Low Energy Data Packet Extensions

One of the most important changes in 4.2 specifications is that the length field has been increased from 5 bits to 8 bits. This has led to an increase in the supported packet size for data packets from 27 to 251 bytes. This is an almost ten-fold increase and is useful in the following cases:

1. Over the air (OTA) firmware upgrades: After the device has been deployed in the field, there may be newer versions of the software containing additional features and bug fixes. The most convenient way to update the firmware is through OTA updates, but with a packet size of 27 bytes, an OTA update would take a long time. Depending on the firmware size, this may take several minutes. Besides taking time, this would also lead to significant power consumption because the device would have been active for a long duration, thereby reducing its battery life. With the increase in packet size, the firmware updates would take a fraction of the time and the energy as compared to the time and energy taken by 4.0 compliant devices.
2. Uploading the logs: One of the primary usages of BLE devices is in sensor tags, where the tags keep collecting the data and then upload it to the Internet. Sometimes, this data may be huge (i.e., a person's temperature and heart rate during the day if a reading is taken every minute), and transporting it on 27-byte data packet sizes would take a lot of time. Transporting it on bigger packets would be much faster and more power efficient.

In general, larger packet sizes are more efficient when compared to smaller ones. This is because the fractional overhead of the bytes used for the header is less. For example, if a total of 6 bytes were used for the header (including bytes used in the lower layers of the protocol stack and message integrity check), then the overhead for 27-byte packets would be 6/27 (22%), while that for 251-byte packets would be 6/251 (2%). Also, increased data transfer speeds and packet sizes reduce the window and chances of transmission losses, which would help in getting the important packets in time to or from the sensor device (for example, a hearing aid or a critical medical equipment).

Besides this, there would be less processing power consumed in fragmenting the packets at the transmitter end and reassembling them at the receiver end. Fragmentation also makes it worse because the packets are only sent at connection intervals, and there is a dead time between connection intervals. This translates to a longer transmission time, effectively keeping the device powered on longer.

8.10 Bit Stream Processing

Figure 8.12 shows the sequence of steps that are carried out by the link layer before transmitting data and the link layer on the other side after receiving the data. The data to send is treated as a bit stream with the LSB first. At the time of transmission, the different steps that are performed are encryption of the data follower by CRC

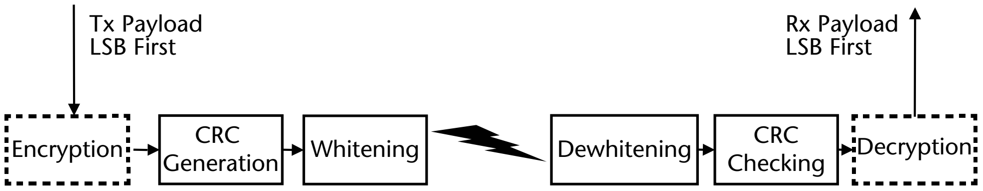


Figure 8.12 Link layer bit stream processing.

generation and then finally whitening. When the data is received on the receiving side, exactly the reverse of these steps are performed.

The Encryption stage on the transmitter side and the decryption stage on the receiver side are optional. Encryption is done only if the host requested an encrypted link.

Data whitening is the process used to avoid long sequences of zeros or ones while transmitting. Before transmitting the header and payload are scrambled with a data whitening word. This randomizes the data to reduce the possibilities of long sequences of zeros or ones. At the receiver end, the data is descrambled using the same data whitening word to get back the original data.

When a packet is received, the first step is to check for errors. This includes the following:

- Check the Access address to ensure that the packet is meant for the channel that the link layer is connected to.
- CRC checking.

One of the optimizations done by LE is that the encryption is done before CRC generation while transmitting data and decryption is done after CRC checking while receiving data. This is opposite to BR/EDR where CRC generation is done before encryption while transmitting and CRC checking is done after decryption while receiving. So in the case of LE, there are the following advantages:

1. CRC checking takes far lesser time as compared to decryption while receiving. So if CRC checking is done before decryption:
   a. The received packet can be acknowledged as soon as the CRC check is complete. So the radio can be immediately switched off instead of waiting for the complete decryption process.
   b. The complete decryption process can then be done offline when the radio is switched off. This helps in reducing the peak power consumption since only decryption is going on and the radio has been switched off.
   c. If the packet got corrupted by the time it was received, then the CRC check will detect it early and the packet can be dropped immediately. For such packets power does not need to be consumed while doing the decryption.

8.11 Link Layer States

The five link layer states were briefly introduced at the beginning of this chapter. This section provides a detailed explanation of each of these states. Broadly the states can be categorized into Non-Connected States and Connection States as shown in Figure 8.13.

To reduce the complexity, LE does not allow scatternet scenarios. (Note that a scatternet is a combination of multiple piconets. It is formed in BR/EDR scenarios when one of the devices acts as Slave in two piconets or as Master in one piconet and Slave in another piconet.)

This imposes the following restrictions on LE devices:

1. The device cannot act as Master and Slave at the same time.
2. The device cannot also act in initiating state if it is already in Slave role. This is because it would lead to a Master connection in addition to the Slave role.
3. The device cannot have more than one Slave connections.
4. If the device is already operating in Connection or Initiating state, it cannot operate in Advertising state with advertising type that will lead to a Slave role connection.

8.11.1 Nonconnected States

In the Nonconnected states the link layer can be in any one of the following four states:

1. Standby State.
2. Advertising State.
3. Scanning State.
4. Initiating State.

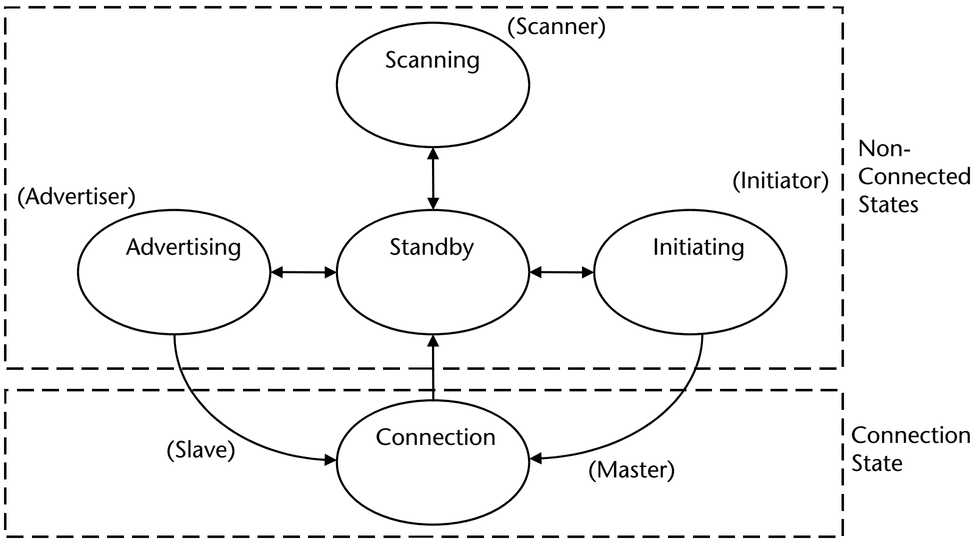


Figure 8.13 Link layer states.

These states are explained in detail below.

8.11.1.1 Standby State

This is the default state of the link layer. No packets are sent or received in this state. From this state, a device can enter advertising state, scanning state or initiating state. It cannot go into connection state directly from the standby state.

A device can enter into this state from any other state. In fact, to aid simplicity of the state machine and reduce the number of possible combinations of transitions from one state to another, the link layer state machine has been designed to only have the minimum needed transitions and this state is used as an intermediate state. For example, to go from scanning state to initiating state the link layer first goes from scanning state to standby and then from standby to initiating state.

8.11.1.2 Advertising State

In this state, the link layer transmits advertising PDUs in advertising events.

During an advertising event, the link layer transmits one or more advertising PDUs on each of the used advertising channels (The host may request the link layer to use either all or a subset of the three advertising channels—37, 38 and 39). The device in this state is known as Advertiser.

The advertising events are of following four types. The advertising PDUs associated with each of these events were shown in Table 8.4.

1. Connectable undirected event.
2. Connectable directed event.
3. Non connectable undirected event.
4. Scannable undirected event.

Connectable Undirected Event

The connectable undirected event is sent by an Advertiser when it wants another device to connect to it. It sends an advertising indication (ADV_IND) PDU on the advertising channel. The other device may also request for additional information before deciding to connect to the Advertiser.

Table 8.4 Advertising Event Types, PDUs Used and Acceptable Responses

		Acceptable Response PDU from Remote Device	
Advertising Event Type	*PDU Used*	*Scanner (SCAN_REQ)*	*Initiator (CONNECT_REQ)*
Connectable Undirected Event	ADV_IND	Yes	Yes (From any Initiator)
Connectable Directed Event	ADV_DIRECT_IND	No	Yes (Only from the addressed Initiator)
Nonconnectable Undirected Event	ADV_NONCONN_IND	No	No
Scannable Undirected Event	ADV_SCAN_IND	Yes	No

The receiver of the connectable undirected event can be either in the scanning state or initiating state.

- If it's in the scanning state, it may request for more information using SCAN_REQ PDU.
- If it's in the initiating state, it may send a connect request using CONNECT_REQ PDU.

The two scenarios are shown in Figure 8.14. A device may use the first one or the second one or first one followed by the second one.

The payload in a connectable undirected event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0-31 octets): This contains the advertising data from the Advertiser's host.

One possible example of this could be a thermometer that is placed in a building.

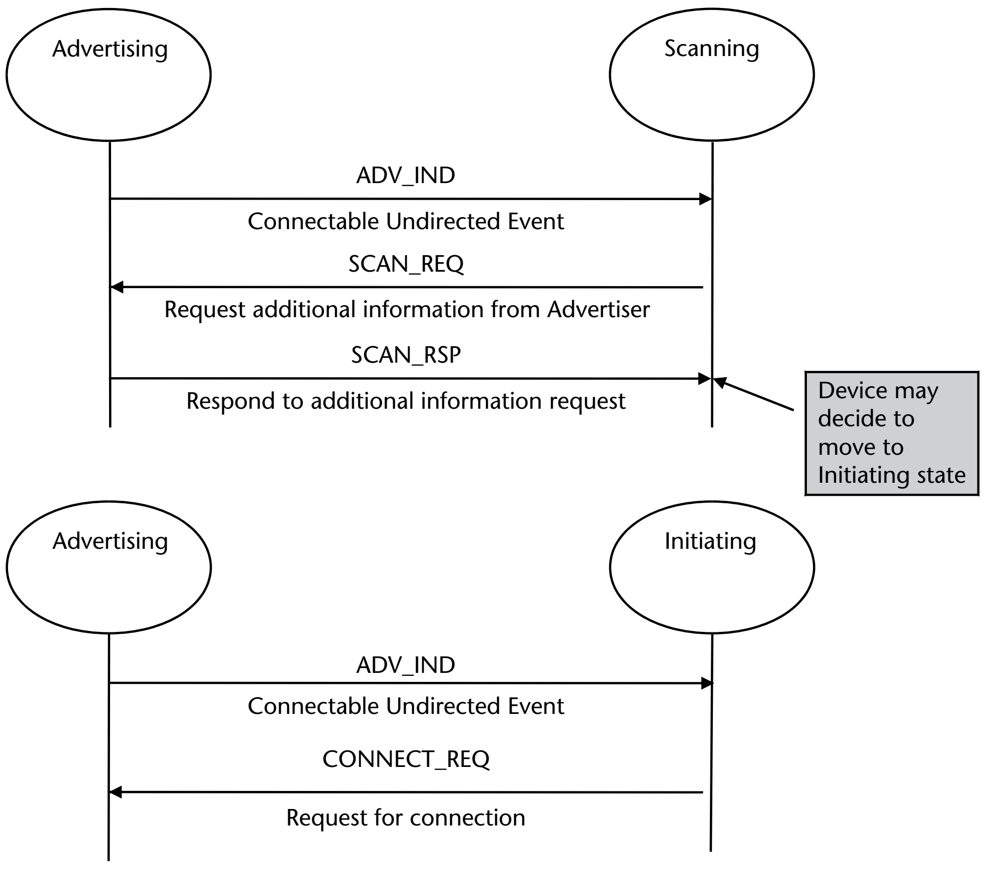

Advertising
Scanning
ADV_IND
Connectable Undirected Event
SCAN_REQ
Request additional information from Advertiser
SCAN_RSP
Respond to additional information request
Device may decide to move to Initiating state
Advertising
Initiating
ADV_IND
Connectable Undirected Event
CONNECT_REQ
Request for connection

Figure 8.14 Connectable undirected advertising event.

- The thermometer could keep on advertising—'I am a thermometer';
- Any mobile phone in the vicnity could query—'Do you display temperature in Fahrenheit?';
- The thermometer could say—'Yes';
- The mobile phone could then connect to the thermometer and get the temperature.

An example of air logs for ADV_IND was shown in Figure 8.5. The Advertiser sends an advertising PDU in frame #425, #426, etc. In response to the ADV_IND packet sent in Frame #427, the Scanner requests for additional information using the SCAN_REQ PDU.

Connectable Directed Event

The connectable directed event type is used when an Advertiser wants a particular device to connect to it. It sends a directed advertising indication (ADV_DIRECT_IND) PDU on the advertising channel. The other device may request to connect to the Advertiser on receiving this PDU.

The ADV_DIRECT_IND PDU contains the device address of both the Initiator and the Advertiser. So only the Initiator for which the address was contained in the PDU is allowed to make a connection.

The payload in a connectable directed event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- InitA (6 octets): Public address or random address of the Initiator. The type is indicated by RxAdd field (See Figure 8.9).

This is in contrast to connectable undirected event where any device in the scanning or initiating state could request for additional information or connect to the advertiser. Here the request for additional information is not permitted and only a particular device can initiate the connection. The sequence diagram for this is shown in Figure 8.15.

One possible example of this could be a pedometer placed in the jogger's shoe. The pedometer may need to send information to the person's mobile phone.

- The pedometer could advertise: 'I'm a pedometer. I want to send data to mobile phone A.'
- Mobile phone A could receive this request and connect to the pedometer and get the data and display to the person.

Nonconnectable Undirected Event

The nonconnectable undirected event type is used when an Advertiser wants to provide some information to all devices but does not want the devices to connect to it or ask for more information. It sends a nonconnectable advertising indication

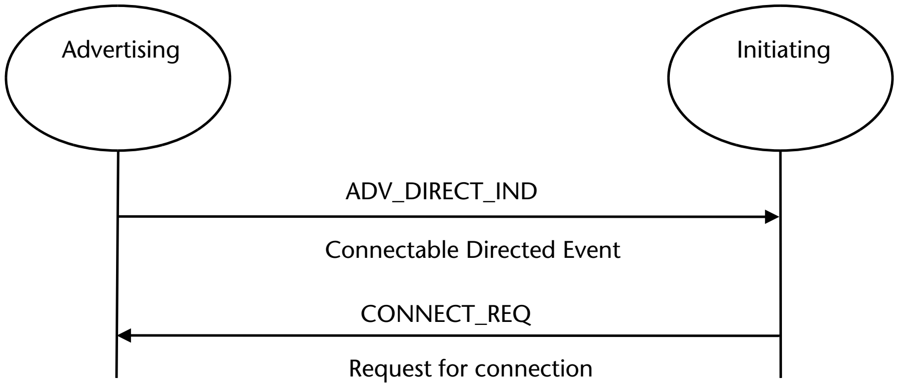


Figure 8.15 Connectable directed advertising event.

(ADV_NONCONN_IND) PDU on the advertising channel. The other device may only listen to this information.

This is in contrast to connectable undirected and connectable directed events because the receiver (which has to be in the scanning state) can just receive this information. It can neither connect nor ask for more information.

The payload in a non-connectable undirected event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.16.

One possible example of this could be a microwave oven.

- The microwave could advertise: 'I'm a microwave. The food is cooked. Please take it out.'
- The people in the house could receive this information on their mobile phone, television or set top box and take appropriate action.

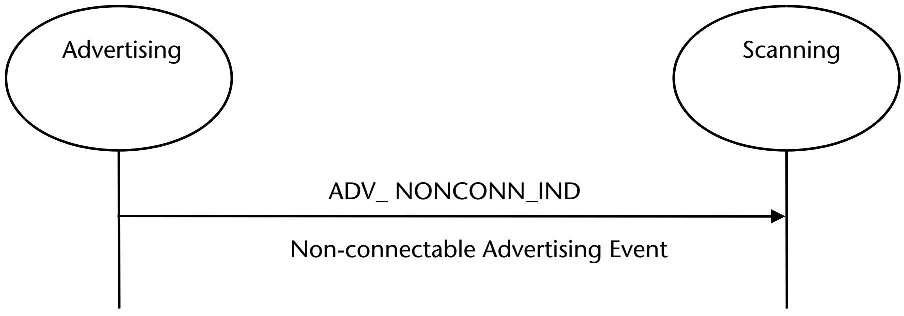


Figure 8.16 Nonconnectable undirected advertising event.

Another example could be an Advertiser at the airport:

- The Advertiser could advertise that flight ABC will take off from gate 123.
- Any person who is interested in the flight information could scan for that information.

Scannable Undirected Event

The scannable undirected event type is used when an Advertiser wants to allow a Scanner to request more information from it. It sends a scannable advertising indication (ADV_SCAN_IND) and the Scanner may request more information using the SCAN_REQ PDU.

This is slightly different from the Nonconnectable Undirected advertising event since in the nonconnectable advertising event, the Scanner cannot send any request back. Here the Scanner may send a scan request to get more information.

The payload in a scannable undirected event contains the following two fields:

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.17.

One possible example of this could be a remote control.

- The remote control could advertise: 'I'm a remote. A key has been pressed.'
- The Scanner could send a SCAN_REQ PDU to gather information about which key has been pressed.

A summary of the four advertising event types is provided in Table 8.4.

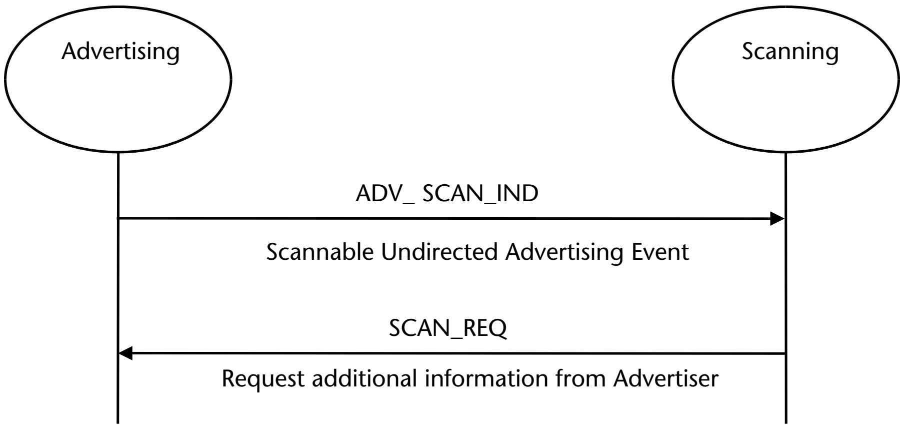

Advertising
Scanning
ADV_ SCAN_IND
Scannable Undirected Advertising Event
SCAN_REQ
Request additional information from Advertiser

Figure 8.17 Scannable undirected advertising event.

8.11.1.3 Scanning State

In this state, the link layer listens on the advertising channels (Advertising channels 37, 38, 39) for any PDUs from the Advertiser. The device in this state is known as Scanner.

The scanning events are of following two types:

- Passive Scanning.
- Active Scanning.

The advertising PDUs associated with each of these events were shown earlier in Table 8.2

Passive Scanning

In passive scanning, the link layer only receives the packets. It does not send back any packets. Once it receives the packets, it removes the duplicates and then sends the advertising reports to the host. The sequence diagram for this is shown in Figure 8.18.

Active Scanning

In active scanning, the link layer listens to the advertising PDUs and then depending on the advertising PDU type, it may request additional information from the Advertiser using the SCAN_REQ PDU.

The Scanner is permitted to send a SCAN_REQ only if the Advertiser used a connectable undirected event (ADV_IND PDU) or a scannable undirected event (ADV_NONCONN_IND PDU). These are the ones that are marked as Yes in the column for SCAN_REQ in Table 8.4.

The payload in a SCAN_REQ PDU contains the following two fields:

- ScanA (6 octets): Public address or random address of the Scanner. The type is indicated by TxAdd field (See Figure 8.9).
- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by RxAdd field (See Figure 8.9).

The payload in a SCAN_RSP PDU contains the following two fields:

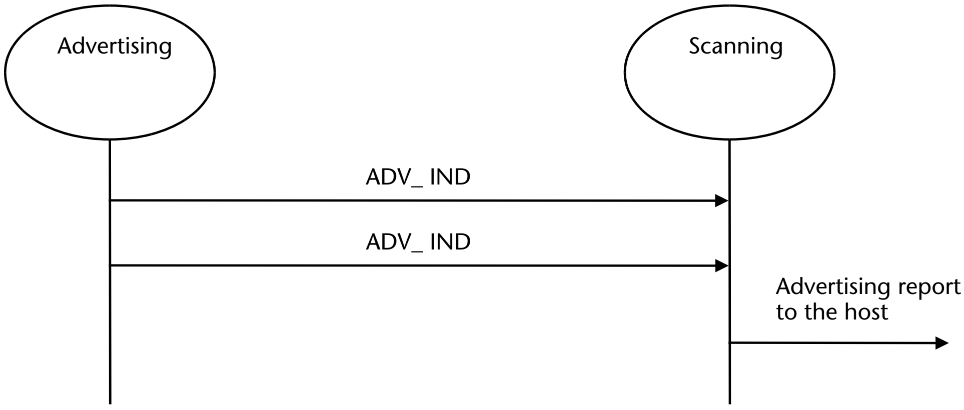


Figure 8.18 Passive scanning.

- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by TxAdd field (See Figure 8.9).
- AdvData (0–31 octets): This contains the advertising data from the Advertiser's host.

The sequence diagram for this is shown in Figure 8.19.

A practical example of active scanning was shown in Figure 8.5. At Frame #428, the Scanner sent a SCAN_REQ to get more information from the Advertiser.

8.11.1.4 Initiating State

In the initiating state, the link layer listens on the advertising channels (Advertising channels 37, 38, 39) for any PDUs from the Advertiser and if it's permitted, it sends a connection request to the Advertiser.

The Initiator is permitted to send a CONNECT_REQ only if the Advertiser used a connectable undirected event (ADV_IND PDU) or a connectable directed event (ADV_DIRECT_IND PDU). In the latter case, the address of the Initiator must match the address that was provided in the connectable directed event PDU. These are the scenarios that are marked as Yes in the column for CONNECT_REQ in Table 8.4. The sequence diagram for this is shown in Figure 8.20.

Figure 8.21 shows a sniffer capture of the connection initiation procedure. The following things may be observed:

- The Advertiser sent an advertising event (ADV_IND) in Frames #670, #671, and #672.
- The Initiator responded with a CONNECT_REQ in Frame #673 to create a connection.

8.11.2 Connection State

This state is entered when the Initiator sends a CONNECT_REQ PDU to the Advertiser. As shown in Figure 8.13, this state can be entered in two ways.

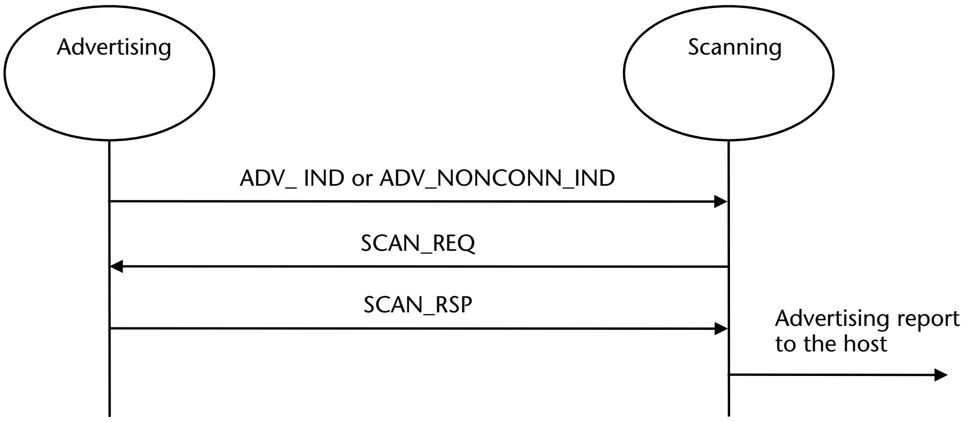


Figure 8.19 Active scanning.

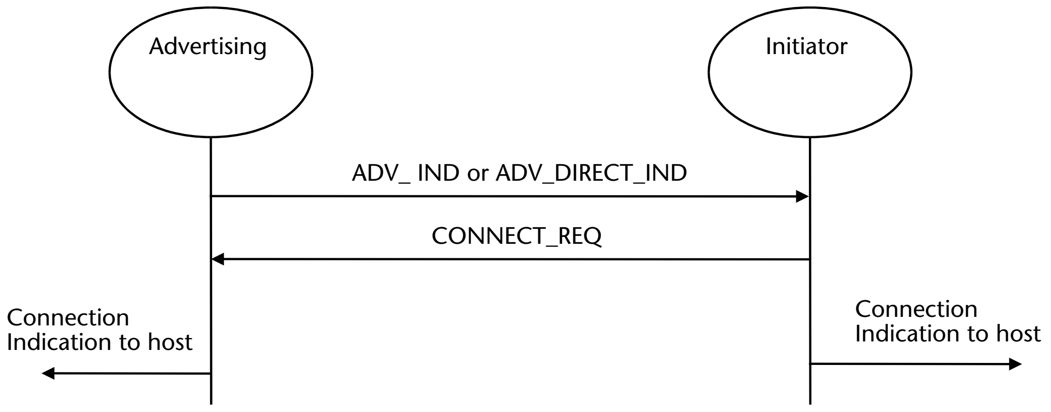
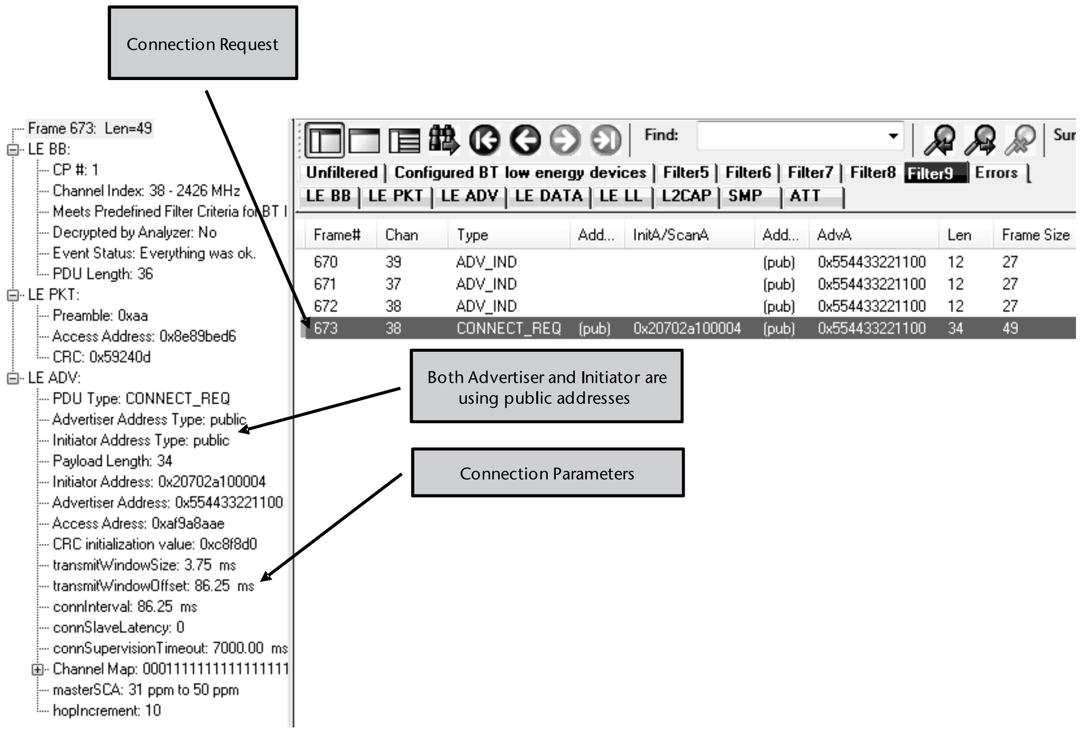

Advertising
Initiator
ADV_ IND or ADV_DIRECT_IND
CONNECT_REQ
Connection Indication to host
Connection Indication to host

Figure 8.20 Initiating connections.


Connection Request
Both Advertiser and Initiator are using public addresses
Connection Parameters
Find:
Sur
Unfiltered | Configured BT low energy devices | Filter5 | Filter6 | Filter7 | Filter8 | Filter9 | Errors
LE BB | LE PKT | LE ADV | LE DATA | LE LL | L2CAP | SMP | ATT

```
Frame 673: Len=49
LE BB:
    CP #: 1
    Channel Index: 38 - 2426 MHz
    Meets Predefined Filter Criteria for BT l
    Decrypted by Analyzer: No
    Event Status: Everything was ok.
    PDU Length: 36
LE PKT:
    Preamble: 0xaa
    Access Address: 0x8e89bed6
    CRC: 0x59240d
LE ADV:
    PDU Type: CONNECT_REQ
    Advertiser Address Type: public
    Initiator Address Type: public
    Payload Length: 34
    Initiator Address: 0x20702a100004
    Advertiser Address: 0x554433221100
    Access Adress: 0xaf9a8aae
    CRC initialization value: 0xc8f8d0
    transmitWindowSize: 3.75 ms
    transmitWindowOffset: 86.25 ms
    connInterval: 86.25 ms
    connSlaveLatency: 0
    connSupervisionTimeout: 7000.00 ms
    Channel Map: 0001111111111111111
    masterSCA: 31 ppm to 50 ppm
    hopIncrement: 10
```

Frame#	Chan	Type	Add...	InitA/ScanA	Add...	AdvA	Len	Frame Size
670	39	ADV_IND			(pub)	0x554433221100	12	27
671	37	ADV_IND			(pub)	0x554433221100	12	27
672	38	ADV_IND			(pub)	0x554433221100	12	27
673	38	CONNECT_REQ	(pub)	0x20702a100004	(pub)	0x554433221100	34	49

Figure 8.21 Example of initiating a connection.

- From a link layer in the initiating state. The Initiator becomes the Master of the connection.
- From a link layer in the advertising state. The Scanner becomes the Slave of the connection.

After entering the connection state, the connection is considered to be created (but not established). After the connection is created, once a data channel packet has been received from the peer device, the connection is considered to be established. There can only be one LE connection between two devices.

The payload in a CONNECT_REQ PDU contains the following three fields:

- InitA (6 octets): Public address or random address of the Initiator. The type is indicated by TxAdd field (See Figure 8.9).
- AdvA (6 octets): Public address or random address of the Advertiser. The type is indicated by RxAdd field (See Figure 8.9).
- LLData (22 octets): This contains various parameters related to the connection.
  - AA (4 octets): The Access Address for the Link Layer's connection.
  - CRCInit (3 octets): Initialization value for the CRC calculation. (Random value.)
  - WinSize (1 octet): Indicates transmit window size.
  - WinOffset (2 octets): Indicates transmit window offset.
  - Interval (2 octets): Connection Interval (connInterval).
  - Latency (2 octets): Connection Latency (connSlaveLatency).
  - Timeout (2 octets): Connection Supervision Timeout (connSupervisionTimeout).
  - ChM (5 octets): Channel bitmap showing used and unused channels.
  - Hop (5 bits): Hop increment to be used for frequency hopping algorithm.
  - SCA (3 bits): Indicate worst case Master's sleep clock accuracy.

Some of these parameters were already explained earlier in this chapter in the section related to Connection Events. The sequence diagram for this is shown in Figure 8.22. Once the connection is established, the Master and Slave can exchange data channel PDUs in connection events.

Figure 8.21 showed a sniffer capture of the connection initiation procedure. The parameters of the CONNECT_REQ PDU are shown on the left hand side. These include the following:

- InitA (6 octets): Public address of the Initator.
- AdvA (6 octets): Public address of the Advertiser.
- Access Address of the connection.
- LLData (22 octets) containing the following:
  - CRC Initialization value. Random value provided by Master to be used for CRC calculations.
  - transmitWindowSize = 3.75 ms.
  - transmitWindowsOffset = 86.25 ms.
  - connInterval = 86.25 ms.
  - connSlaveLatency = 0 (This means that the Slave has to listen for each connection event).

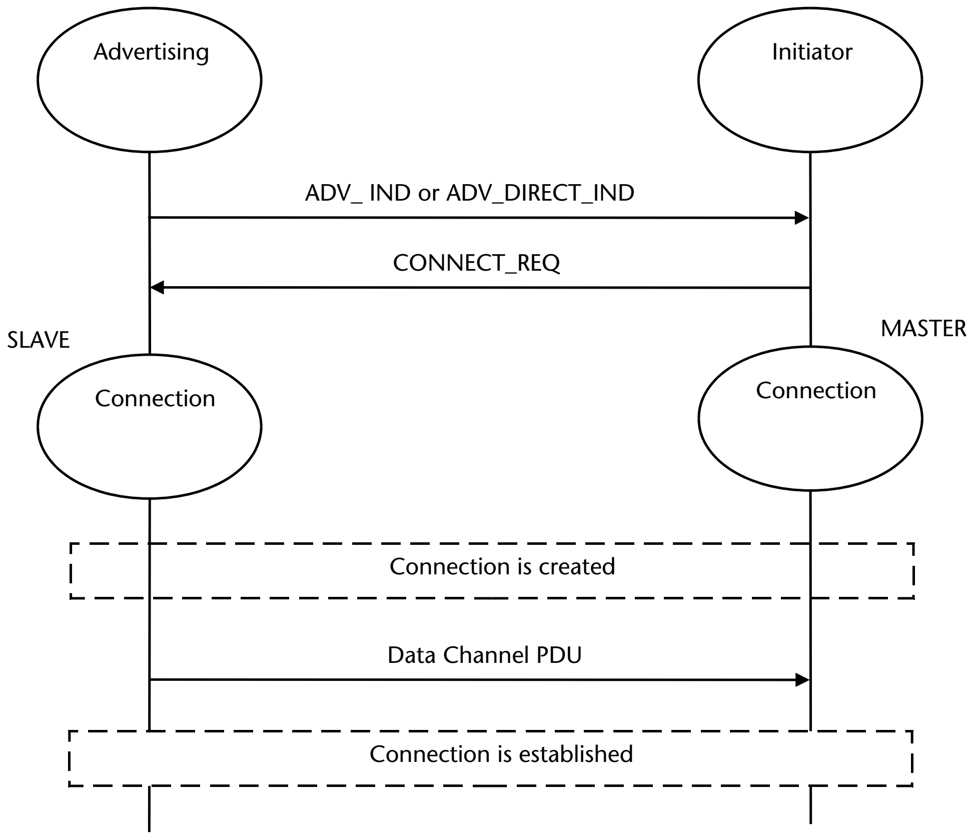


Figure 8.22 Connection state.

- connSupervisionTimeout = 7000 ms. (This means that if no packet is received for 7 seconds then the connection is considered to be lost).
- ChannelMap: Indicating which channels can be used for data transfer.
- HopIncrement: 10. This indicates the Hop Increment to be used in the algorithm to calculate the next hopping frequency.

8.12 Link Layer Control Procedures

The Link Layer Control Protocol (LLCP) is used to control and negotiate the connection between the two link layers. The summary of the link layer control procedures and the associated PDUs is shown in Table 8.5. Specifications 4.1 and 4.2 introduced certain new procedures and certain new PDUs within existing procedures. These are mentioned in Table 8.5. The PDUs are encapsulated within the link layer control PDUs that are part of the data channel PDUs. The format of the data channel PDUs was explained earlier in Figure 8.10.

These procedures can only be invoked sequentially. This means that at any given time only one link layer procedure is initiated. The next link layer procedure can only be initiated after the previous one either completes or has a timeout. The only exception is the termination procedure which can be initiated at any time.

Table 8.5 Link Layer Control Procedures and PDUs

Link Layer Procedure	*Link Layer Control PDU Name*	*Brief Purpose*
Connection Update Procedure	LL_CONNECTION_UPDATE_REQ	This procedure is used by the Master any time after entering the connection state to update the link layer parameters of a connection.
Channel Map Update Procedure	LL_CHANNEL_MAP_REQ	This procedure is used by the Master after entering the connection state to update the channel map to be used for frequency hopping.
Encryption Start Procedure	LL_ENC_REQ LL_ENC_RSP LL_START_ENC_REQ LL_START_ENC_RSP	This procedure is used by the Master to start encryption or to re-start encryption after a pause encryption procedure.
Encryption Pause Procedure	LL_PAUSE_ENC_REQ LL_PAUSE_ENC_RSP	This procedure is used if the Master wants to change the encryption key. A Pause procedure is followed by the Encryption Start procedure to change the link key.
Feature Exchange Procedure	LL_FEATURE_REQ LL_FEATURE_RSP	This procedure is used by the Master after a connection has been established to initiate exchange of feature set information.
Version Exchange Procedure	LL_VERSION_IND	This procedure is used by either the Master or the Slave after entering the connection state to exchange version information.
Termination Procedure	LL_TERMINATE_IND	This procedure is used in the connection state by either the Master or the state to terminate the connection.
Unused/ Unsupported PDU	LL_UNKNOWN_RSP	This PDU is sent as a response if an unused or unsupported PDU is received.
Rejection	LL_REJECT_IND	This PDU is sent if a request from the other side is rejected. For example the Slave sends this response to Master if the Master tries to enable encryption and the Slave does not support it.
Connection Parameters Request Procedure (Enhancement in 4.1)	LL_CONNECTION_PARAM_REQ LL_CONNECTION_PARAM_RSP	This procedure is used by the Master or the Slave to request the remote device to update the connection parameters.
LE Ping Procedure (Enhancement in 4.1)	LL_PING_REQ LL_PING_RSP	This procedure is used by the Master or the Slave to verify the presence of the remote link layer.
Data Length Update Procedure (Enhancement in 4.2)	LL_LENGTH_REQ LL_LENGTH_RSP	This procedure is used by the Master or the Slave to inform the remote link layer about changes in the values of data PDU length and PDU time.
Feature Exchange Procedure (Enhancement in 4.1)	LL_SLAVE_FEATURE_REQ	This is an additional PDU defined by specifications 4.1 to allow the Slave to request the features supported by the Master.
Rejection (Enhancement in 4.1)	LL_REJECT_IND_EXT	This is an additional PDU defined by specifications 4.1 to support extended reject indication to the remote side.

8.12.1 Connection Update Procedure

The connection update procedure is used to update the following link layer parameters of the connection:

- Connection Interval.
- Connection Slave Latency.
- Connection Supervision Timeout.

This procedure can only be initiated by the Master after entering the connection state.The sequence diagram for this procedure is shown in Figure 8.23.

8.12.2 Channel Map Update Procedure

The channel map procedure is used to update the channel map of the connection. The channel map contains two parameters:

- Channel Map: The bitmap indicating which channels are enabled.
- Hop Increment: This indicates the number of channels to hop for each subsequent hop.

This procedure allows the Master to disable frequency hopping on channels which potentially have interference thereby reducing the number of retransmissions that would have been required if the packets were transmitted on those channels. This is the key procedure that provides support for adaptive frequency hopping (AFH).

This procedure can only be initiated by the Master after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.24.

8.12.3 Encryption Procedure

8.12.3.1 Encryption Start Procedure

The encryption start procedure is used by the link layer to enable encryption of packets. It is initiated by the host of the Master by sending a request to the link layer to start encryption. The host of the link layer also provides the Long_Term_Key (LTK). The Long_Term_Key is a 128-bit key used to generate the session key to be

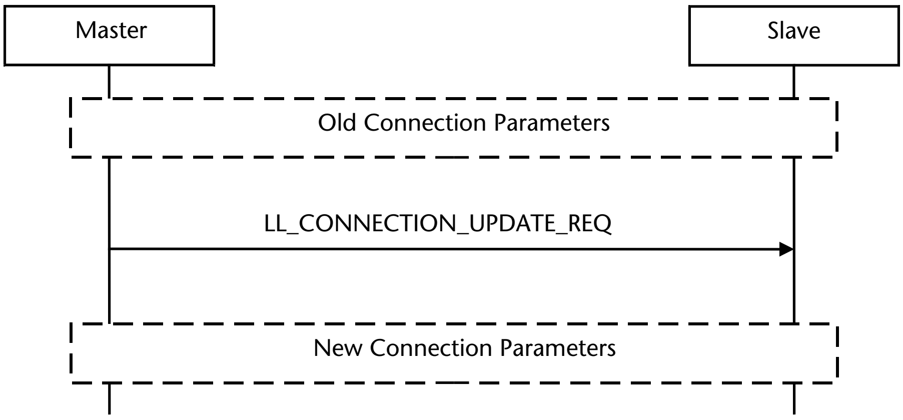


Figure 8.23 Connection update procedure.

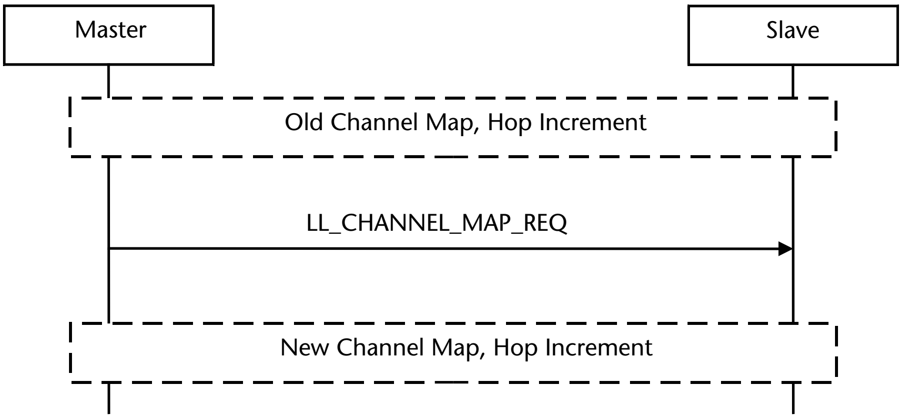

Master
Slave
Old Channel Map, Hop Increment
LL_CHANNEL_MAP_REQ
New Channel Map, Hop Increment

Figure 8.24 Channel map update procedure.

used for the encrypted connection. If the connection is not already encrypted, then the encryption start procedure is used.

If the connection is already encrypted, then the encryption pause procedure followed by encryption start procedure is used. The encryption pause procedure will be explained in the next section. The sequence diagram for this procedure is shown in Figure 8.25.

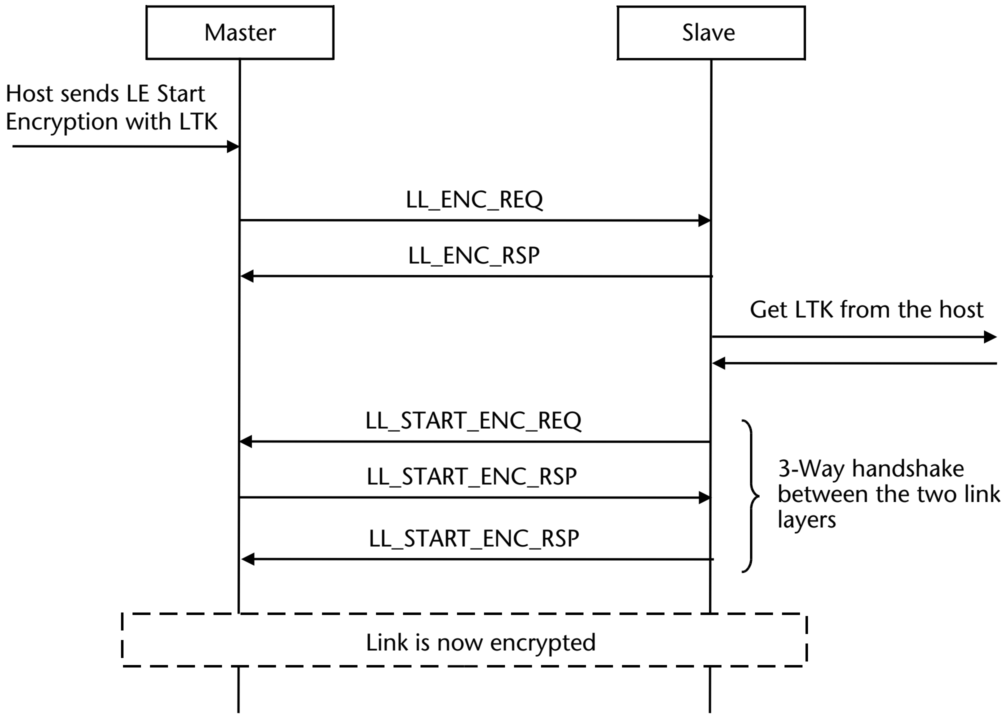


Figure 8.25 Encryption start procedure.

8.12.3.2 Encryption Pause Procedure and Encryption Restart Procedure

This procedure is used by the link layer if the link is already encrypted but a new encryption key is to be used without disconnecting the link. So, the link layer pauses encryption and then follows the same procedure as described in the previous section with the new encryption key.

Encryption restart procedure is very useful when the level of encryption has to be increased or decreased dynamically. So a link can be established with one level of security, and later, if the security needs to be increased it can be done using this procedure.

One example of this could be when a connection was established with a lower level of security but the security level needs to be increased (for example) to transmit some sensitive data. In that case, the encryption pause procedure is used followed by changing the link key and finally the encryption restart procedure.

The sequence diagram for this procedure along with the procedure to restart encryption with the new link key is shown in Figure 8.26.

8.12.4 Feature Exchange Procedure

The feature exchange procedure is used to exchange information about the current supported feature set.

Feature set is a bitmap that provides the feature capabilities of the Master or Slave. As per specifications 4.0, the feature set contained only information on whether encryption was supported on not. Other fields of this bitmap were reserved for future use. This procedure should only be initiated by the Master after

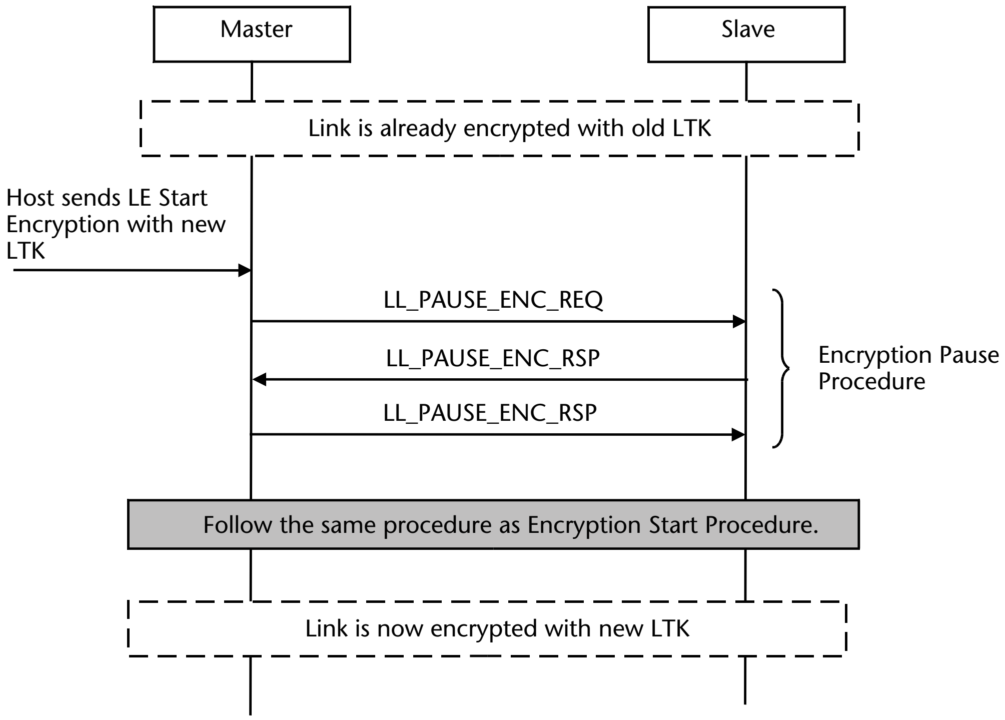

Master
Slave
Link is already encrypted with old LTK
Host sends LE Start Encryption with new LTK
LL_PAUSE_ENC_REQ
LL_PAUSE_ENC_RSP
LL_PAUSE_ENC_RSP
Encryption Pause Procedure
Follow the same procedure as Encryption Start Procedure.
Link is now encrypted with new LTK

Figure 8.26 Encryption pause procedure and encryption restart procedure.

entering the connection state. The sequence diagram for this procedure is shown in Figure 8.27.

Specifications 4.1 and 4.2 made several enhancements to the feature set and feature exchange procedure.

The information about the supported features can be exchanged at two levels:

- Information sent from a Controller to the Host: In this case, the features that are not supported are indicated by setting the corresponding bit to 0 in the FeatureSet field.
- Information sent from a Controller to the Peer Controller: In this case, if the Controller allows a feature to be used, it sets the corresponding bit to 1 in the FeatureSet field.

The following features were added to the Feature Set in Specifications 4.1:

- Connection parameters request procedure;
- Extended reject indication;
- Slave-initiated features exchange;
- LE ping.

Specifications 4.1 and above allow both the Master and the Slave to initiate a feature exchange procedure. This is shown in Figure 8.27.

The following features were added to the Feature Set in Specifications 4.2 and above:

- LE data packet length extension;

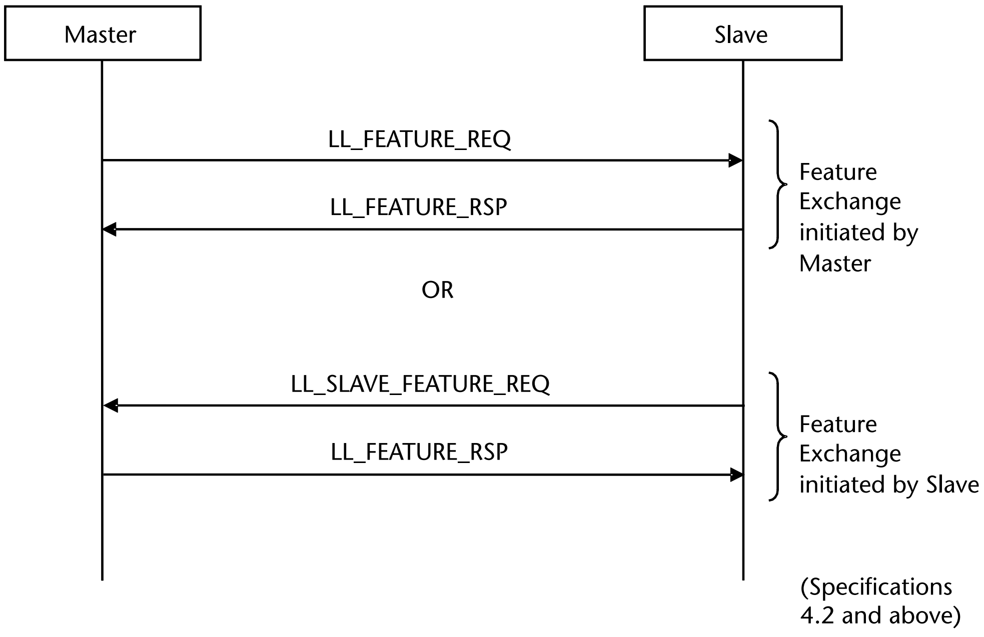

Master
Slave
LL_FEATURE_REQ
LL_FEATURE_RSP
Feature Exchange initiated by Master
OR
LL_SLAVE_FEATURE_REQ
LL_FEATURE_RSP
Feature Exchange initiated by Slave
(Specifications 4.2 and above)

Figure 8.27 Feature exchange procedure.

- LL privacy;
- Extended scanner filter policies.

The details on bit positions, their corresponding features, and whether these bits are valid from Controller to Host or Controller to Controller are show in Table 8.6.

8.12.5 Version Exchange Procedure

The version exchange procedure is used by either the Master or the Slave after entering the connection state to exchange version information. The version information consists of:

- Company ID;
- Link Layer Version;
- Sub Version Number.

The remote side responds with its own version information using the same PDU (LL_VERSION_IND) if it has not sent it in the past for the same connection. This procedure can be initiated by either the Master or the Slave after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.28.

8.12.6 Termination Procedure

The termination procedure is used by either the Master or the Slave to terminate the current connection. This procedure can be used after entering the connection state. The sequence diagram for this procedure is shown in Figure 8.29.

Table 8.6 Bit Mapping for Feature Set

Bit position	*Link Layer Feature*	*Specifications Version in which this was introduced*	*Valid from Controller to Host*	*Valid from Controller to peer Controller*
0	LE Encryption	4.0	Y	Y
1	Connection Parameters Request Procedure	4.1	Y	Y
2	Extended Reject Indication	4.1	Y	Y
3	Slave-initiated Features Exchange	4.1	Y	Y
4	LE Ping	4.1	Y	N
5	LE Data Packet Length Extension	4.2	Y	Y
6	LL Privacy	4.2	Y	N
7	Extended Scanner Filter Policies	4.2	Y	N
8–63	RFU			

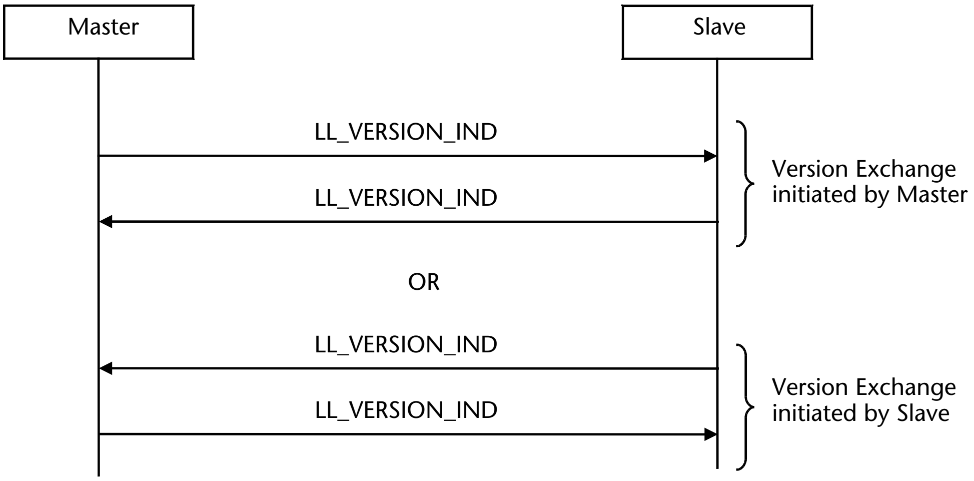

Master
Slave
LL_VERSION_IND
LL_VERSION_IND
Version Exchange initiated by Master
OR
LL_VERSION_IND
LL_VERSION_IND
Version Exchange initiated by Slave

Figure 8.28 Version exchange procedure.

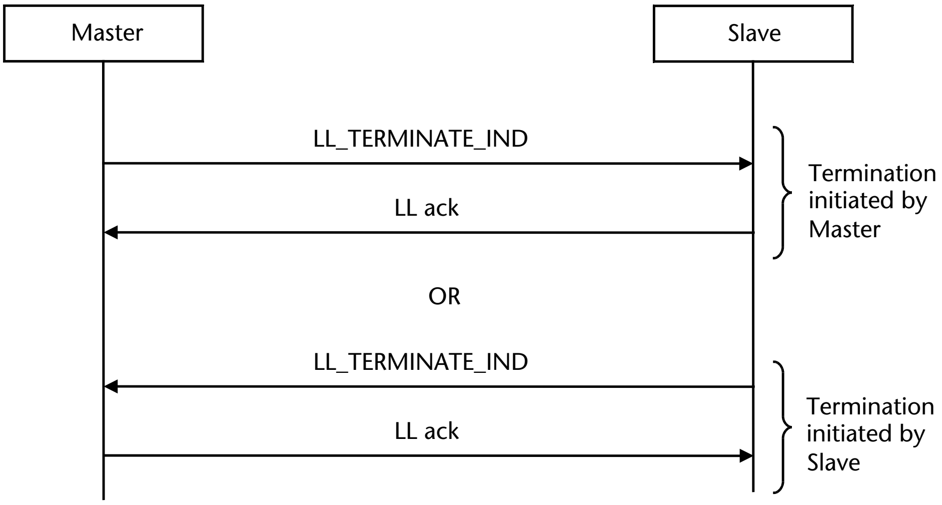

Master
Slave
LL_TERMINATE_IND
LL ack
Termination initiated by Master
OR
LL_TERMINATE_IND
LL ack
Termination initiated by Slave

Figure 8.29 Termination procedure.

The procedures mentioned above are supported by specifications 4.0 and above. Besides these, specifications 4.1 introduced two new procedures (the connection parameters request procedure and the LE ping procedure). Specifications 4.2 introduced one additional procedure (the data length update procedure). These procedures are described below.

8.12.7 Connection Parameters Request Procedure

The connection parameters request procedure is used by either the Master or the Slave to request the remote device in order to update the connection parameters during the connection state. The connection parameters include:

- connInterval;