to a single Master. At a time only seven Slaves can be active. The remaining Slaves are

put into park state. If a parked Slave needs to be unparked then it can be swapped with another active Slave that will be put to park state.

3.4 Link Manager (LM)

The Link Manager performs the functions of link setup and control. The Link Manager Protocol (LMP) is used for communication between the Link Managers of the two devices. The communication messages between the devices are carried over the ACL-C logical link. (As mentioned in the previous section, ACL-C logical link carries control data while the ACL-U logical link carries the user data).

The LMP messages are transmitted using DM1 packets. If HV1 SCO link has been established between the two devices and the length of the payload is less than 9 bytes, then DV packets may also be used. In practice, DV packets are seldom used.

The packets transmitted by LM are referred to as PDUs. Each PDU contains the following fields:

- Transaction ID (TID);
- OpCode;
- Payload.

A transaction is a set of messages that are transmitted to achieve a particular purpose. It may have more than one PDU and all PDUs contain the same TID. The OpCode is used to uniquely identify different type of PDUs. The LMP messages are denoted as LMP_*message_name*. For example, the message to establish a connection is denoted as LMP_host_connection_req. A response PDU to this could be LMP_accepted or LMP_not_accepted.

The functionality supported by LMP includes the following:

- *Connection Control:* This includes procedures for creation and removal of a connection as well as controlling all aspects of a connection.
- *Security:* This includes procedures for authentication, pairing, and encryption.
- *Informational Requests:* This includes various PDUs that the LMs exchange with each other to get the characteristics of the remote device. For example, the LM may find out the version of the remote device using the LMP_version_req PDU. Similarly it may find out the list of features supported by the remote device by sending the LMP_features_req PDU. The remote side would respond using an LMP_features_res PDU.
- *Role switch:* If the Slave device wants to reverse the roles and become the Master, then these procedures are used. A practical example of this would be if a Bluetooth mouse establishes a connection with a laptop. Since the connection establishment is initiated by the mouse, it becomes the Master of the connection. After the connection is established the laptop may prefer to become the Master of the connection so that it can control other peripherals as well. So it can invoke the role switch procedures to become the Master.

- *Modes of operation:* In the baseband section, hold mode, sniff mode, and park state were explained. These different modes of operations are invoked by the link manager. Besides this the link managers of the two devices negotiate the parameters for these modes.
- *Logical transports*: The LM invokes the procedures to create and remove the SCO and eSCO logical transports. The parameters of these transports are also negotiated by the LMs of the two devices. These can also be renegotiated at any time during the connection. For example an eSCO link is established by sending an LMP_eSCO_link_req and removed by sending the LMP_remove_eSCO_link_req.
- *Test Mode:* This includes various PDUs for certification and compliance testing of the Bluetooth radio and baseband.

The connection control and security functionality of link manager is explained in further detail below.

3.4.1 Connection Control

The connection functionality includes all procedures related to creation and removal of a connection and controlling all aspects of the operation of that connection.

Some of the major functions include the following:

- *Connection establishment:* This includes the procedures for establishing a connection with a remote device. If security on the connection is required, then security procedures are also invoked. At the end of this procedure, the device which initiated the connection establishment becomes the Master and the other device becomes the Slave. It's also possible to switch the roles by invoking the role switch procedure during connection establishment. In that scenario, the device which initiated the connection becomes the Slave and the other device becomes the Master.
- *Detaching a connection:* This includes the procedures for detaching a connection. It may be started by either the Master or the Slave.
- *Link supervision:* This feature is used to detect loss of a physical link, for example, when the devices move away from range or one of the devices loses battery power. The LMs of both devices use a supervision timer to detect if the link has been lost. If the link is lost, then the LM reports the link loss to the host. One of the sections earlier explained the POLL and NULL packets. These packets are like a heartbeat between the pair of devices in a piconet. These ensure that the link is still active and that the other device has not gone out of range or has been reset. Hence there is a timer (timeout is negotiated right at the link establishment time) which is fired at both sides for detecting an inactivity on these heartbeat signals. Once there is an inactivity detected for the stipulated timeslots, the link managers on both sides assume that there is something wrong on the link, or the devices have moved out of range. Hence they indicate a link supervision timeout to the upper layers.

- *Enable, disable AFH and update of channel map:* Adaptive Frequency Hopping (AFH) was explained in previous sections. This feature helps in improving the performance in case of interference by removing the channels which have interference from the frequency hopping pattern. The LM of the Master enables or disables this feature and provides the updated channel map to each Slave. The LM of the Slave receives these PDUs and updates it's channel map accordingly. The LM of the Master may also request the Slave for information about the quality of channels. This is an optional feature and if the Slave supports it then it provides a channel map to the Master indicating whether the channels are good, bad, or unknown.
- *Control of the transmit power level:* If the receiver observes that the characteristics of the received signal differ too much from preferred values, it may ask the transmitter to increase or decrease the power level of the transmitting device. This feature is optional and it allows the devices to use the most optimal power levels for communication. As an example, as soon as the devices move away from each other, the receiver may observe degradation of the received signal. So it may request the transmitter to increase the transmit power level. If the devices move closer, the receiver may again request the transmitter to reduce the transmit power level. This will allow more efficient use of battery power.
- *Quality of Service:* This feature is used for bandwidth allocation on ACL logical transport so that the requested amount of bandwidth can be reserved on the ACL logical transport.

An example of air logs of different LMP transactions is shown in Figure 3.11. Some of the points worth observing are:

1. There are two different columns: *Role* and *Initiated By*
   a. Role indicates which device sent the current packet.
   b. Initiated By indicates which device originally initiated this transaction. For example if the transaction was initiated by the Master and the Slave is now responding, then Role will indicate Slave and Initiated By will indicate Master.
2. Frame #1: Connection request initiated by the Master.
3. Frame #9: Connection completed.
4. Frame #2 to #8: Master and Slave exchange information about supported features and version.
5. Frame #14: SET_AFH command sent by Master to Slave to enable Adaptive Frequency Hopping.
6. Frame #292: detach command sent by Master to disconnect the connection.

3.4.2 Security

The security procedures include the following:

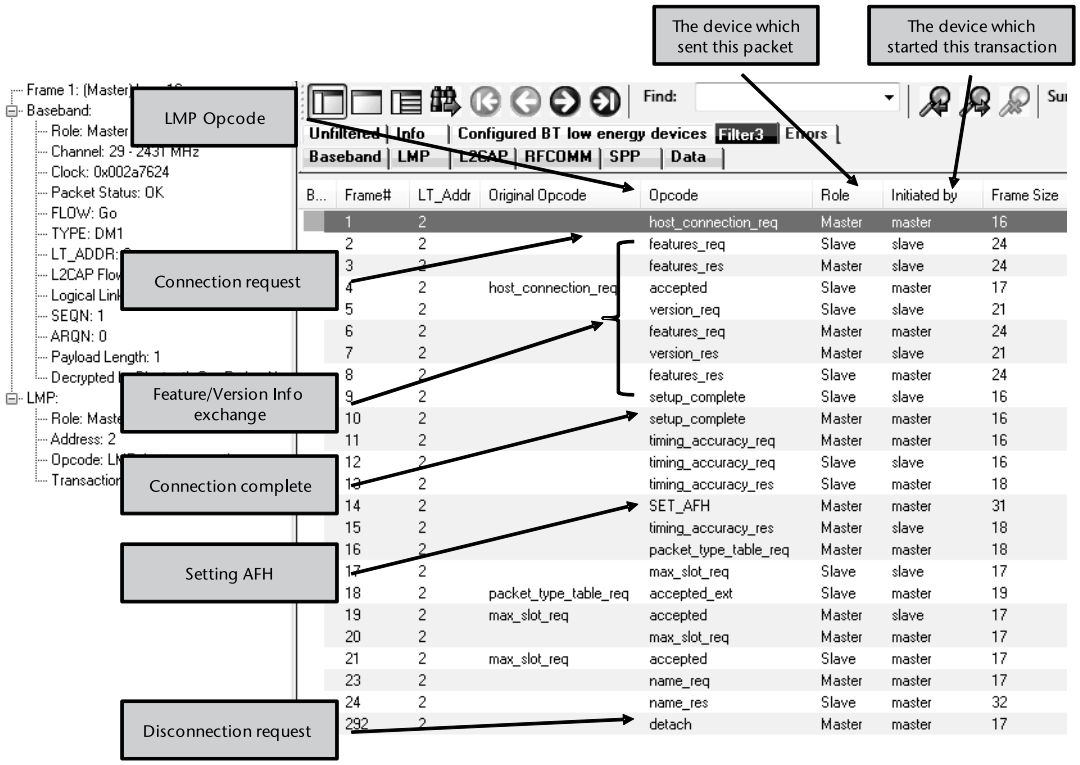

Figure 3.11 Example of LMP transactions.

- Pairing;
- Authentication;
- Encryption;
- Secure Simple Pairing.

3.4.2.1 Pairing

Pairing is the process of associating two devices with each other. When the two devices agree to communicate with each other, they exchange a passkey. This passkey can be considered to be similar to a password that is shared between the two devices. The passkey is also referred to as the Bluetooth PIN and is generally entered on the UI. For example when connecting a mobile phone to a laptop, the user will have to enter identical PIN codes on both the laptop and the mobile phone.

Pairing can be done by using older legacy pairing procedures or by *secure simple pairing* procedures. (Secure simple pairing was introduced in version 2.1 + EDR of the Bluetooth specification).

The Bluetooth PIN that is entered on the UI is used along with a random number and BD_ADDR to create a link key. This link key is used for authentication between the two devices for all subsequent connections. This key is stored in the devices so that when the next time the devices are connected, the user does not need to enter the PIN again.

3.4.2.2 Authentication

Authentication is the process of verifying *who* is at the other end of the link. The authentication process starts when the two devices initiate a connection establishment. It is based on a challenge-response scheme. The verifier sends a challenge to the other device that contains a random number (the challenge). The other device calculates a response that is the function of the challenge, its own BD_ADDR and a secret key. The response is sent back to the verifier that checks whether it is correct or not.

The success of the authentication procedure requires that the two devices share a secret key. This key was generated during the pairing process. If they do not share a secret key, then the pairing procedure is initiated.

3.4.2.3 Encryption

This is an optional procedure and the Master and Slave must agree whether to use encryption or not. To use encryption, it's mandatory to perform authentication.

If encryption is enabled, then all data exchanged on the link is encrypted using an encryption key. The encryption key can be from 8-bits to 128-bits.

3.4.2.4 Secure Simple Pairing

Secure simple pairing was introduced in version 2.1 + EDR of the Bluetooth specification to both simplify the pairing mechanism and improve security. This is explained in further detail later in this chapter.

3.5 Host Controller Interface (HCI)

One of the strengths of Bluetooth specification is that it provides a uniform method for accessing the controller's capabilities. This has several advantages. First the development of the software for controller and host can go on independently. Secondly a host from one vendor can work easily with a controller from another vendor. For example a Bluetooth dongle from any vendor can be plugged into a PC to use Bluetooth functionality.

The Host Controller Interface (HCI) provides this uniform method for accessing the controller's capabilities. The host interfaces with the controller using this layer. It is optional and is required only in implementations where the software for controller and host run on different processors. It may be skipped in implementations where the software for controller and host run on the same processor.

The communication over the HCI interface happens in form of packets. The host sends HCI command packets to the controller and is asynchronously notified by the controller using HCI events.

The commands and events for LE controllers are also exchanged over the HCI interface. The LE controllers use a reduced set of HCI commands. Some commands and events are re-used for multiple controller types.

There are four possible types of controllers:

1. *BR/EDR Controller*: Supports only BR/EDR functionality.
2. *BR/EDR/LE Controller:* Supports both BR/EDR and LE functionality.
3. *LE Controller:* Supports only LE functionality.
4. *AMP Controller:* Supports functionality of AMP Protocol Adaptation Layer (PAL) (BT 3.0 + HS).

3.5.1 HCI Packet Types

There are four types of packets that can be sent on the HCI Interface.

1. HCI Command Packet.
2. HCI Asynchronous (ACL) Data Packet.
3. HCI Synchronous (SCO/eSCO) Data Packet.
4. HCI Event Packet.

These are shown in Figure 3.12.

3.5.1.1 HCI Command Packet

The HCI Command Packet is used by the host to send commands to the controller. Each command is assigned a 2 byte unique OpCode. The OpCode is further divided into two fields:

1. *OpCode Group Field (OGF)* (upper 6 bits): This field is used to group similar OpCodes together.
2. *OpCode Command Field (OCF)* (lower 10 bits): This field is used to identify a particular command in the OpCode group.

The HCI command packets are denoted as HCI_*xxx* where *xxx* is the command that is given by the host to the controller. For example the command to reset the controller is denoted as HCI_Reset.

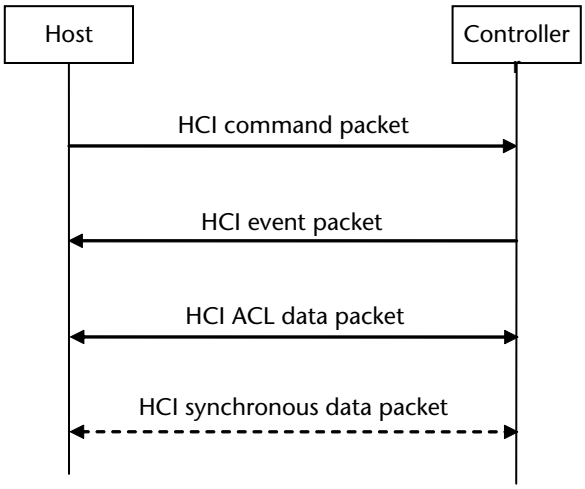

Figure 3.12 HCI packet types.

The format of HCI command packet is shown in Figure 3.13. It consists of a 16-bit OpCode followed by an 8-bit *parameter total length* field. The parameter total length field specifies the total length of all parameters that are contained in the remaining packet measured in octets. (An octet denotes an 8-bit value). This is followed by the command parameters.

3.5.1.2 HCI Event Packet

The HCI Event Packet is used by the controller to notify the host when an event occurs. This may be in response to an HCI command that was sent earlier (For example a command to create a connection), or due to any other event (For example loss of connection or incoming connection request). Errors are also indicated using HCI Event Packets.

The format of HCI event packet is shown in Figure 3.14. It consists of an 8-bit Event Code followed by an 8-bit parameter total length field. The parameter total length field specifies the total length of all parameters that are contained in the remaining packet (measured in octets). This is followed by the event parameters.

3.5.1.3 HCI ACL Data Packet

The HCI ACL Data Packet is used to exchange data between the host and the controller. ACL Data packets can only be exchanged after a connection has been established.

The format of HCI ACL data packet is shows in Figure 3.15. It consists of a 12-bit Handle followed by 2-bit Packet Boundary (PB) and 2-bit Broadcast (BC) flags. This is followed by 16-bit Data Total Length field which specifies the length of the following data in octets. This is followed by the data.

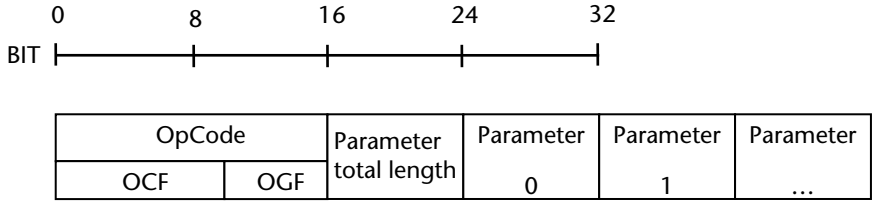

Figure 3.13 HCI command packet format.

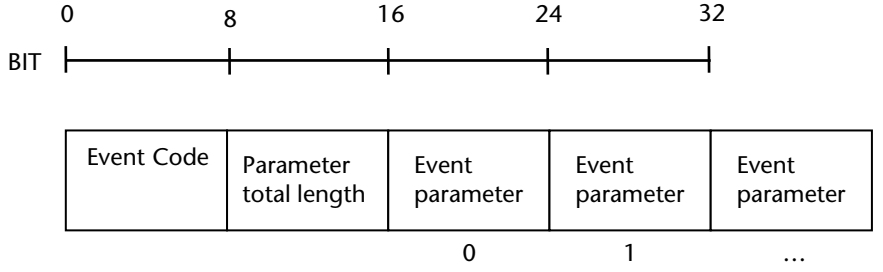

Figure 3.14 HCI event packet format.

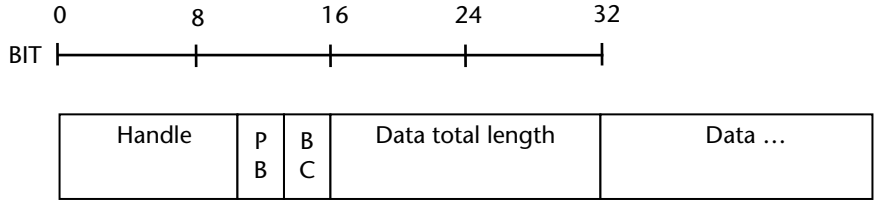

Figure 3.15 HCI ACL data packet format.

The Handle specifies the Connection Handle on which data is to be sent. Each connection is specified by a unique connection handle.

The *packet boundary flag* is useful when a higher layer protocol packet is fragmented into multiple ACL data packets. It indicates whether this packet is the first packet in the fragment or a continuing packet. This field also indicates the flushable characteristics of the packet—whether it is automatically flushable or not. Automatically flushable packets are automatically flushed based on a timer if they cannot be transmitted by the time the timer expires.

The *broadcast flag* indicates whether this is a point-to-point packet or a broadcast packet.

3.5.1.4 HCI Synchronous Data Packet

The HCI Synchronous Data Packet is used to exchange synchronous data between the host and the controller.

The format of HCI synchronous data packet is shown in Figure 3.16. It consists of a 12-bit Connection Handle followed by 2-bit Packet Status (PS) flag and 2-bit reserved (RES) field. This is followed by 16-bit Data Total Length field which specifies the length of the following data in octets. This is followed by the data.

The Connection Handle specifies the Connection Handle on which data is to be sent. Each SCO or eSCO connection is specified by a unique connection handle.

The Packet Status flag gives an indication whether the packet was correctly received or not.

3.5.2 HCI Commands and Events

The HCI commands are grouped into several categories. Some examples of the HCI commands in various groups are provided below.

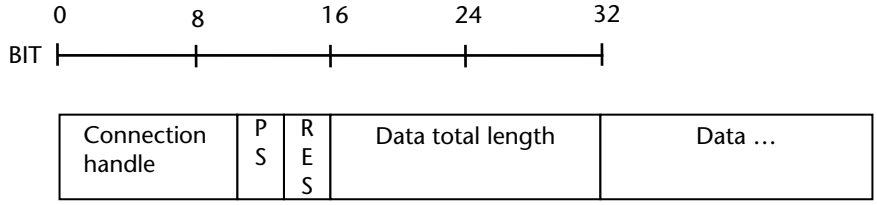

Figure 3.16 HCI synchronous data packet format.

- *Link Control Commands:* These allow a controller to control connection to other controllers.
  - HCI_Inquiry—Discover devices in the vicinity.
  - HCI_Inquiry_Cancel—Cancel Inquiry.
  - HCI_Create_Connection—Create a connection to remote device.
  - HCI_Disconnect—Terminate an existing connection.
  - HCI_Remote_Name_Request—Obtain user friendly name of another controller.
- *Link Policy Commands:* These are used to control how Bluetooth piconets and scatternets are established and maintained.
  - HCI_Hold_Mode—Put the connection in hold mode.
  - HCI_Sniff_Mode—Put the connection in sniff mode.
  - HCI_Switch_Role—Switch the role from Master to Slave and vice versa.
- *Controller and Baseband Commands:* These provide access to various capabilities of Bluetooth hardware.
  - HCI_Set_Event_Mask—Control which events can be generated by the controller to be sent to the host.
  - HCI_Reset—Reset the controller.
  - HCI_Write_Local_Name—Modify the user friendly name of the local device.
  - HCI_Write_Class_of_Device—Write the Class of Device parameter.
- *Informational Parameters:* These provide information about the capabilities of the controller.
  - HCI_Read_Local_Version_Information—Version, Manufacturer name.
  - HCI_Read_BD_ADDR—Reads the BD_ADDR of the controller.
- *Status Parameters:* These provide information about the current state of the controller.
  - HCI_Read_Link_Quality_Information about the link quality.
- *LE Controller Commands:* These are used for LE controllers.
  - HCI_LE_Set_Event_Mask—Controls which LE Events are generated by the controller to be sent to host.
  - HCI_LE_Read_Buffer_Size—Read the maximum size of data packets that can be sent to controller and the total number of buffers.
  - HCI_LE_Create_Connection—Create an LE connection.

Some examples of HCI events are provided below.

- *Inquiry Complete:* Is used to indicate completion of inquiry.
- *Inquiry Result:* Is used to provide information about remote devices found during inquiry.
- *Connection Complete:* Indicates that a connection has been established.
- *Disconnection Complete*: Indicates that a connection has been terminated.

- *Hardware Error:* Indicates some hardware failure.
- *LE Meta Event:* Is used to encapsulate all LE events. A sub-event code indicates the exact event that has occurred. It could be one of the following:
  - LE Connection Complete—Indicates that an LE connection has been established.
  - LE Advertising Report—Indicates that an advertising report has been received.
  - LE Connection Update Complete Event—Indicates that the process to update the connection parameters has been completed.

The following two HCI events are used quite frequently:

- *Command Complete:* This event is used by the controller to transmit the return status of the command as well as the return parameters associated with the command.
- *Command Status:* This event is used by the controller to indicate that it has received the command and has started performing the task for the command though the task may not have completed yet. This event is needed to provide a mechanism of asynchronous operation so that the host can continue doing other operations while the controller performs the tasks requested by the host. The host is notified by another event when the task is completed by the controller.

An example of the HCI commands and events exchanged during inquiry is shown in Figure 3.17.

3.5.3 Buffers

The controller has buffers to receive the commands and ACL/Synchronous data sent by the host.

- *Command Buffers:* The BR/EDR/LE controller uses shared buffers for the commands.
- *Data Buffers:* The controller may either have shared or separate data buffers for BR/EDR and LE.

Whether the controller has separate or shared data buffers for LE can be determined using the HCI_Read_Buffer_Size command. This will be explained in a later section.

3.5.4 HCI Flow Control

In general the controller has much less buffers and processing power as compared to the host. This means that if the host sends a few packets to the controller to process in fast succession, then it is possible that the buffers of the controller get full.

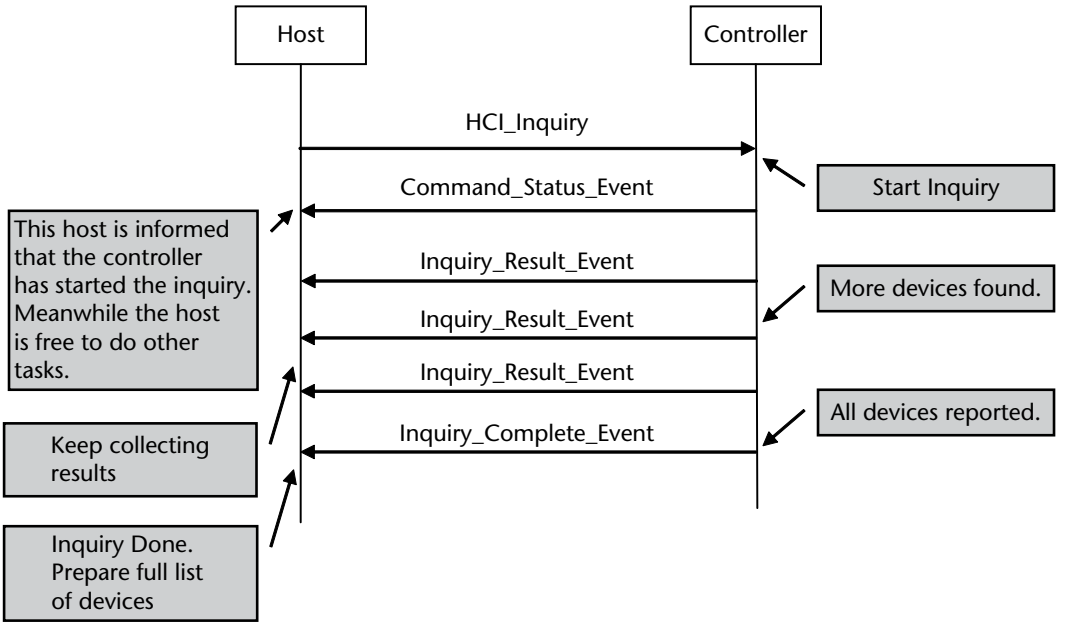

Figure 3.17 HCI commands and events for inquiry.

Flow control mechanism is needed in such a scenario to ensure that the host does not send any packets further till the time the previous ones are processed.

The HCI interface provides separate flow control mechanisms for the following:

1. *Host to Controller Data Flow Control:* ACL Data Packets from host to controller.
2. *Controller to Host Data Flow Control:* ACL Data Packets from controller to host.
3. *Command Flow Control:* HCI Command Packets from host to controller.

Out of these, Controller to Host Data Flow Control is not much widely used since typically the host has more resources than the controller. So most of the time, the controller can send data to the host without any flow control.

3.5.4.1 Host to Controller Data Flow Control

The Host to Controller Data flow control is used to regulate the flow of data from the host to the controller. This is needed because the controller may have much lesser buffers as compared to the host. Besides this the controller also needs time to process (transmit/re-transmit) those packets. During that time, the flow control ensures that the host sends only as many packets as the number of available buffers in the controller.

There are two mechanisms:

1. Packet Based Flow Control.
2. Data-Block-Based Flow Control.

The Packet Based Flow Control is used quite widely and this is explained in detail in this section. It is shown in Figure 3.18.

During initialization the host sends the Read Buffer Size command to get information about the buffers in the controller. The response to the Read Buffer Size contains the following:

1. Number of ACL buffers.
2. Size of each ACL buffer.
3. Number of SCO buffers.
4. Size of each SCO buffer.

If the controller supports LE, then it is possible for the controller to either have separate LE buffers or share the same buffers. The host can send the LE Read Buffer Size command to find this out. If the response to this indicates zero as the length of the buffers, then it means that the same buffers are shared between BR/EDR and LE. Otherwise the response would indicate the number of LE buffers present in the controller and the size of those buffers.

Once the host has information on the number of ACL buffers, it knows that at any given time, it can have a maximum of that many packets outstanding (to be processed) on the controller side. For example if the Number of ACL buffers was four, then the host can have a maximum of four packets outstanding on the controller side at any given time.

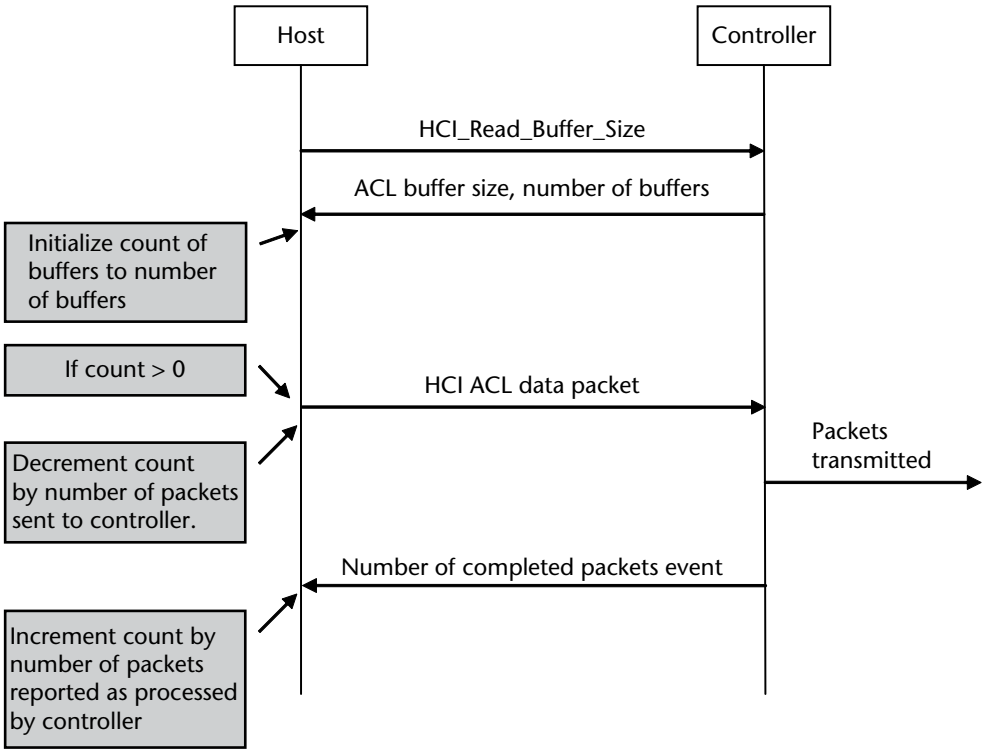

Figure 3.18 Packet-based Host to Controller data flow control.

The host maintains a count of the number of ACL buffers available in the controller. It initializes this value to the number of ACL buffers it received in the response to Read Buffer Size command. After that every time it sends a packet to the controller, it decreases this count by one.

Once the controller has completed processing one or more packets, it sends that count of processed packets in the Number of Completed Packets event. The host knows that some additional buffers have been freed up on the controller side and it increments its count by the number of packets reported in the Number of Completed Packets event.

3.5.4.2 Command Flow Control

The flow control used for commands is very simple. The command complete and command status events that are sent by the controller to the host have a field which indicates how many more commands can be sent to the controller. The host can send up to that many commands to the controller before waiting for the next command status or command complete event.

In general, if the controller is not able to complete a command right away, it sends a command status event. Later on when the command is completed it either sends a command complete event or another relevant event depending on the command that was sent. For example if the host requests the controller to start an inquiry, the controller starts the inquiry and returns a command status event. This indicates to the host that the controller has started processing of the inquiry command. Once the inquiry is complete, the controller sends an inquiry complete event to indicate that the inquiry command has been completed. This was shown in Figure 3.17.

3.5.5 Connection Handle

The HCI interface assigns a Connection Handle when a new logical link is created. The value of this connection handle is returned in the Connection_Complete_Event. The host uses this connection handle to manage this connection, send data and finally disconnect the connection.

3.5.6 HCI Transport Layer

The Host Controller Interface defines 4 transport layers that can be used:

1. UART Transport Layer.
2. USB Transport Layer.
3. Secure Digital (SD) Transport Layer.
4. Three-Wire UART Transport Layer.

The UART Transport Layer in described in detail here. This interface is commonly used in systems where the Host and Controller are on the same PCB (Printed Circuit Board). Some examples are Smartphones, Tablets and Printers.

Each packet that is exchanged between the Host and the Controller is prefixed with a packet indicator immediately before the HCI Packet. This allows the differentiation of various packets.

The following Packet Indicators are used on the HCI UART Interface:

1. HCI Command Packet: Packet Indicator 0x01.
2. HCI Asynchronous (ACL) Data Packet: Packet Indicator 0x02.
3. HCI Synchronous (SCO/eSCO) Data Packet: Packet Indicator 0x03.
4. HCI Event Packet: Packet Indicator 0x04.

Figure 3.19 indicates the direction in which these packets are sent along with the packet indicators. (The arrows indicate the direction in which packets are sent.)

In some implementations (e.g., some Smartphones), the HCI Synchronous Data packets are not sent on the UART Transport Layer. Rather these packets are directly sent by the host's application processor or modem to the Bluetooth chip on a different interface. For example a PCM/I2S interface may be used to send voice packets between the Application Processor and Bluetooth Chip. This is shown in Figure 3.20.

3.5.6.1 Decoding HCI Packets

Figure 3.21 shows an example of how to decode the packets on the UART transport layer. This is quite useful when debugging the HCI transactions happening on the UART. Similar method can be used to also debug packets on other transport layers.

In general the UART Transport Layer assumes that the line is free from line errors. (Why? - Since they are closely located on the same PCB so it's assumed that there will not be any errors). In case there is a synchronization loss in the communication from the Host to the Controller, the Controller sends a Hardware Error Event to the Host. Once the Host is aware of the synchronization loss, it has to terminate any pending Bluetooth activity and then reset the Controller to regain

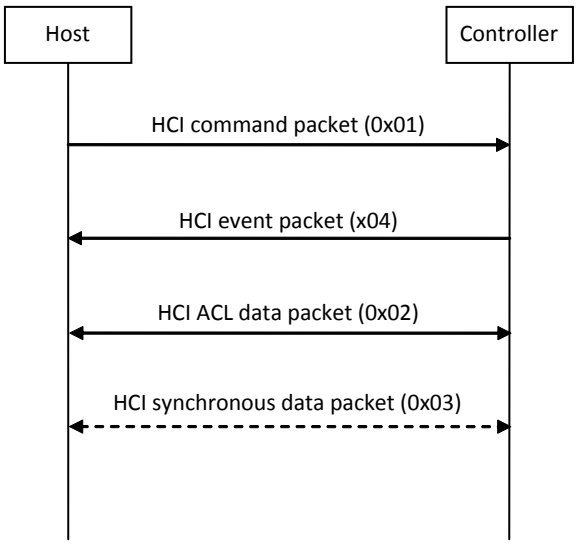

Figure 3.19 HCI UART transport layer.

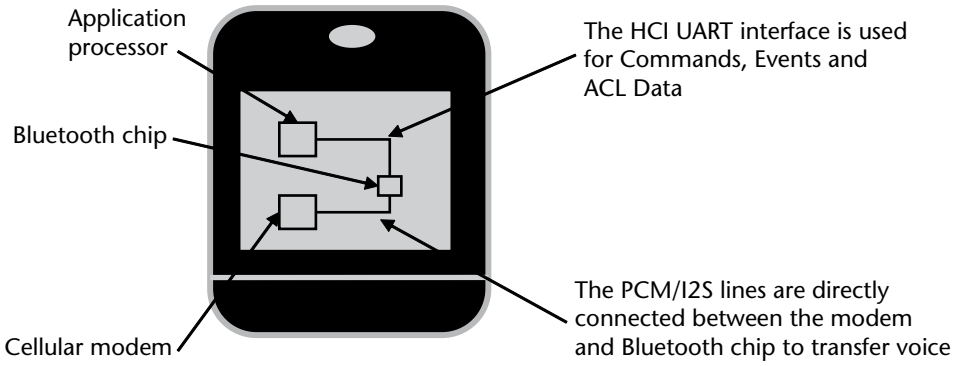

Figure 3.20 Typical HCI UART connections in a smartphone.

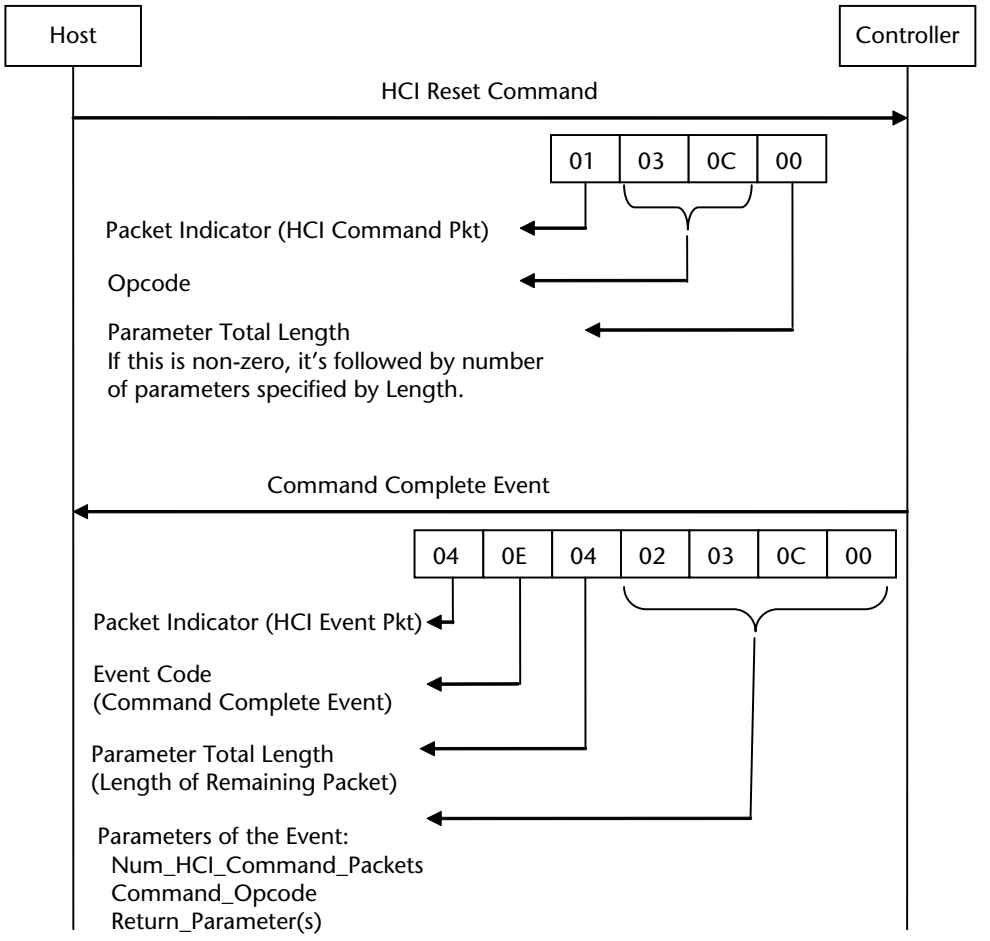

Figure 3.21 How to decode packets sent on HCI UART transport layer.

communications. This is done by the host by sending the HCI_Reset command to the controller.

In order to avoid synchronization losses, the Host and Controller use Hardware Flow Control. For details on this as well as the details on the settings of the UART you may refer to the Bluetooth Core specification (Ref [1]).

3.6 Security—Secure Simple Pairing (SSP)

Secure Simple Pairing was introduced in Bluetooth spec 2.1 + EDR. The main goal was to simplify the pairing procedure from the user perspective. It also introduced improved level of security as compared to previous versions.

SSP increased the security level as compared to previous version of the Bluetooth specification. Till Bluetooth spec 2.0 + EDR, only 4-digit numeric pin keys were supported. On top of that several devices used frequently used pin keys like 0000, 8888, or 1234. So hacking those codes was comparatively easier. SSP introduced 16 alphanumeric pin which makes it more difficult to hack the codes.

SSP has two security goals:

- Passive Eavesdropping Protection.
- Man-In-The-Middle (MITM) Attack Protection.

3.6.1 Passive Eavesdropping Protection

Passive Eavesdropping protection is provided by two mechanisms:

- *Strong Link Key:* The strength of a link key is dependent on the amount of randomness (entropy) in its generation. In previous versions of Bluetooth, the only source of this entropy was the 4-digit numeric PIN key. This was relatively easy to break in short time. In comparison SSP uses 16-bit alphanumeric PIN key which provides for much higher entropy.
- *Strong Encryption Algorithm:* SSP uses Elliptic Curve Diffie Hellman (ECDH) public key cryptography. This provides a high degree of strength against passive eavesdropping attacks. It's also computationally less complex than standard Diffie Hellman (DH76) cryptography and suitable for Bluetooth controllers which may have limited computational power.

3.6.2 Man-in-the-Middle (MITM) Attack Protection

An MITM attack occurs when a rouge device attacks by sitting in the middle of two devices that want to connect and relays messages between them. The two devices believe that they are directly talking to each other without knowing that all their messages are being intercepted and relayed by a third device which is setting between them. This is also known as active eavesdropping.

Let's say devices A and B want to make a connection and M is an attacking device (as shown in Figure 3.22).

M receives all information from A and relays it to B and vice versa. So, A and B have an illusion that they are directly connected. They are not aware of the existence of M between them. Since M is relaying the information between the two

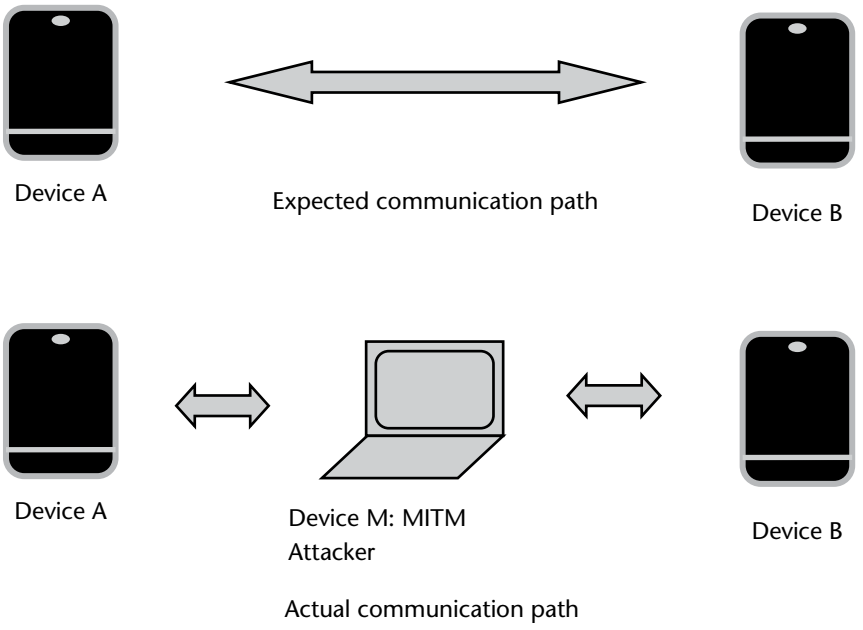

Figure 3.22 Example of MITM attack.

devices, it can interpret all this information and misuse it. Besides this, M can also attack by inducing rogue information between the two devices.

Since A and B can only communicate via M, this type of attack can get detected only if the connection to M is lost. In that case, A and B will not be able to communicate any further and user can detect an MITM attack.

To prevent MITM attacks, SSP provides two mechanisms: Numeric Comparison and Passkey entry. These are described in further detail below.

3.6.3 Association Models

SSP provides four association models based on the I/O capabilities of the two devices. These are as follows:

- Numeric Comparison;
- Just Works;
- Out of Band;
- Passkey Entry.

3.6.3.1 Numeric Comparison

This is used in scenarios where both devices are capable of displaying a six digit number and both are capable of having the user enter a binary "yes" or "no" response. This method displays a 6-bit numeric code on both the devices. The user is then asked whether the number is the same on both the devices. If the user enters yes on both devices, the pairing is successful.

This method has the following advantages:

1. It provides additional confirmation that the correct devices are being paired. This is especially true in cases where the device names are not unique and it's quite hard to remember and identify the device using BD_ADDR
2. It helps to provide protection against MITM attacks.
3. It is also applicable in scenarios where a device may not have a full-fledged keyboard (for entering the PIN) but has only a display. For this method a binary Yes/No input is sufficient.

3.6.3.2 Just Works

As the name implies, this method just works without any user intervention. It's designed for scenarios where the device does not have capability to enter 6 decimal digits nor it has capability to display 6 decimal digits.

It's quite commonly used in pairing mobile phones with mono headsets since mono headsets do not have display and pin entry capabilities. Internally this method still uses the Numeric Comparison though the numbers are not displayed to the user.

It provides protection against passive eavesdropping but does not provide MITM protection. So the security level is still higher than older 2.0 devices but not as good as Numeric Comparison.

3.6.3.3 Out-of-Band (OOB)

Out Of Band Pairing uses an external means for discovering the devices and exchanging pairing information. It's expected that the Out Of Band channel provides protection against MITM attacks to ensure that the security is not compromised.

Typically this could be NFC (Near Field Communication) where the user may touch the two devices. An option would be given to pair the two devices and if the user confirms, the pairing would be successful.

3.6.3.4 Passkey Entry

The passkey mechanism is used when one device has input capability but does not have display capabilities and the second device has display capabilities. One example of such scenario is pairing between keyboard and PC. The user is shown a six digit number on the device which has display capabilities and is then asked to enter this number from the device which has input capabilities. Pairing is successful if the value entered by the user is correct.

3.7 Practical Scenarios

This section describes how the HCI, Baseband Controller and Link Manager interact to implement the following practical scenarios:

- Inquiry;
- Connection Establishment;
- Disconnection.

3.7.1 Inquiry

Inquiry is the procedure to discover other Bluetooth devices in the vicinity. It is also known as scanning or discovering.

Prior to performing an inquiry, it is important that the device that is to be discovered be set in discoverable mode. This includes sending the HCI_Write_Scan_Enable command with *Inquiry Scan* parameter set to *Enabled*. Besides this, HCI_Write_Inquiry_Scan_Activity can also be used to configure additional parameters. Once this is done the device is said to be in inquiry scan mode (or discoverable mode).

The inquiry procedure is started by the host of the first device (which is supposed to discover other devices) by sending an HCI_Inquiry command to the controller. On receipt of this command, the controller initiates an inquiry. The devices in the vicinity respond with inquiry responses.

The controller collects the inquiry responses and provides the results of inquiry to the host in Inquiry_Result events. These events contain the BD_ADDR, Class_Of_Device and Clock_Offset. Multiple devices may be reported in a single Inquiry_Result event. Once the inquiry is complete, an Inquiry_Complete event is sent by the controller to the host (on the device that initiated the inquiry).

This is illustrated in Figure 3.23. (Some of the events are omitted to aid simplicity).

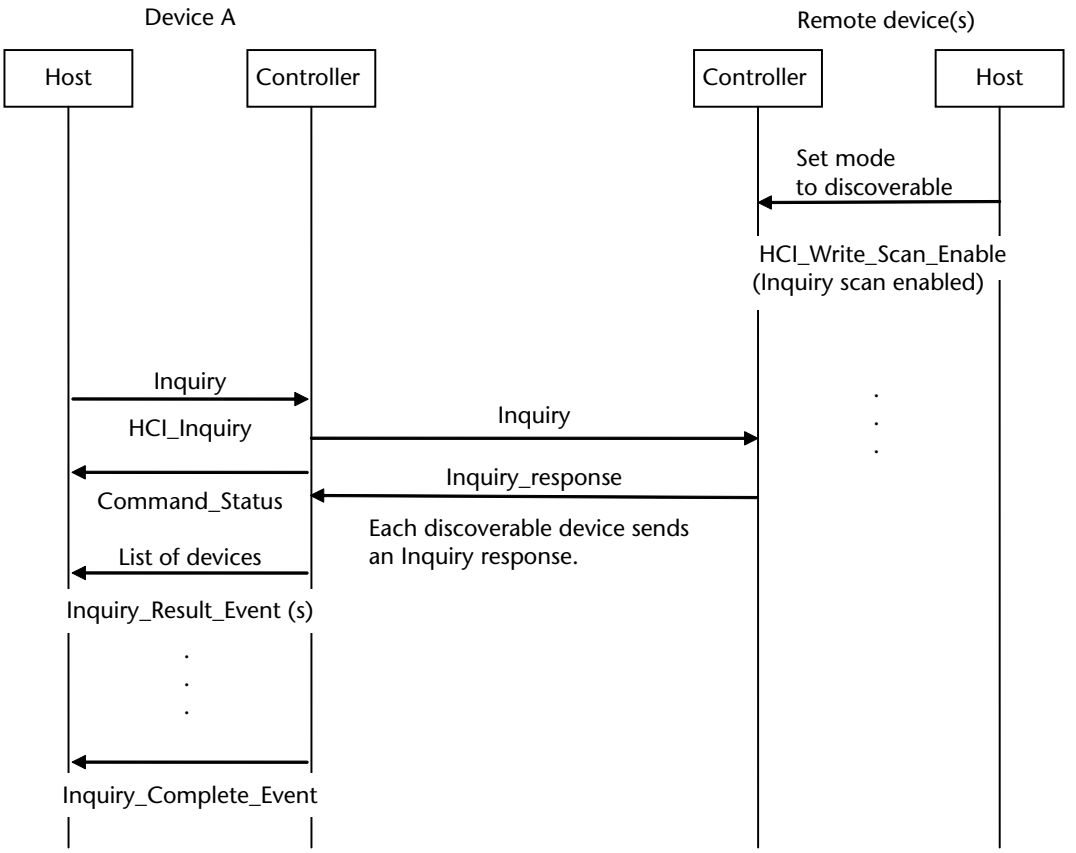

Figure 3.23 Inquiry procedure.

3.7.1.1 Periodic Inquiry

If the inquiry procedure is to be repeated periodically, then HCI_Periodic_Inquiry_Mode command can be used. This command takes the maximum and minimum period between consecutive inquiries as parameters. Once this command is given by the host, the controller performs an automatic inquiry periodically. This can be stopped by the host by sending the HCI_Exit_Periodic_Inquiry_Mode command.

A practical use of this is when a device wants to be regularly alerted of any devices which are coming in or going out of the vicinity. To achieve this periodic inquiry can be used. The time range between two consecutive inquiries is provided as a parameter. The new device list can be compared with the previous device list. If a device was not present in the previous list but is present in the new list then it has entered the Bluetooth vicinity recently. Any specific action like finding details about the device, connecting to it, sending a message or file to it can then be initiated.

3.7.1.2 Extended Inquiry Response (EIR)

The EIR enhancement was added in Bluetooth 2.1 + EDR version of the specification. If a device supports EIR it can provide some additional data while responding to the inquiry. This extended data may contain the name of the device, supported services, Received Signal Strength Indicator (RSSI) etc.

This is more efficient than a normal inquiry response since the inquiring device gets all the information in one go instead of first doing an inquiry, then a get name and finally a service search. This leads to an overall faster connection setup procedure.

3.7.2 Connection Establishment

The procedure to connect to a remote device is known as paging or connecting procedure. In the text below, the device that initiates the connection is referred to as the initiator and the device to which it connects is referred to as the target.

Prior to connecting, the target device (that is to be connected to) is set to Connectable mode. This is done by sending the HCI_Write_Scan_Enable command. This command takes Page_Scan as one of the parameters. The Page_Scan parameter is set to Enabled to make the device connectable. Besides this, HCI_Write_Page_Scan_Activity can be used to configure additional parameters. Once this is done the device is in page scan mode (or Connectable Mode).

Connection procedure is started by the host of the initiator by sending an HCI_Create_Connection command to the controller. The parameters of this command include BD_ADDR, Packet Types, Page_Scan_Repetition_Mode, Clock_Offset and Allow_Role_Switch.

On receipt of this command the controller initiates a connection request using the link manager command LMP_host_connection_req.

Figure 3.24 depicts a very simplified view of the connection establishment procedure. The optional parts like Feature Exchange, Authentication, Encryption and some of the HCI events are omitted to aid simplicity.

The initiator device becomes the Master after successful connection establishment.

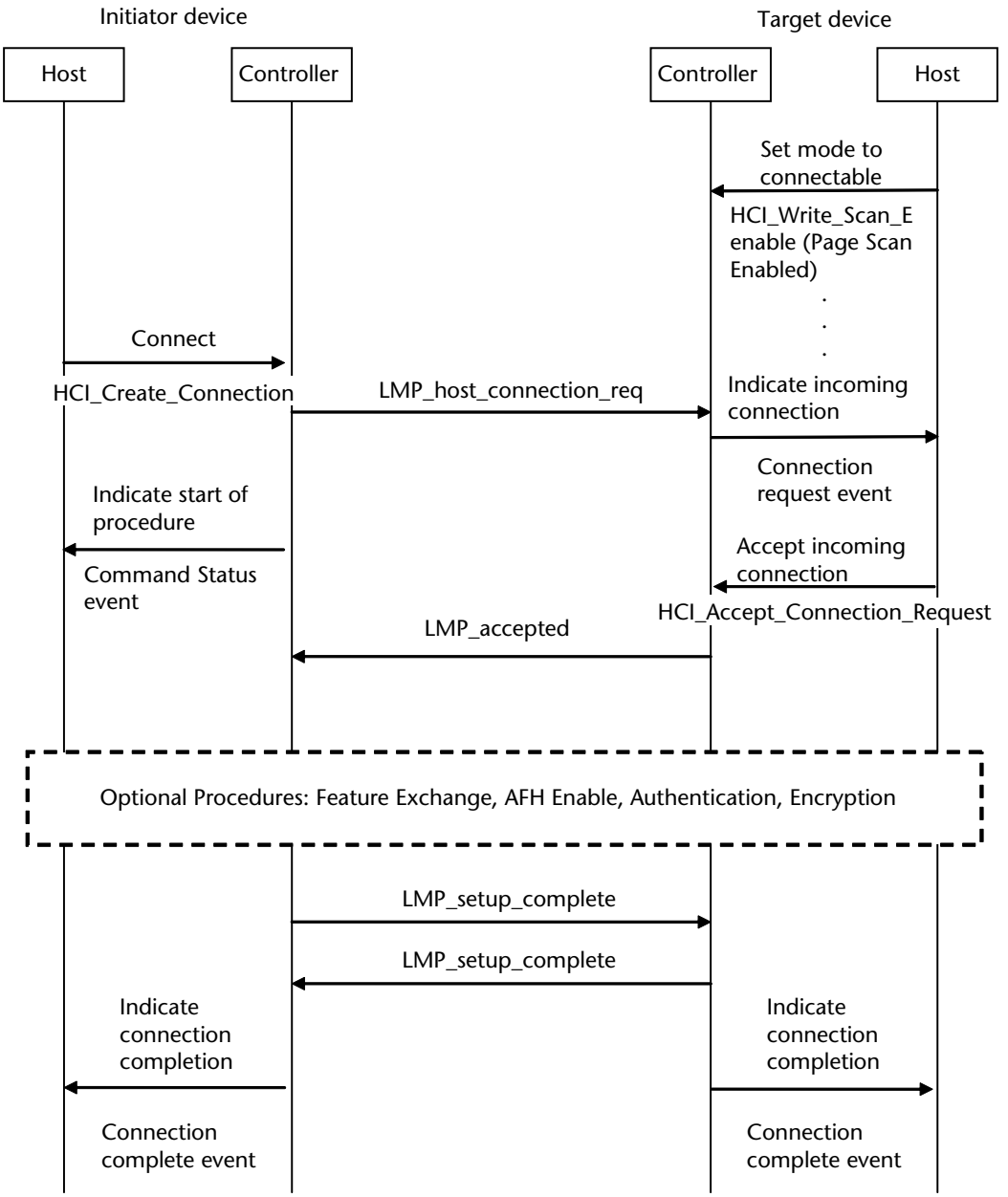

Figure 3.24 Simplified connection establishment procedure.

Figure 3.25 illustrates the disconnection procedure. Either of the devices may decide that the connection is no longer needed and initiate this procedure.

3.8 Summary

This chapter explained the lower layers of the Bluetooth Protocol stack. These include the stack layers up to the HCI interface: Bluetooth Radio, Baseband Controller, and Link Manager. These layers are typically implemented in a controller.

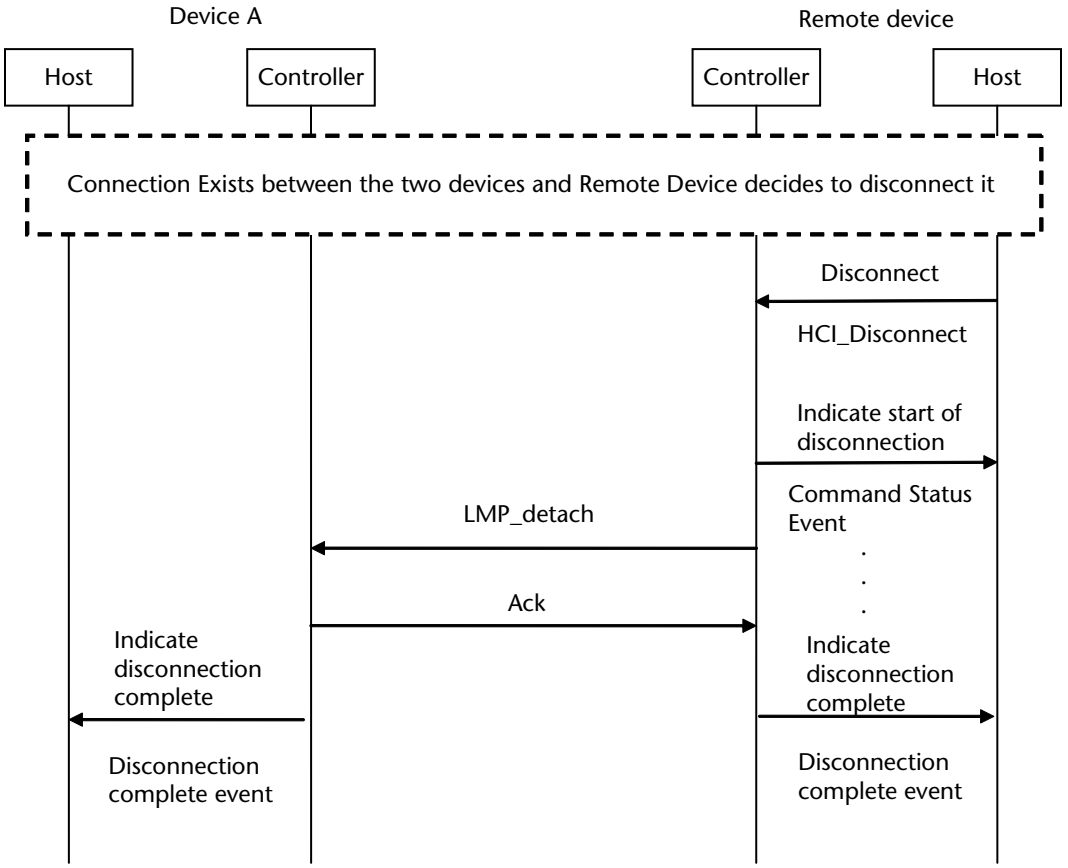

Figure 3.25 Disconnection procedure.

The Bluetooth Radio is responsible for transmitting and receiving the packets to and from the air interface. It uses frequency hopping across 79 channels in the ISM band to combat interference. The Baseband Controller is responsible for carrying out procedures like inquiry, connection, formation of piconet and scatternet, connection states, and low power modes. The Link Manager provides the functionality of link setup and control, security, Master-Slave role switch, etc.

The HCI interface provides a standard mechanism for interfacing Bluetooth upper layers with the controller.

The next chapter will be the last in the series of chapters explaining the Bluetooth architecture. It will cover the Bluetooth upper layers.

Reference

[1] Bluetooth Core Specification 4.0 http://www.bluetooth.org.

CHAPTER 4

Bluetooth Upper Layers and Profiles

4.1 Introduction

The previous chapter described the lower layers of the Bluetooth protocol stack. This chapter continues explanation of the Bluetooth protocol stack and covers the upper layers of the Bluetooth protocol stack and the profiles.

The detailed Bluetooth architecture was presented in Chapter 2. It is shown again in Figure 4.1 for ease of reference. The upper layers make use of the functionality provided by lower layers to provide more complex functionality like serial port emulation, transferring big chunks of data, streaming music, and synchronizing information. These help the applications to conveniently implement end user scenarios.

One of the design principles of the Bluetooth protocol stack was the reuse of existing protocols wherever possible instead of rewriting everything from scratch. On one hand this helped to easily and quickly build further on existing and proven technologies, on the other hand it also helped in reusing existing applications that were already implemented and available. Protocols such as Object Exchange (OBEX) and RFCOMM were adopted from other standard bodies and are referred to as *adopted protocols*. These are shown by shaded rectangles in Figure 4.1. OBEX was adopted from the IrOBEX protocol which was defined by the Infrared Data Association. So applications that were designed to run on Infrared transport can generally be reused to run on Bluetooth as well. Similarly RFCOMM was adopted from ETSI standard 07.10. It allows legacy serial port applications to be used on top of Bluetooth without much change.

Some protocols were defined from scratch. These were the protocols that provided the core Bluetooth functionality and are called *core protocols*. These are shown by plain rectangles in Figure 4.1. For example the Service Discovery Protocol (SDP) is one of the core protocols that allows for support of discovering the services of the devices in the vicinity. Since Bluetooth is an ad hoc peer to peer protocol, this layer was essential because there is no prebuilt infrastructure to provide such information and devices can come into vicinity at any time. So this protocol was defined to query the services from the device itself instead of the need of a central server to store information about all devices.

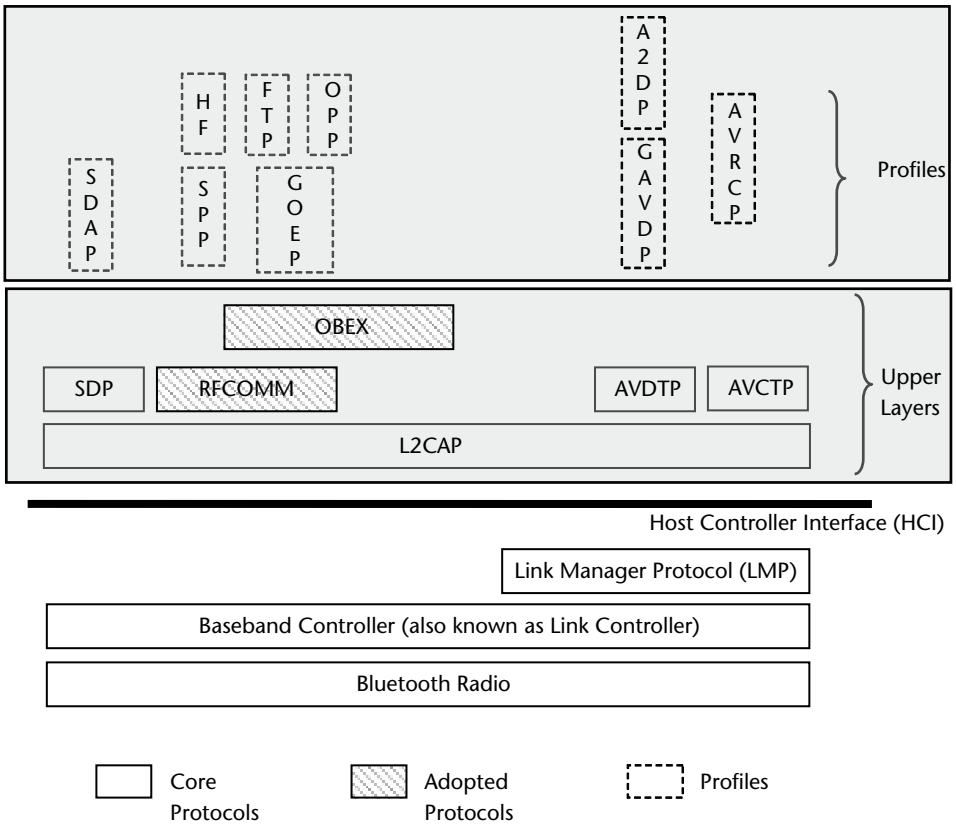

Figure 4.1 Detailed Bluetooth architecture.

Bluetooth profiles provide a usage model of how the different layers of the protocol stack come together to implement a particular usage model. Profiles define the protocols and the features of each of the protocol that are needed to support a particular usage model. For example the File Transfer Profile (FTP) profile defines how and what features of the underlying protocols like OBEX, RFCOMM and L2CAP are needed in order to provide support for transferring files. Profiles are shown by dotted rectangles in Figure 4.1.

4.2 Logical Link Control and Adaptation Protocol (L2CAP)

The L2CAP protocol sits above the Baseband layer and provides data services to the upper layer protocols. It uses the ACL links to transfer packets and allows the protocols above it to send data packets up to 64 KB in length.

L2CAP is based on the concept of Channels. A channel represents a data flow path between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless.

Connection-oriented L2CAP channels are used to transport point-to-point data between two devices. These channels provide a context within which specific properties may be applied to data transported on these channels. For example QoS

parameters may be applied to the connection-oriented channels. These channels are setup using the L2CAP_CONNECTION_REQUEST command before any data can be transferred. Once data transfer is completed, these channels are disconnected using the L2CAP_DISCONNECT_REQUEST command.

Connectionless L2CAP channels are generally used for broadcasting data though they may also be used for transporting unicast data. The Master uses these channels to broadcast the data to all Slaves in the piconet. These channels do not need separate procedures to setup and disconnect the channel. So latency incurred during the channel setup is removed.

Each endpoint of an L2CAP Channel is referred to as a Channel Identifier (CID). So CID is the local name representing a logical channel end point on the device. CIDs are assigned from 0x0001 to 0xFFFF. Out of these, CIDs from 0x0001 to 0x003F are reserved. These are known as fixed channels. The CID 0x0001 is reserved as the L2CAP signaling channel and 0x0005 is reserved as the L2CAP LE signaling channel. CID assignment is specific to the device and is done independent of the other devices. So it's possible, for example, to have one CID number assigned on one device and a different CID number assigned on the other device by the L2CAP entities executing on the respective devices.

The fixed channels (0x0001 for BR/EDR and additionally 0x0005 for LE) are available as soon as the ACL-U logical link is established with the remote device. The L2CAP Signaling Channel is used for negotiating configuration parameters and setting up the other channels.

The allocation of CID numbers is shown in Table 4.1.

Table 4.1 CID Name Space

CID	*Description*
0x0000	Null Identifier. Usage is not allowed.
0x0001	L2CAP Signaling channel. Used to send the signaling commands in the form of requests and responses. Some examples of signaling commands are: • Connection Request • Connection Response • Configuration Request • Configuration Response • Disconnection Request • Disconnection Response
0x0002	Connectionless channel. Used for the following: • Broadcast from Master to all Slaves in the piconet. There is no acknowledgement or retransmission of this data. • Unicast transmission from either a Master or Slave to a single remote device.
0x0003	AMP Manager Protocol. This is used for BT 3.0 + HS operations.
0x0004	Attribute Protocol. Attribute Protocol will be covered in detail in later chapters.
0x0005	LE L2CAP Signaling Channel.
0x0006	Security Manager Protocol (SMP): SMP will be covered in detail in later chapters.
0x0007 – 0x003E	Reserved for future use
0x0040 – 0xFFFF	Dynamically allocated by the L2CAP layer.

4.2.1 Modes of Operation

There are six modes of operation for L2CAP. The mode of operation is selected independently for each channel.

1. *Basic L2CAP Mode:* This is the default mode. The header contains only minimal information including the length of the packet and the channel ID.
2. *Flow Control Mode:* In flow control mode no retransmissions are done. The missing PDUs are detected and reported as lost.
3. *Retransmission Mode:* In retransmission mode a timer is used to ensure that all PDUs are delivered to the peer. The PDUs are retransmitted if they are not ack'ed by the remote side.
4. *Enhanced Retransmission Mode:* The enhanced retransmission mode is similar to the retransmission mode and adds some enhancements to it. For example it adds a POLL bit to request a response from the remote L2CAP layer.
5. *Streaming Mode:* The streaming mode can be used for all streaming applications. In this mode, the PDUs from the transmitting side are numbered but are not acknowledged by the receiver. The numbering of PDUs ensures that they are processed in the correct sequence on the receiving side. A flush timeout is used so that if the PDUs are not sent within that timeout, they are flushed.
6. *LE Credit-Based Flow Control Mode:* This mode was introduced in specifications 4.1. Support for connection-oriented channels for LE was introduced in this version. At the time of connection establishment, each side provides the number of credits that are available. The number of credits indicates the number of LE-frames that the device is capable of accepting. The remote side can send as many LE-frames as the number of credits it has received; if the credits become zero, it stops sending packets. As and when the packets are processed on the receiver side and buffer space becomes available, more credits are given to the transmitter side so that it can send more packets.

4.2.2 L2CAP PDUs

The L2CAP Protocol Data Unit (PDU) is the term used to refer to the packets that are sent and received the L2CAP layer. The PDUs contain control information or data.

L2CAP defines 5 types of PDUs:

1. *B-frame (Basic Frame):* A B-frame is a PDU used in basic L2CAP mode for L2CAP data packets.
2. *I-frames (Information Frame):* An I-frame is a PDU used in enhanced retransmission mode, streaming mode, retransmission mode and flow control mode. It contains additional information encapsulated in the L2CAP header.

3. *S-frame (Supervisory Frame):* An S-frame is a PDU used in Enhanced retransmission mode, retransmission mode and flow control mode. It contains protocol information only and no data.
4. *C-frame (Control Frame):* A C-frame is a PDU that contains L2CAP signaling messages that are exchanged between peer L2CAP entities. This frame is exchanged on the L2CAP signaling channel.
5. *G-frame (Group Frame):* A G-frame is used on the connectionless L2CAP channel. It may be used to broadcast data to multiple Slaves or unicast data to a single remote device.

4.2.3 L2CAP Features

L2CAP performs the following major functions:

- Higher layer Protocol Multiplexing and Channel Multiplexing;
- Segmentation and Reassembly (SAR);
- Per Channel Flow Control;
- Error Control and Retransmissions;
- Streaming Channels;
- Quality of Service;
- Group Management.

L2CAP uses BR/EDR Controller, LE Controller or a Dual mode controller for transporting data packets. It can also use AMP controllers if they exist.

4.2.3.1 Higher layer Protocol Multiplexing and Channel Multiplexing

L2CAP permits several higher layer protocols to share the same ACL links. Each higher layer protocol is assigned a separate CID to transfer data. For example once an L2CAP channel is established with the remote device it may be shared by SDP, RFCOMM and other protocols. Each of these protocols will use a different CID to talk to the respect peer entity on the remote device. This is shown in Figure 4.2.

The Protocol/Service Multiplexer (PSM) field is used during L2CAP connection establishment to identify the higher level protocol that is making a connection on that particular channel. The PSM values are predefined for many of the protocols by the Bluetooth SIG and can be referred to on the Assigned Numbers page on the Bluetooth SIG website. Some of the PSM values for the protocols are show in Table 4.2.

L2CAP also allows the channel to be operated over different controllers though the channel can be active on only one controller at a time. This is useful in scenarios where the channel is initially established over BR/EDR controller and then it's moved to an AMP controller to achieve higher data throughput.

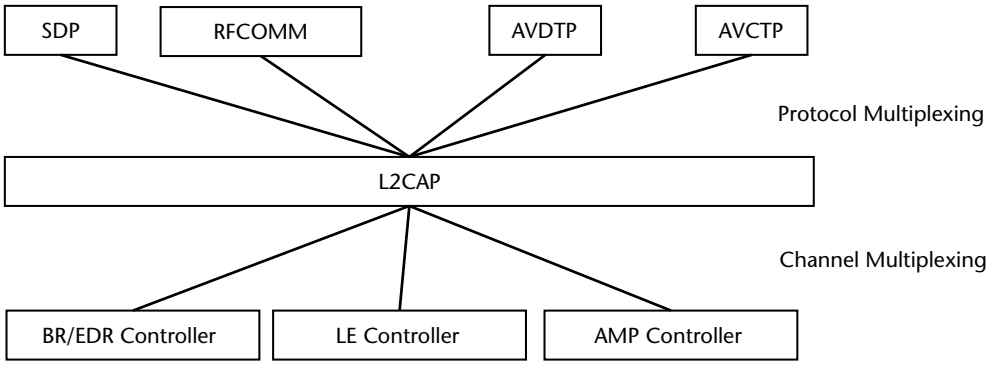

Figure 4.2 L2CAP: protocol/channel multiplexing.

Table 4.2 Assigned Protocol/Service Multiplexer Values (PSM)

Protocol	*PSM*	*Remarks*
SDP	0x0001	Service Discovery Protocol
RFCOMM	0x0003	RFCOMM Protocol
TCS-BIN	0x0005	Telephony Control Specification, TCS Binary Protocol
TCS-BIN-CORDLESS	0x0007	Telephony Control Specification, TCS Binary Protocol
BNEP	0x000F	Bluetooth Network Encapsulation Protocol
HID_Control	0x0011	Human Interface Device Profile
HID_Interrupt	0x0013	Human Interface Device Profile
UPnP	0x0015	
AVCTP	0x0017	Audio/Video Control Transport Protocol
AVDTP	0x0019	Audio/Video Distribution Transport Protocol
UDI_C-Plane	0x001D	Unrestricted Digital Information Profile.

4.2.3.2 Segmentation and Reassembly (SAR)

L2CAP allows higher layer protocols to transmit and receive data packets up to 64 kilobytes. While transmitting, L2CAP breaks the packets into smaller packets depending on the packet size supported by the controller. On the receiving end, L2CAP receives all these segments and reassembles them to form the complete packet. This complete packet is then sent to the higher layers.

As an example, let's say RFCOMM sends a packet of 60 KB to L2CAP. L2CAP will break this packet into smaller chunks of 1021 bytes which can then be transferred over a BR/EDR controller (Note that this is the maximum packet size that can be transmitted over ACL using 3-DH5 packets. A BR/EDR controller may support this as the maximum size or a smaller size). This will result in 60 chunks of 1021 byte packets and one chunk of 180 bytes.

$$60\ \text{KB} = 61440\ \text{Bytes} = 60 * 1021 + 1 * 180$$

All these 61 chunks will be reassembled by the L2CAP on the receiving side and the original 60 KB RFCOMM packet will be recreated. This packet will then be given to the RFCOMM of the remote side.

(Note that the above example is a bit simplified. In practice, L2CAP will also prefix its own 4 byte header to the RFCOMM packet. So it will actually be segmenting and reassembling a packet of 61444 bytes. (61440+ 4 byte L2CAP header).)

Segmentation and Reassembly of packets provides the following advantages:

1. The higher layer protocol doesn't need to care about the size of packets that can be transmitted over the ACL link. The L2CAP layers of the peer devices negotiate a mutually suitable MTU size and use it for the data transfer. This makes the design of the higher layer protocol simpler.
2. Since the packet is split into smaller chunks, if there is an error in transmission of one of the chunks then only that chunk will be retransmitted. The whole packet doesn't need to be retransmitted.
3. L2CAP can interleave packets of different higher layer protocols. This ensures that if one of the higher layer protocols is sending a big packet, the other protocols don't get starved for bandwidth.
4. In general, the controllers have limited buffer space for keeping transmit and receive packets while the host may have much larger buffer space. So L2CAP allows the upper layer protocols to send bigger data packets even though the controller may support much smaller packet sizes.

The MTU (Maximum Transmission Unit) is specified by each device independently and is not negotiated between the two L2CAP entities. This means, for example, if a mobile phone is connected to the headset then the mobile phone may specify a higher MTU size while the headset may specify a smaller MTU size. The mobile phone will always send packets which are smaller than or equal to in length to the MTU size of the headset.

The MTU parameter is very significant in cases where speed of data transfer is important. Lower MTU values will result in the need of segmenting a big chunk into more number of smaller packets. This will decrease the overall speed of data transfer.

4.2.3.3 Per Channel Flow Control

Flow control mechanisms are already in place for the data that is transferred on the HCI interface but that is at an aggregate level to control the data that is flowing from the host to the controller and on the air from one controller to another. L2CAP extends this mechanism to provide individual flow control to each of the channels that are multiplexed on top of it. This means that the data for the RFCOMM layer may be stopped while the data for the SDP layer may still be allowed to flow.

4.2.3.4 Error Control and Retransmissions

The first level of error control is done by the baseband layer. If the received packet has an error, then it is not passed on to the host. L2CAP provides an additional level of error control that detects any erroneous packets that are not detected by

the baseband. L2CAP retransmits packets if they are not received correctly on the remote side. Besides this, it's possible that some of the packets are dropped by the time they reach the host entity on the remote side. This could be, for example, if the packet had an error and was dropped by the baseband layer. The L2CAP layer retransmits these packets so that the layers above it receive all the packets in the correct sequence without any errors.

L2CAP uses a Transmit/Acknowledge mechanism. Each packet that is transmitted is acknowledged by the remote device. If the acknowledgement is not received, then L2CAP may decide to either retransmit that packet or to drop it. Whether packets on a certain channel are to be retransmitted or not is specified using the Flush Timeout option. There are three possible values:

1. No retransmissions at the baseband level.
2. Use a specified flush timeout and continue retransmitting till the time the timeout expires.
3. An infinite amount of retransmissions. In this case, the baseband continues retransmissions until the physical link is lost.

4.2.3.5 Streaming Channels

Streaming channels are useful in scenarios where data with a specific data rate (like audio) is being transferred. The audio applications can setup an L2CAP channel with the specific data rate. A flush timeout is used if L2CAP is not able to transfer packets within the correct time period to comply with that data rate. This ensures that packets don't get queued up indefinitely if, for example, the link quality deteriorates. Since audio packets are real time in nature, it's better to drop a delayed packet and transmit the next packet than to continue trying to retransmit the delayed packet.

4.2.3.6 Quality of Service

Isochronous data has time constraints associated with it. The information has a time bound relation with the previous and successive entities. Audio is a good example of isochronous data. For such a data, the lifetime is limited after which the data becomes invalid and there is no point in delivering that data anymore.

L2CAP can support both isochronous (Guaranteed bandwidth) and asynchronous (Best Effort) data flows over the same ACL logical link. This is done by marking the isochronous packets as automatically flushable and asynchronous packets as nonflushable in the Packet_Boundary_Flag in HCI ACL data packets. This flag was explained in detail in the HCI section in previous chapter. The automatically flushable packets will be automatically flushed by the controller if they are not transmitted within the time window of the flush timeout set for the ACL link.

4.3.3 L2CAP Signaling

The Signaling commands are used between L2CAP entities on peer devices for operations like setting up the connection, configuring the connection, and disconnection. The fixed channels (0x0001 and additionally 0x0005 for LE) are available as soon

as the ACL-U logical link is established with the remote device. The L2CAP Signaling Channel is used for negotiating configuration parameters and setting up the other channels. The L2CAP commands are encapsulated within C-Frames (control frames).

The format of C-Frames is shown in Figure 4.3. The Channel ID is 0x0001 for BR/EDR signaling and 0x0005 for LE signaling. The Code field identifies the type of command. The Identifier field is used to match responses with the requests.

The various L2CAP signaling packets are shown in Table 4.3.

Out of the commands mentioned in Table 4.3, Connection Parameter Update Request and Response are used only for LE. Command Reject can be used for both LE and BR/EDR. These will be explained in detail in Chapter 10.

An example of various L2CAP Signaling PDUs is shown in Figure 4.4. This figure shows an air sniffer capture of L2CAP signaling packets when RFCOMM uses the services of L2CAP.

Some of the points worth noting are:

1. The Signaling packets are being exchanged on Signaling channel (CID 0x0001). See Frames #22–#29.
2. The RFCOMM data packets are exchanged on CID 0x0040. This is the CID allocated to RFCOMM.
3. During the Connection Request, the master provided the following information:
   a. Source Channel ID = 0x0040: The CID that the master will be using.
   b. PSM = RFCOMM: To indicate that L2CAP is creating a channel for RFCOMM to use.
4. The configure request and configure response packets are exchanged between the Master and Slave in both directions to negotiate the connection parameters.
5. Once the data transfer is done, the Master sends a disconnection request on the Signaling channel.

4.4 Service Discovery Protocol (SDP)

Bluetooth provides support for ad hoc connections. This means devices can dynamically discover each other and then decide to connect to each other. Before

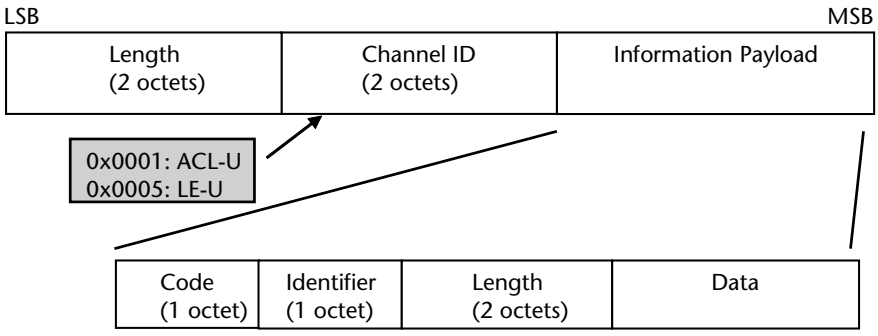

Figure 4.3 Format of L2CAP signaling PDUs (C-Frames).

Table 4.3 L2CAP Signaling Packets

Code	*Signaling Packet*	*Purpose*
0x00	Reserved	
0x01	Command Reject	This is sent in response to a command packet that contains an unknown command or when sending the corresponding response is inappropriate.
0x02	Connection Request	This is used to create an L2CAP channel between two devices.
0x03	Connection Response	This is sent in response to a Connection Request.
0x04	Configure Request	This is used to negotiate configuration parameters of the connection. These include parameters like MTU, Flush Timeout, QoS, etc.
0x05	Configure Response	This is sent in response to Configure Request.
0x06	Disconnection Request	This is used to terminate an L2CAP channel.
0x07	Disconnection Response	This is sent in response to Disconnection Request.
0x08	Echo Request	This is used to request a response from the remote L2CAP entity. This request is generally used to check the status of the link.
0x09	Echo Response	This is sent in response to Echo Request.
0x0A	Information Request	This is used to request implementation specific information from the remote L2CAP entity.
0x0B	Information Response	This is sent in response to Information Request.
0x0C	Create Channel Request	This is used to create an L2CAP channel between two devices over the specific controller. This is used when BT 3.0 + HS is supported and an alternate controller is used.
0x0D	Create Channel Response	This is sent in response to Create Channel Request.
0x0E	Move Channel Request	These are used to move an existing L2CAP channel from one controller to another. These are used when BT 3.0 + HS is supported.
0x0F	Move Channel Response	
0x010	Move Channel Confirmation	
0x11	Move Channel Confirmation Response	
0x12	Connection Parameter Update Request	This command is sent from an LE Slave to an LE Master to update the connection parameters. This will be described in further detail in Chapter 10.
0x13	Connection Parameter Update Response	This is sent in response to Connection Parameter Update Request.

making the decision to connect to each other, one device may need to "discover" details about the other device. These details include which services are available and what are the characteristics of those services. Service discovery protocol provides a mechanism to discover the services provided by the remote devices and the characteristics of those services.

As an example, let's say a laptop needs to play an audio file on Bluetooth wireless speakers. It will first do an inquiry to find out the devices in the Bluetooth vicinity. Once this is done, it will connect to these devices and search for the services provided by those devices. In order to play an audio file, it will search for a device that has the services registered for A2DP. Once it finds such a device, it may create an A2DP connection with that device to play the file.

(Note: In practice the laptop need not connect to all the devices to discover the A2DP service. It can already narrow down its search based on the Class of Device (CoD) information provided by that device during inquiry. It needs to only discover services on the devices of the Audio/Video Major class.)

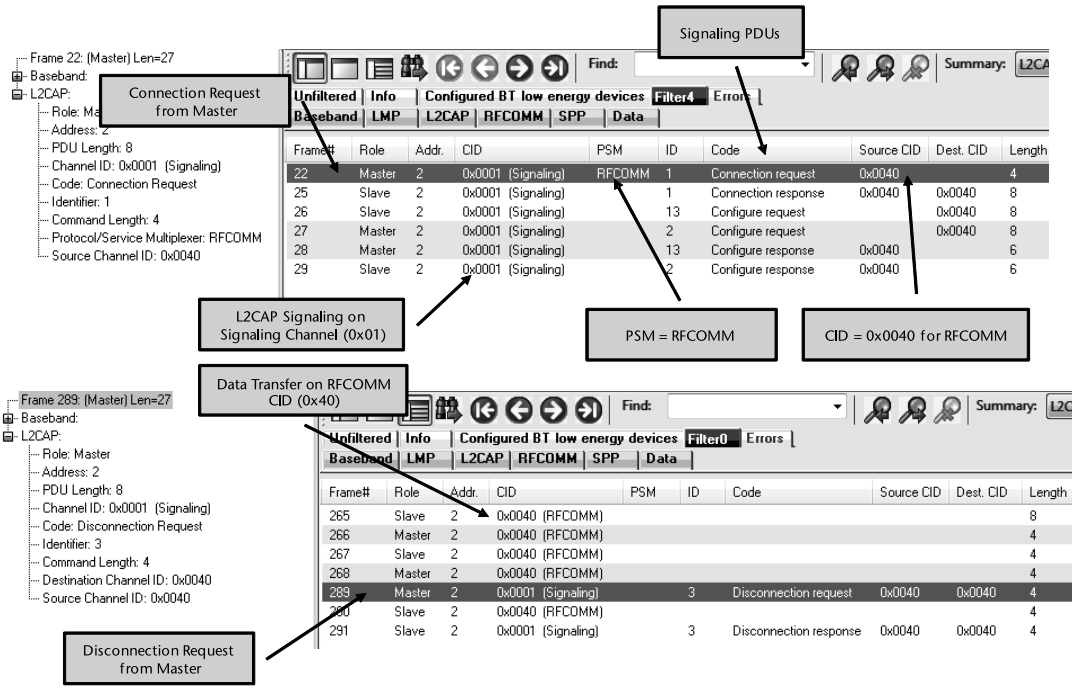

Figure 4.4 Air sniffer capture of L2CAP signal packets.

SDP focuses primarily on providing a uniform method for discovering services available on the Bluetooth devices. It does not define the method to access those services once they are discovered. This is done by the other layers depending on the type of service. For example if a device provides a printing service, then the printing related profiles will make use of that service to print documents.

SDP follows a client server model. The services supported by a device are registered with an SDP server. The server maintains a list of these services in the form of service records. The SDP client queries for these services.

This is illustrated in Figure 4.5.

Only one SDP server is permitted per Bluetooth device. If the device does not need to provide any services to other devices, it can act as a client only device. In such cases it does not need to implement an SDP server.

4.4.1 Service Record, Service Attributes and Service Class

All the server applications register their services with the SDP server in the form of Service Records. A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. It may be implemented as software, hardware or a combination of both. For example a printer could provide the service of color printing; a smartphone could provide the service of a dial-up-networking gateway.

A service record contains all information that an SDP server wants to provide about the service to the SDP clients. This is in the form of a list of Service Attributes. Each Service Attribute describes a single characteristic of the service and is the form of an Attribute ID and Attribute Value. An Attribute ID is a 16-bit

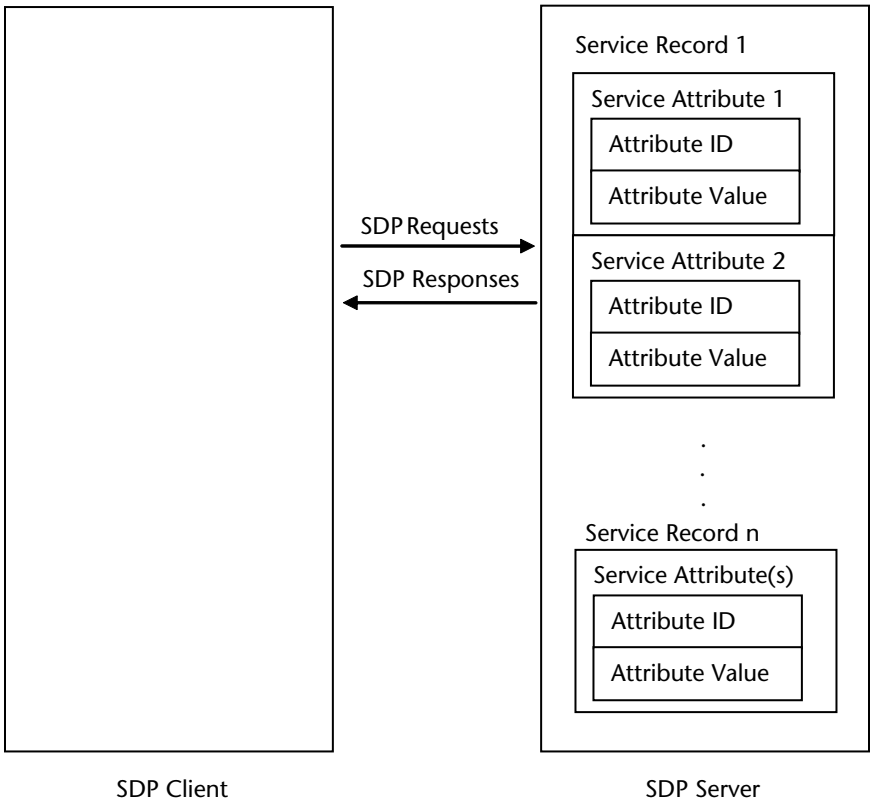

Figure 4.5 SDP Client and Server.

unsigned integer that distinguishes each service attribute from other service attributes within a service record. An Attribute Value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained.

Each service is an instance of a service class. The service class definition provides the definitions of all service records that can be present in the service. Each service class is assigned a unique identifier called the service identifier. It is represented as a Universally Unique Identifier (UUID).

What is UUID?

A UUID is a universally unique identifier that is guaranteed to be unique across all space and time. UUIDs can be independently created in a distributed fashion. No central registry of assigning UUIDs is required.

A UUID is a 128-bit value. To reduce the burden of storing and transferring 128-bit values, a range of UUIDs has been pre-defined along with 16-bit or 32-bit aliases. A 16-bit or 32-bit UUID may be converted into 128-bit UUID by pre-defined formulas.