

order to distribute the starting states more uniformly. The operation is defined in (EQ 30).

When the encryption key has been created the LFSRs are loaded with their initial values. Then, 200 stream cipher bits are created by operating the generator. Of these bits, the last 128 are fed back into the key stream generator as an initial value of the four LFSRs. The values of c_t and c_{t-1} are kept. From this point on, when clocked the generator produces the encryption (decryption) sequence which is bitwise XORed to the transmitted (received) payload data.

In the following, we will denote octet i of a binary sequence X by the notation $X[i]$. We define bit 0 of X to be the LSB. Then, the LSB of $X[i]$ corresponds to bit $8i$ of the sequence X , the MSB of $X[i]$ is bit $8i + 7$ of X . For instance, bit 24 of the Bluetooth address is the LSB of $ADR[3]$.

The details of the initialization are as follows:

1. Create the encryption key to use from the 128-bit secret key K_C and the 128-bit publicly known EN_RAND . Let $L, 1 \leq L \leq 16$, be the effective key length in number of octets. The resulting encryption key will be denoted K'_C :

$$K'_C(x) = g_2^{(L)}(x)(K_C(x) \bmod g_1^{(L)}(x)), \quad (\text{EQ 30})$$

where $\deg(g_1^{(L)}(x)) = 8L$ and $\deg(g_2^{(L)}(x)) \leq 128 - 8L$. The polynomials are defined in Table 14.6.

2. Shift in the 3 inputs K'_C , the Bluetooth address, the clock, and the six-bit constant 111001 into the LFSRs. In total 208 bits are shifted in.
 - a) Open all switches shown in Figure 14.8 on page 168;
 - b) Arrange inputs bits as shown in Figure 14.8; Set the content of all shift register elements to zero. Set $t = 0$.
 - c) Start shifting bits into the LFSRs. The rightmost bit at each level of Figure 14.8 is the first bit to enter the corresponding LFSR.
 - d) When the first input bit at level i reaches the rightmost position of $LFSR_i$, close the switch of this LFSR.
 - e) At $t = 39$ (when the switch of $LFSR_4$ is closed), reset both blend registers $c_{39} = c_{39-1} = 0$; Up to this point, the content of c_t and c_{t-1} has been of no concern. However, from this moment forward their content will be used in computing the output sequence.
 - f) From now on output symbols are generated. The remaining input bits are continuously shifted into their corresponding shift register. When the last bit has been shifted in, the shift register is clocked with input = 0;

Note: When finished, $LFSR_1$ has effectively clocked 30 times with feedback closed, $LFSR_2$ has clocked 24 times, $LFSR_3$ has

clocked 22 times, and LFSR₄ has effectively clocked 16 times with feedback closed.

3. To mix initial data, continue to clock until 200 symbols have been produced with all switches closed ($t = 239$);
4. Keep blend registers c_t and c_{t-1} , make a parallel load of the last 128 generated bits into the LFSRs according to Figure 14.9 at $t = 240$;

After the parallel load in item 4, the blend register contents will be updated for each subsequent clock.

L	deg	$g_1^{(L)}$	deg	$g_2^{(L)}$
1	[8]	00000000 00000000 00000000 0000011d	[119]	00e275a0 abd218d4 cf928b9b bf6cb08f
2	[16]	00000000 00000000 00000000 0001003f	[112]	0001e3f6 3d7659b3 7f18c258 cff6efef
3	[24]	00000000 00000000 00000000 010000db	[104]	000001be f66c6c3a b1030a5a 1919808b
4	[32]	00000000 00000000 00000001 000000af	[96]	00000001 6ab89969 de17467f d3736ad9
5	[40]	00000000 00000000 00000100 00000039	[88]	00000000 01630632 91da50ec 55715247
6	[48]	00000000 00000000 00010000 00000291	[77]	00000000 00002c93 52aa6cc0 54468311
7	[56]	00000000 00000000 01000000 00000095	[71]	00000000 000000b3 f7fffee2 79f3a073
8	[64]	00000000 00000001 00000000 0000001b	[63]	00000000 00000000 a1ab815b c7ec8025
9	[72]	00000000 00000100 00000000 00000609	[49]	00000000 00000000 0002c980 11d8b04d
10	[80]	00000000 00010000 00000000 00000215	[42]	00000000 00000000 0000058e 24f9a4bb
11	[88]	00000000 01000000 00000000 0000013b	[35]	00000000 00000000 0000000c a76024d7
12	[96]	00000001 00000000 00000000 000000dd	[28]	00000000 00000000 00000000 1c9c26b9
13	[104]	00000100 00000000 00000000 0000049d	[21]	00000000 00000000 00000000 0026d9e3
14	[112]	00010000 00000000 00000000 0000014f	[14]	00000000 00000000 00000000 00004377
15	[120]	01000000 00000000 00000000 000000e7	[7]	00000000 00000000 00000000 00000089
16	[128]	1 00000000 00000000 00000000 00000000	[0]	00000000 00000000 00000000 00000001

Table 14.6: Polynomials used when creating K_C . All polynomials are in hexadecimal notation. The LSB is in the rightmost position.

In Figure 14.8, all bits are shifted into the LFSRs, starting with the least significant bit (LSB). For instance, from the third octet of the address, ADR[2], first ADR₁₆ is entered, followed by ADR₁₇, etc. Furthermore, CL₀ corresponds to CLK₁,..., CL₂₅ corresponds to CLK₂₆.

Note that the output symbols x_p^i , $i = 1, \dots, 4$ are taken from the positions 24, 24, 32, and 32 for LFSR₁, LFSR₂, LFSR₃, and LFSR₄, respectively (counting the leftmost position as number 1).

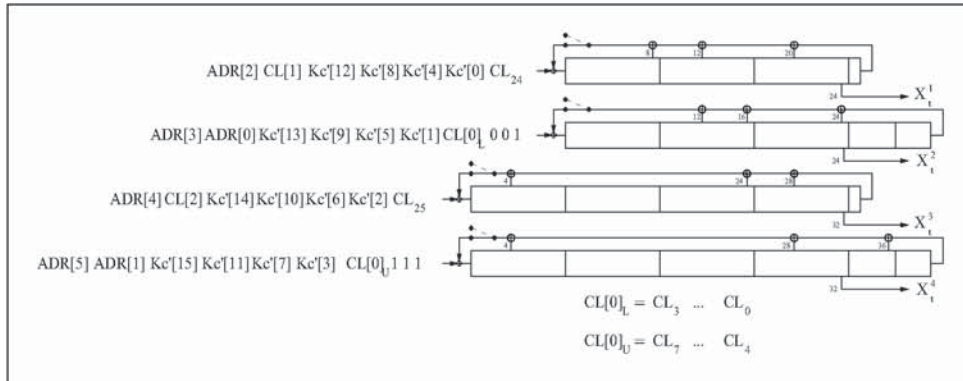


Figure 14.8: Arranging the input to the LFSRs.

In Figure 14.9, the 128 binary output symbols Z_0, \dots, Z_{127} are arranged in octets denoted $Z[0], \dots, Z[15]$. The LSB of $Z[0]$ corresponds to the first of these symbols, the MSB of $Z[15]$ is the latest output from the generator. These bits shall be loaded into the LFSRs according to the figure. It is a parallel load and no update of the blend registers is done. The first output symbol is generated at the same time. The octets are written into the registers with the LSB in the left-most position (i.e. the opposite of before). For example, Z_{24} is loaded into position 1 of LFSR₄.

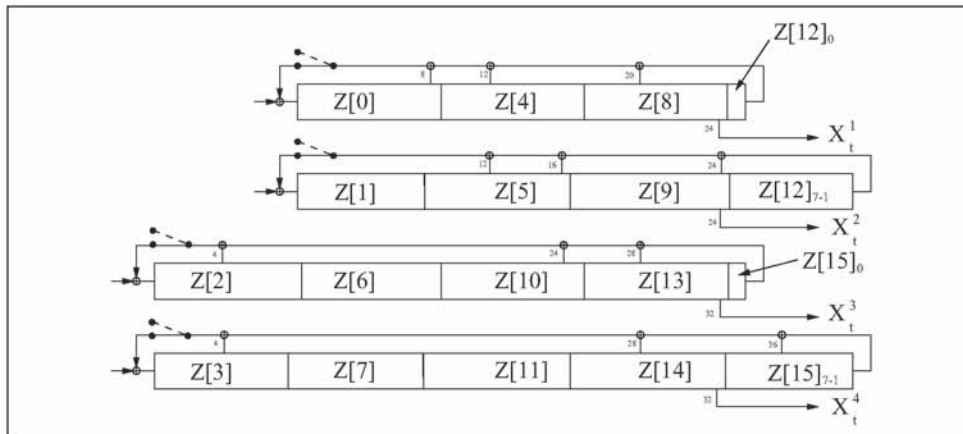


Figure 14.9: Distribution of the 128 last generated output symbols within the LFSRs.

14.3.6 Key stream sequence

When the initialization is finished, the output from the summation combiner is used for encryption/decryption. The first bit to use is the one produced at the parallel load, i.e. at $t = 240$. The circuit is run for the entire length of the current payload. Then, before the reverse direction is started, the entire initialization process is repeated with updated values on the input parameters.

Sample data of the encryption output sequence can be found in "Appendix IV" on page 899, Encryption Sample Data. A necessary, but not sufficient, condition for all Bluetooth-compliant implementations is to produce these encryption streams for identical initialization values.

14.4 AUTHENTICATION

The entity authentication used in Bluetooth uses a challenge-response scheme in which a claimant's knowledge of a secret key is checked through a 2-move protocol using symmetric secret keys. The latter implies that a correct claimant/verifier pair share the same secret key, for example K . In the challenge-response scheme the verifier challenges the claimant to authenticate a random input (the challenge), denoted by AU_RAND_A , with an authentication code, denoted by E_1 , and return the result $SRES$ to the verifier, see Figure 14.10 on page 169. This figure shows also that in Bluetooth the input to E_1 consists of the tuple AU_RAND_A and the Bluetooth device address (BD_ADDR) of the claimant. The use of this address prevents a simple reflection attack¹. The secret K shared by units A and B is the current link key.

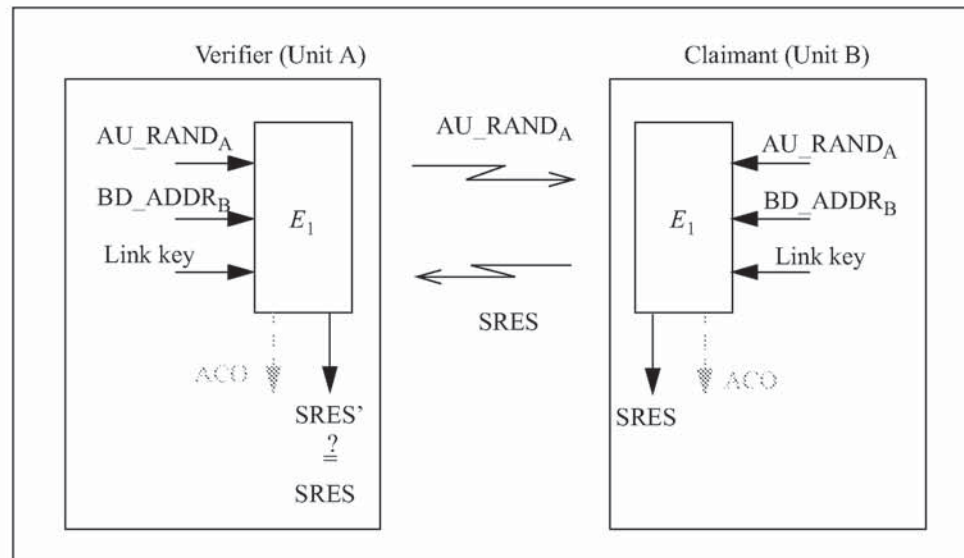


Figure 14.10: Challenge-response for the Bluetooth.

The challenge-response scheme for symmetric keys used in the Bluetooth is depicted in Figure 14.11 on page 170.

1. The reflection attack actually forms no threat in Bluetooth because all service requests are dealt with on a FIFO bases. When pre-emption is introduced, this attack is potentially dangerous.

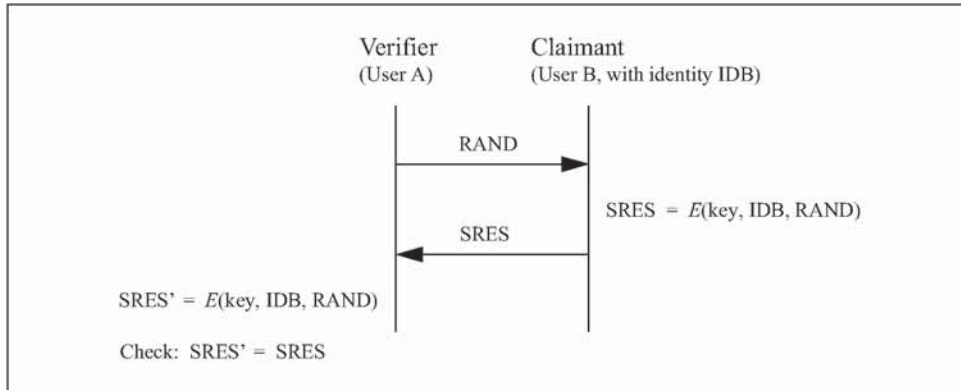


Figure 14.11: Challenge-response for symmetric key systems.

In the Bluetooth, the verifier is not necessarily the master. The application indicates who has to be authenticated by whom. Certain applications only require a one-way authentication. However, in some peer-to-peer communications, one might prefer a mutual authentication in which each unit is subsequently the challenger (verifier) in two authentication procedures. The LM coordinates the indicated authentication preferences by the application to determine in which direction(s) the authentication(s) has to take place. For mutual authentication with the units of Figure 14.10 on page 169, after unit A has successfully authenticated unit B, unit B could authenticate unit A by sending a AU_RAND_B (different from the AU_RAND_A that unit A issued) to unit A, and deriving the SRES and SRES' from the new AU_RAND_B , the address of unit A, and the link key K_{AB} .

If an authentication is successful the value of ACO as produced by E_1 should be retained.

14.4.1 Repeated attempts

When the authentication attempt fails, a certain waiting interval must pass before a new authentication attempt can be made. For each subsequent authentication failure with the same Bluetooth address, the waiting interval shall be increased exponentially. That is, after each failure, the waiting interval before a new attempt can be made, for example, twice as long as the waiting interval prior to the previous attempt¹. The waiting interval shall be limited to a maximum. The maximum waiting interval depends on the implementation. The waiting time shall exponentially decrease to a minimum when no new failed attempts are being made during a certain time period. This procedure prevents an intruder to repeat the authentication procedure with a large number of different keys.

1. An other appropriate value larger than 1 may be used.

To make the system somewhat less vulnerable to denial-of-service attacks, the Bluetooth units should keep a list of individual waiting intervals for each unit it has established contact with. Clearly, the size of this list must be restricted only to contain the N units with which the most recent contact has been made. The number N can vary for different units depending on available memory size and user environment.

14.5 THE AUTHENTICATION AND KEY-GENERATING FUNCTIONS

This section describes the algorithmic means for supporting the Bluetooth security requirements on authentication and key generation.

14.5.1 The authentication function E_1

The authentication function E_1 proposed for the Bluetooth is a computationally secure authentication code, or often called a MAC. E_1 uses the encryption function called SAFER+. The algorithm is an enhanced version¹ of an existing 64-bit block cipher SAFER-SK128, and it is freely available. In the sequel the block cipher will be denoted as the function A_r , which maps under a 128-bit key, a 128-bit input to a 128-bit output, i.e.

$$A_r: \{0, 1\}^{128} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128} \quad (\text{EQ 31})$$

$$(k \times x) \mapsto t.$$

The details of A_r are given in the next section. The function E_1 is constructed using A_r as follows

$$E_1: \{0, 1\}^{128} \times \{0, 1\}^{128} \times \{0, 1\}^{48} \rightarrow \{0, 1\}^{32} \times \{0, 1\}^{96} \quad (\text{EQ 32})$$

$$(K, \text{RAND}, \text{address}) \mapsto (\text{SRES}, \text{ACO}),$$

where $\text{SRES} = \text{Hash}(K, \text{RAND}, \text{address}, 6)[0, \dots, 3]$, where Hash is a keyed hash function defined as²,

$$\text{Hash}: \{0, 1\}^{128} \times \{0, 1\}^{128} \times \{0, 1\}^{8 \times L} \times \{6, 12\} \rightarrow \{0, 1\}^{128} \quad (\text{EQ 33})$$

$$(K, I_1, I_2, L) \mapsto A_r([\tilde{K}], [E(I_2, L) +_{16} (A_r(K, I_1) \oplus_{16} I_1)]),$$

and where

1. It is presently one of the contenders for the Advanced Encryption Standard (AES) submitted by Cylink, Corp, Sunnyvale, USA
2. The operator $+_{16}$ denotes bitwise addition mod 256 of the 16 octets, and the operator \oplus_{16} denotes bitwise XORing of the 16 octets.

$$E: \{0, 1\}^{8 \times L} \times \{6, 12\} \rightarrow \{0, 1\}^{8 \times 16} \quad (\text{EQ 34})$$

$$(X[0, \dots, L-1], L) \mapsto (X[i(\bmod L)] \text{ for } i = 0 \dots 15),$$

is an expansion of the L octet word X into a 128-bit word. Thus we see that we have to evaluate the function A_r twice for each evaluation of E_1 . The key \tilde{K} for the second use of A_r (actually A'_r) is offseted from K as follows¹

$$\begin{aligned} K[0] &= (K[0] + 233) \bmod 256, & K[1] &= K[1] \oplus 229, \\ \tilde{K}[2] &= (K[2] + 223) \bmod 256, & K[3] &= K[3] \oplus 193, \\ \tilde{K}[4] &= (K[4] + 179) \bmod 256, & K[5] &= K[5] \oplus 167, \\ \tilde{K}[6] &= (K[6] + 149) \bmod 256, & K[7] &= K[7] \oplus 131, \\ \tilde{K}[8] &= K[8] \oplus 233, & \tilde{K}[9] &= (K[9] + 229) \bmod 256, \\ \tilde{K}[10] &= K[10] \oplus 223, & \tilde{K}[11] &= (K[11] + 193) \bmod 256, \\ \tilde{K}[12] &= K[12] \oplus 179, & \tilde{K}[13] &= (K[13] + 167) \bmod 256, \\ \tilde{K}[14] &= K[14] \oplus 149, & \tilde{K}[15] &= (K[15] + 131) \bmod 256. \end{aligned} \quad (\text{EQ 35})$$

A data flowchart of the computation of E_1 is depicted in Figure 14.12 on page 173. E_1 is also used to deliver the parameter ACO (Authenticated Ciphering Offset) that is used in the generation of the ciphering key by E_3 , see equations (EQ 23) and (EQ 43). The value of ACO is formed by the octets 4 through 15 of the output of the hash function defined in (EQ 33), i.e.

$$\text{ACO} = \text{Hash}(K, \text{RAND}, \text{address}, 6)[4, \dots, 15]. \quad (\text{EQ 36})$$

1. The constants are the first largest primes below 257 for which 10 is a primitive root.

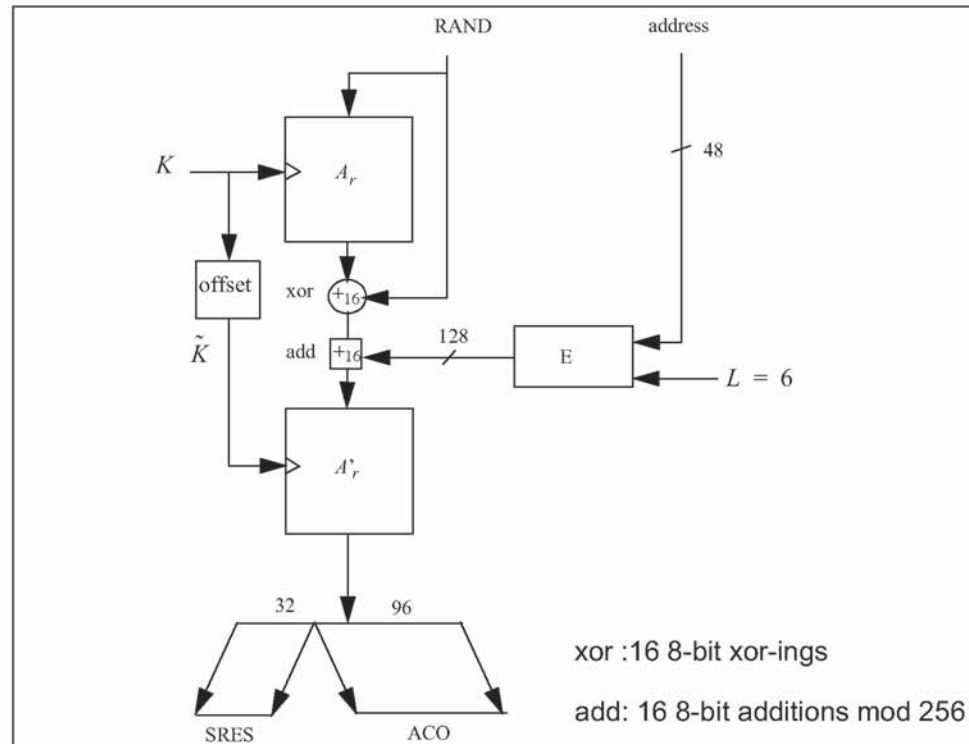


Figure 14.12: Flow of data for the computation of E_1 .

14.5.2 The functions A_r and A'_r

The function A_r is identical to SAFER+. It consists of a set of 8 layers, (each layer is called a round) and a parallel mechanism for generating the sub keys $K_p[j]$, $p = 1, 2, \dots, 17$, the so-called round keys to be used in each round. The function will produce a 128-bit result from a 128-bit “random” input string and a 128-bit “key”. Besides the function A_r , a slightly modified version referred to as A'_r is used in which the input of round 1 is added to the input of the 3rd round. This is done to make the modified version non-invertible and prevents the use of A'_r (especially in E_{2x}) as an encryption function. See Figure 14.13 on page 174 for details.

14.5.2.1 The round computations

The computations in each round are a composition of encryption with a round key, substitution, encryption with the next round key, and, finally, a Pseudo Hadamard Transform (PHT). The computations in a round are shown in Figure 14.13 on page 174. The sub keys for round r , $r = 1, 2, \dots, 8$ are denoted

$K_{2r-1}[j], K_{2r}[j], j = 0, 1, \dots, 15$. After the last round $k_{17}[j]$ is applied in a similar fashion as all previous odd numbered keys.

14.5.2.2 The substitution boxes “e” and “l”

In Figure 14.13 on page 174 two boxes occur, marked “e” and “l”. These boxes implement the same substitutions as used in SAFER+; i.e. they implement

$$\begin{aligned}
 e, l & : \{0, \dots, 255\} \rightarrow \{0, \dots, 255\}, \\
 e & : i \mapsto (45^i \pmod{257}) \pmod{256}, \\
 l & : i \mapsto j \text{ s.t. } i = e(j).
 \end{aligned}$$

Their role, as in the SAFER+ algorithm, is to introduce non-linearity.

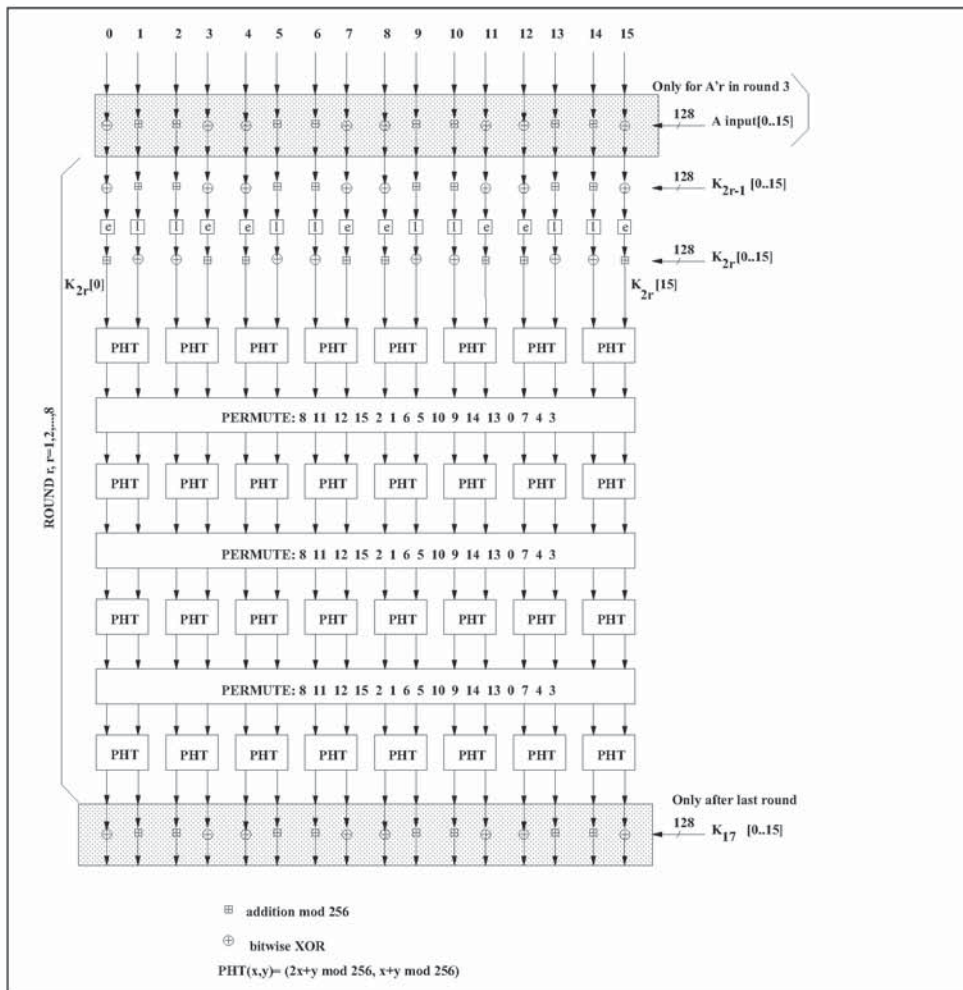


Figure 14.13: One round in A_r and A'_r . The permutation boxes show how input byte indices are mapped onto output byte indices. Thus, position 0 (leftmost) is mapped on position 8, position 1 is mapped on position 11, et cetera.

14.5.2.3 Key scheduling

In each round, 2 batches of 16 octet-wide keys are needed. These so-called round keys are derived as specified by the key scheduling in SAFER+. Figure 14.14 on page 175 gives an overview of how the round keys $K_p[j]$ are determined. The bias vectors B_2, B_3, \dots, B_{17} are computed according to following equation:

$$B_p[i] = ((45^{(45^{17p+i+1} \bmod 257)} \bmod 257) \bmod 256), \text{ for } i = 0, \dots, 15. \quad (\text{EQ 37})$$

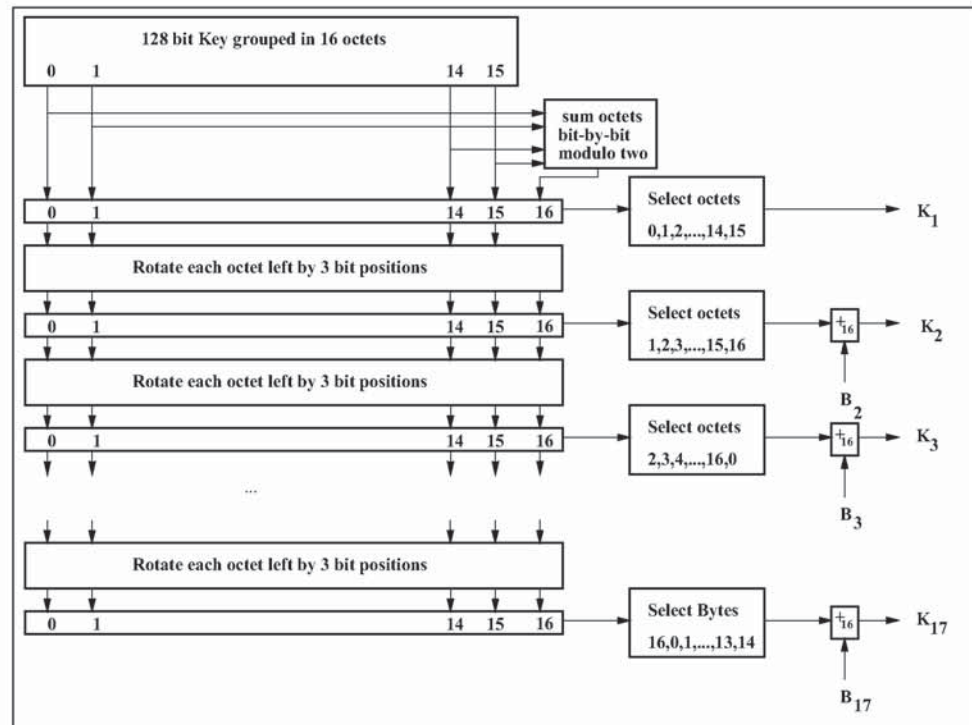


Figure 14.14: Key scheduling in A_p .

14.5.3 E_2 -Key generation function for authentication

The key used for authentication is derived through a procedure that is shown in Figure 14.15 on page 177. The figure shows two different modes of operation for the algorithm. In the first mode, the function E_2 should produce on input of a 128-bit RAND value and a 48-bit address, a 128-bit link key K . This mode is utilized when creating unit keys and combination keys. In the second mode the function E_2 should produce, on input of a 128-bit RAND value and an L octet user PIN, a 128-bit link key K . The second mode is used to create the initialization key, and also whenever a master key is to be generated.

When the initialization key is generated, the PIN is augmented with the BD_ADDR of the claimant unit. The augmentation always starts with the least significant octet of the address immediately following the most significant octet of the PIN. Since the maximum length of the PIN used in the algorithm cannot exceed 16 octets, it is possible that not all octets of BD_ADDR will be used.

This key generating algorithm again exploits the cryptographic function. Formally E_2 can be expressed for mode 1 (denoted E_{21}) as

$$E_{21}: \{0, 1\}^{128} \times \{0, 1\}^{48} \rightarrow \{0, 1\}^{128} \quad (\text{EQ 38})$$

$$(\text{RAND}, \text{address}) \mapsto A'_r(X, Y)$$

where (for mode 1)

$$\begin{cases} X = \text{RAND}[0\dots 14] \cup (\text{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \text{address}[i \pmod{6}] \end{cases} \quad (\text{EQ 39})$$

Let L be the number of octets in the user PIN. The augmenting is defined by

$$\text{PIN}' = \begin{cases} \text{PIN}[0\dots L-1] \cup \text{BD_ADDR}_B[0\dots \min\{5, 15-L\}], & L < 16, \\ \text{PIN}[0\dots L-1], & L = 16, \end{cases} \quad (\text{EQ 40})$$

where it is assumed that unit B is the claimant. Then, in mode 2, E_2 (denoted E_{22}) can be expressed as

$$E_{22}: \{0, 1\}^{8L'} \times \{0, 1\}^{128} \times \{1, 2, \dots, 16\} \rightarrow \{0, 1\}^{128} \quad (\text{EQ 41})$$

$$(\text{PIN}', \text{RAND}, L') \mapsto A'_r(X, Y)$$

where

$$\begin{cases} X = \bigcup_{i=0}^{15} \text{PIN}'[i \pmod{L'}], \\ Y = \text{RAND}[0\dots 14] \cup (\text{RAND}[15] \oplus L'), \end{cases} \quad (\text{EQ 42})$$

and $L' = \min\{16, L+6\}$ is the number of octets in PIN'.

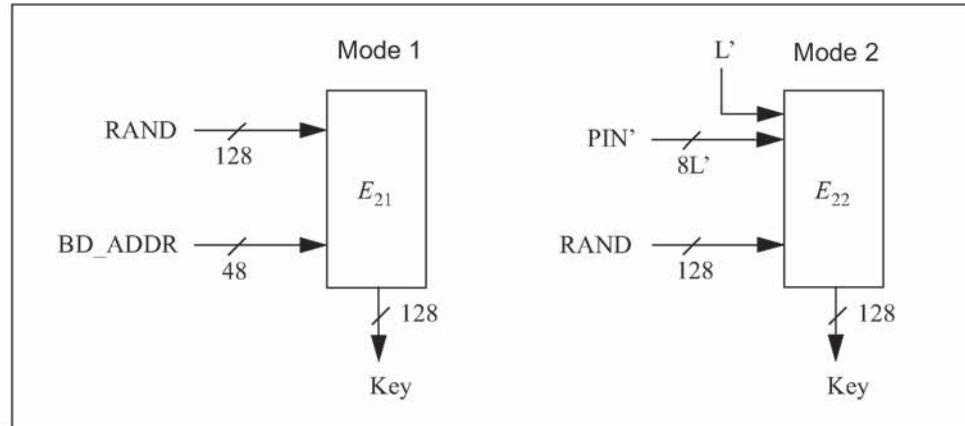


Figure 14.15: Key generating algorithm E_2 and its two modes. Mode 1 is used for unit and combination keys, while mode 2 is used for K_{init} and K_{master} .

14.5.4 E_3 -Key generation function for encryption

The ciphering key K_C used by E_0 is generated by E_3 . The function E_3 is constructed using A_7 as follows

$$E_3: \{0, 1\}^{128} \times \{0, 1\}^{128} \times \{0, 1\}^{96} \rightarrow \{0, 1\}^{128} \tag{EQ 43}$$

$$(K, RAND, COF) \mapsto Hash(K, RAND, COF, 12)$$

where $Hash$ is the hash function as defined by (EQ 33). Note that the produced key length is 128 bits. However, before use within E_0 , the encryption key K_C will be shortened to the correct encryption key length, as described in Section 14.3.5 on page 165. A block scheme of E_3 is depicted in Figure 14.16.

The value of COF is determined as specified by equation (EQ 23).

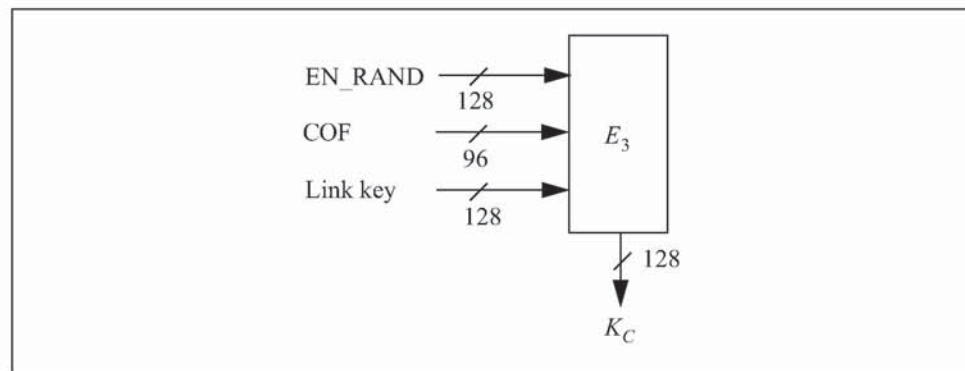


Figure 14.16: Generation of the encryption key.

15 LIST OF FIGURES

Figure 1.1:	Different functional blocks in the Bluetooth system	41
Figure 1.2:	Piconets with a single slave operation (a), a multi-slave operation (b) and a scatternet operation (c).	42
Figure 2.1:	TDD and timing	44
Figure 2.2:	Multi-slot packets	44
Figure 4.1:	Standard packet format.	47
Figure 4.2:	Access code format	48
Figure 4.3:	Preamble	49
Figure 4.4:	Trailer in CAC when MSB of sync word is 0 (a), and when MSB of sync word is 1 (b).	50
Figure 4.5:	Header format.	51
Figure 4.6:	Format of the FHS payload	56
Figure 4.7:	DV packet format	59
Figure 4.8:	Payload header format for single-slot packets.	62
Figure 4.9:	Payload header format for multi-slot packets.	62
Figure 5.1:	Bit-repetition encoding scheme.	67
Figure 5.2:	LFSR generating the (15,10) shortened Hamming code.	68
Figure 5.3:	Receive protocol for determining the ARQN bit.	70
Figure 5.4:	Retransmit filtering for packets with CRC.	71
Figure 5.5:	Broadcast repetition scheme	73
Figure 5.6:	The LFSR circuit generating the HEC.	74
Figure 5.7:	Initial state of the HEC generating circuit.	75
Figure 5.8:	HEC generation and checking.	75
Figure 5.9:	The LFSR circuit generating the CRC.	75
Figure 5.10:	Initial state of the CRC generating circuit.	76
Figure 5.11:	CRC generation and checking	76
Figure 7.1:	Data whitening LFSR.	79
Figure 8.1:	Functional diagram of TX buffering.	81
Figure 8.2:	Functional diagram of RX buffering	84
Figure 8.3:	Header bit processes.	86
Figure 8.4:	Payload bit processes.	86
Figure 9.1:	RX/TX cycle of Bluetooth master transceiver in normal mode for single-slot packets.	88
Figure 9.2:	RX/TX cycle of Bluetooth slave transceiver in normal mode for single-slot packets.	88
Figure 9.3:	RX timing of slave returning from hold state.	90

Figure 9.4: RX/TX cycle of Bluetooth transceiver in PAGE mode. 91

Figure 9.5: Timing of FHS packet on successful page in first half slot. 92

Figure 9.6: Timing of FHS packet on successful page in second half slot. . 92

Figure 9.7: RX/TX timing in multi-slave configuration 93

Figure 10.1: Bluetooth clock. 96

Figure 10.2: Derivation of CLKE 97

Figure 10.3: Derivation of CLK in master (a) and in slave (b). 97

Figure 10.4: State diagram of Bluetooth link controller. 98

Figure 10.5: Conventional page (a), page while one SCO link present (b),
page while two SCO links present (c). 103

Figure 10.6: Messaging at initial connection when slave responds to first page
message. 105

Figure 10.7: Messaging at initial connection when slave responds to second
page message. 105

Figure 10.8: General beacon channel format 117

Figure 10.9: Definition of access window 117

Figure 10.10: Access procedure applying the polling technique. 118

Figure 10.11: Disturbance of access window by SCO traffic 118

Figure 10.12: Extended sleep interval of parked slaves. 119

Figure 11.1: General block diagram of hop selection scheme. 128

Figure 11.2: Hop selection scheme in CONNECTION state. 128

Figure 11.3: Block diagram of hop selection kernel for the 79-hop system. 129

Figure 11.4: Block diagram of hop selection kernel for the 23-hop system. 129

Figure 11.5: XOR operation for the 79-hop system. The 23-hop system is the
same except for the Z^{1/4}/Z⁴ wire that does not exist. 130

Figure 11.6: Permutation operation for the 79 hop system. 132

Figure 11.7: Permutation operation for the 23 hop system. 132

Figure 11.8: Butterfly implementation. 132

Figure 12.1: Block diagram of CVSD encoder with syllabic companding. .. 140

Figure 12.2: Block diagram of CVSD decoder with syllabic companding. .. 140

Figure 12.3: Accumulator procedure 140

Figure 13.1: Format of BD_ADDR 143

Figure 13.2: Construction of the sync word. 145

Figure 13.3: LFSR and the starting state to generate 147

Figure 14.1: Generation of unit key. When the unit key has been exchanged,
the initialization key shall be discarded in both units. 155

Figure 14.2: Generating a combination key. The old link key (K) shall be
discarded after the exchange of a new combination key has
succeeded 156

Figure 14.3: Master link key distribution and computation of the corresponding encryption key.159

Figure 14.4: Stream ciphering for Bluetooth with E0.160

Figure 14.5: Functional description of the encryption procedure162

Figure 14.6: Concept of the encryption engine.163

Figure 14.7: Overview of the operation of the encryption engine. Between each start of a packet (TX or RX), the LFSRs are re-initialized.165

Figure 14.8: Arranging the input to the LFSRs.168

Figure 14.9: Distribution of the 128 last generated output symbols within the LFSRs.168

Figure 14.10: Challenge-response for the Bluetooth.169

Figure 14.11: Challenge-response for symmetric key systems.170

Figure 14.12: Flow of data for the computation of E_1173

Figure 14.13: One round in A_r and A_r'174

Figure 14.14: Key scheduling in A_r175

Figure 14.15: Key generating algorithm E_2 and its two modes.
 Mode 1 is used for unit and combination keys, while mode 2 is used for K_{init} and K_{master} 177

Figure 14.16: Generation of the encryption key.177



16 LIST OF TABLES

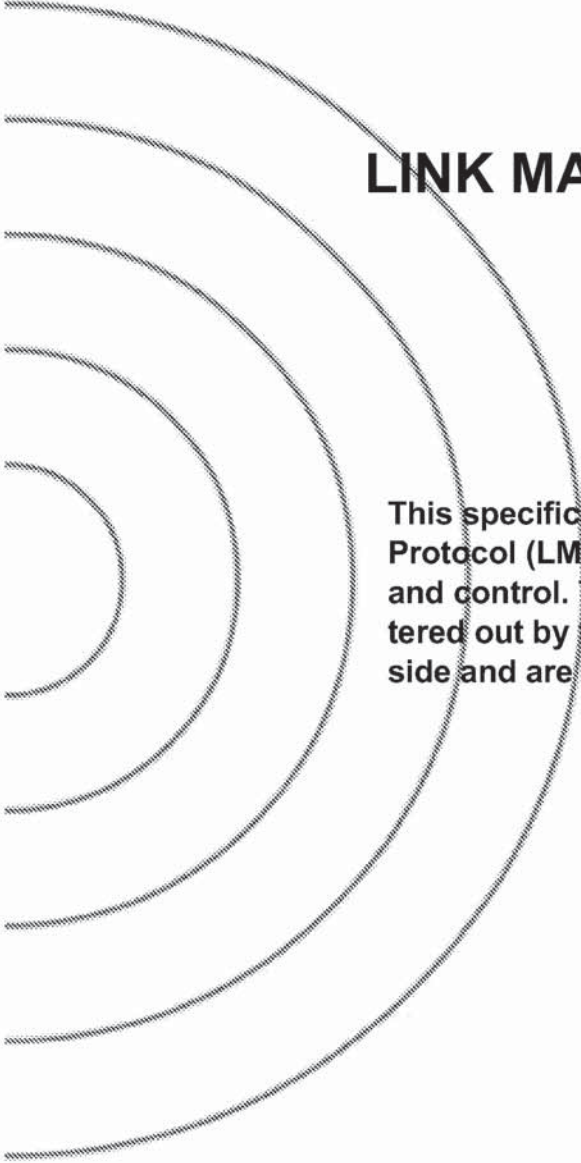
Table 2.1:	Available RF channels	43
Table 4.1:	Summary of access code types.	49
Table 4.2:	Packets defined for SCO and ACL link types.....	54
Table 4.3:	Description of the FHS payload	56
Table 4.4:	Contents of SR field	57
Table 4.5:	Contents of SP field	57
Table 4.6:	Contents of page scan mode field.....	58
Table 4.7:	Logical channel L_CH field contents.....	63
Table 4.8:	Use of payload header flow bit on the logical channels.	63
Table 4.9:	Link control packets	65
Table 4.10:	ACL packets.....	65
Table 4.11:	SCO packets	65
Table 10.1:	Relationship between scan interval, train repetition, and paging modes R0, R1 and R2.....	101
Table 10.2:	Relationship between train repetition, and paging modes R0, R1 and R2 when SCO links are present.....	103
Table 10.3:	Initial messaging during start-up.	104
Table 10.4:	Increase of train repetition when SCO links are present.....	111
Table 10.5:	Messaging during inquiry routines.	112
Table 10.6:	Mandatory scan periods for P0, P1, P2 scan period modes.	112
Table 11.1:	Control of the butterflies for the 79 hop system	131
Table 11.2:	Control of the butterflies for the 23 hop system	131
Table 11.3:	Control for 79-hop system.....	134
Table 11.4:	Control for 23-hop system.....	134
Table 12.1:	Voice coding schemes supported on the air interface.....	139
Table 12.2:	CVSD parameter values. The values are based on a 16 bit signed number output from the accumulator.....	141
Table 14.1:	Entities used in authentication and encryption procedures.....	149
Table 14.2:	Possible traffic modes for a slave using a semi-permanent link key.....	161
Table 14.3:	Possible encryption modes for a slave in possession of a master key.....	161
Table 14.4:	The four primitive feedback polynomials.....	164
Table 14.5:	The mappings T_1 and T_2	165

Table 14.6: Polynomials used when creating .
 All polynomials are in hexadecimal notation. The LSB is
 in the rightmost position. 167

Table 14.6: Polynomials used when creating .
 All polynomials are in hexadecimal notation. The LSB is
 in the rightmost position. 167

Part C

LINK MANAGER PROTOCOL



This specification describes the Link Manager Protocol (LMP) which is used for link set-up and control. The signals are interpreted and filtered out by the Link Manager on the receiving side and are not propagated to higher layers.



CONTENTS

1 General.....191

2 Format of LMP.....192

3 The Procedure Rules and PDUs193

3.1 General Response Messages.....193

3.2 Authentication194

3.2.1 Claimant has link key194

3.2.2 Claimant has no link key194

3.2.3 Repeated attempts.....195

3.3 Pairing.....195

3.3.1 Claimant accepts pairing195

3.3.2 Claimant requests to become verifier.....195

3.3.3 Claimant rejects pairing.....196

3.3.4 Creation of the link key.....196

3.3.5 Repeated attempts197

3.4 Change Link Key.....197

3.5 Change the Current Link Key.....198

3.5.1 Change to a temporary link key.....198

3.5.2 Make the semi-permanent link key the current link key199

3.6 Encryption199

3.6.1 Encryption mode200

3.6.2 Encryption key size200

3.6.3 Start encryption201

3.6.4 Stop encryption202

3.6.5 Change encryption mode, key or random number202

3.7 Clock Offset Request.....202

3.8 Slot Offset Information203

3.9 Timing Accuracy Information Request203

3.10 LMP Version.....205

3.11 Supported Features205

3.12 Switch of Master-Slave Role.....206

3.13 Name Request.....207

3.14 Detach.....207

3.15 Hold Mode.....208

3.15.1 Master forces hold mode.....208

3.15.2 Slave forces hold mode.....208

3.15.3 Master or slave requests hold mode209

3.16	Sniff Mode.....	209
3.16.1	Master forces a slave into sniff mode.....	210
3.16.2	Master or slave requests sniff mode	210
3.16.3	Moving a slave from sniff mode to active mode	211
3.17	Park Mode	211
3.17.1	Master forces a slave into park mode	213
3.17.2	Master requests slave to enter park mode.....	213
3.17.3	Slave requests to be placed in park mode.....	213
3.17.4	Master sets up broadcast scan window	214
3.17.5	Master modifies beacon parameters.....	214
3.17.6	Unparking slaves.....	214
3.18	Power Control	215
3.19	Channel Quality-driven Change Between DM and DH.....	217
3.20	Quality of Service (QoS).....	218
3.20.1	Master notifies slave of the quality of service.....	218
3.20.2	Device requests new quality of service	219
3.21	SCO Links.....	219
3.21.1	Master initiates an SCO link.....	220
3.21.2	Slave initiates an SCO link.....	220
3.21.3	Master requests change of SCO parameters.....	221
3.21.4	Slave requests change of SCO parameters.....	221
3.21.5	Remove an SCO link.....	221
3.22	Control of Multi-slot Packets	222
3.23	Paging Scheme	223
3.23.1	Page mode.....	223
3.23.2	Page scan mode	223
3.24	Link Supervision	224
4	Connection Establishment.....	225
5	Summary of PDUs.....	226
5.1	Description of Parameters	231
5.1.1	Coding of features.....	234
5.1.2	List of error reasons	235
5.2	Default Values	236

6	Test Modes	237
6.1	Activation and Deactivation of Test Mode	237
6.2	Control of Test Mode	237
6.3	Summary of Test Mode PDUs	238
7	Error Handling	239
8	List of Figures	241
9	List of Tables	243



1 GENERAL

LMP messages are used for link set-up, security and control. They are transferred in the payload instead of L2CAP and are distinguished by a reserved value in the L_CH field of the payload header. The messages are filtered out and interpreted by LM on the receiving side and are not propagated to higher layers.

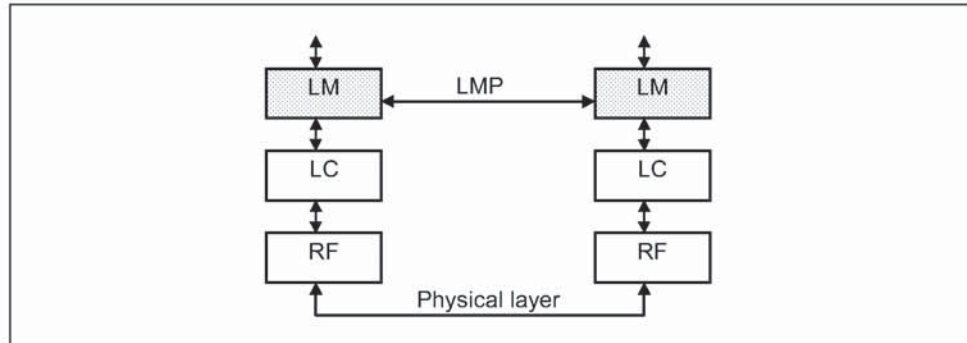


Figure 1.1: Link Manager's place on the global scene.

Link Manager messages have higher priority than user data. This means that if the Link Manager needs to send a message, it shall not be delayed by the L2CAP traffic, although it can be delayed by many retransmissions of individual baseband packets.

We do not need to explicitly acknowledge the messages in LMP since LC (see Baseband Specification Section 5, on page 67) provides us with a reliable link.

The time between receiving a baseband packet carrying an LMP PDU and sending a baseband packet carrying a valid response PDU, according to the procedure rules in Section 3 on page 193, must be less than the LMP Response Timeout. The value of this timeout is 30 seconds.

2 FORMAT OF LMP

LM PDUs are always sent as single-slot packets and the payload header is therefore one byte. The two least significant bits in the payload header determine the logical channel. For LM PDUs these bits are set.

L_CH code	Logical Channel	Information
00	NA	undefined
01	UA/I	Continuing L2CAP message
10	UA/I	Start L2CAP message
11	LM	LMP message

Table 2.1: Logical channel L_CH field contents.

The FLOW bit in the payload header is always one and is ignored on the receiving side. Each PDU is assigned a 7-bit opcode used to uniquely identify different types of PDUs, see Table 5.1 on page 226. The opcode and a one-bit transaction ID are positioned in the first byte of the payload body. The transaction ID is positioned in the LSB. It is 0 if the PDU belongs to a transaction initiated by the master and 1 if the PDU belongs to a transaction initiated by the slave. If the PDU contains one or more parameters these are placed in the payload starting at the second byte of the payload body. The number of bytes used depends on the length of the parameters. If an SCO link is present using HV1 packets and length of *content* is less than 9 bytes the PDUs can be transmitted in DV packets. Otherwise DM1 packets must be used. All parameters have little endian format, i.e. the least significant byte is transmitted first.

The source/destination of the PDUs is determined by the AM_ADDR in the packet header.

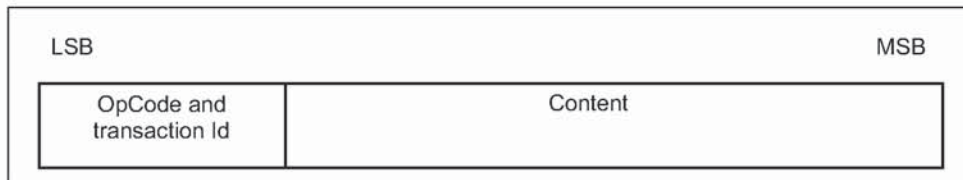


Figure 2.1: Payload body when LM PDUs are sent.

Each PDU is either mandatory or optional. The M/O field in the tables of Section 3 indicates this. The LM does not need to be able to transmit a PDU that is optional. The LM must recognize all optional PDUs that it receives and, if a response is required, send a valid response according to the procedure rules in Section 3. The reason that should be used in this case is *unsupported LMP feature*. If the optional PDU that is received does not require a response, no response is sent. Which of the optional PDUs a device supports can be requested, see Section 3.11 on page 205.

3 THE PROCEDURE RULES AND PDUs

Each procedure is described and depicted with a sequence diagram. The following symbols are used in the sequence diagrams:

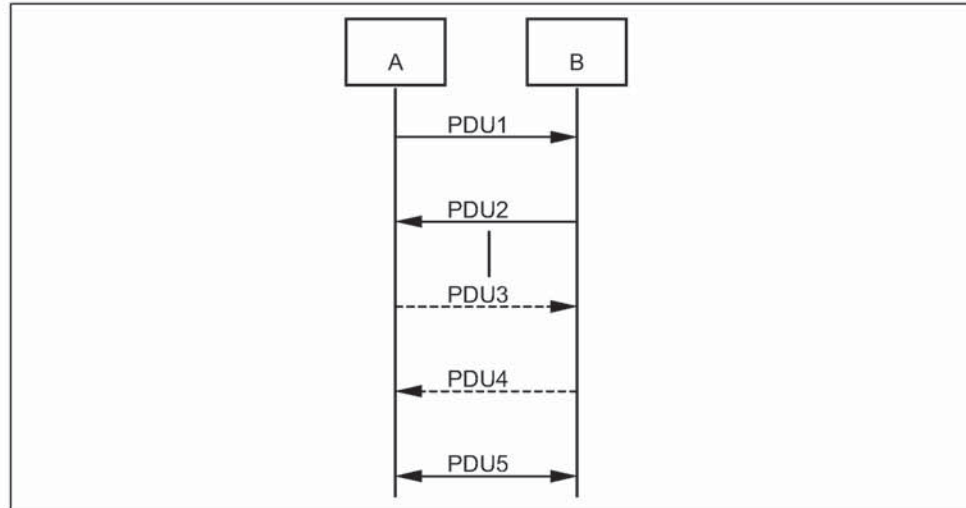


Figure 3.1: Symbols used in sequence diagrams.

PDU1 is a PDU sent from A to B. PDU2 is a PDU sent from B to A. PDU3 is a PDU that is optionally sent from A to B. PDU4 is a PDU that is optionally sent from B to A. PDU5 is a PDU sent from either A or B. A vertical line indicates that more PDUs can optionally be sent.

3.1 GENERAL RESPONSE MESSAGES

The PDUs LMP_accepted and LMP_not_accepted are used as response messages to other PDUs in a number of different procedures. The PDU LMP_accepted includes the opcode of the message that is accepted. The PDU LMP_not_accepted includes the opcode of the message that is not accepted and the reason why it is not accepted.

M/O	PDU	Contents
M	LMP_accepted	op code
M	LMP_not_accepted	op code reason

Table 3.1: General response messages.

3.2 AUTHENTICATION

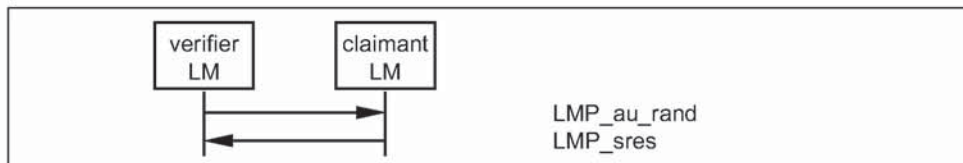
The authentication procedure is based on a challenge-response scheme as described in Baseband Specification Section 14.4, on page 169. The verifier sends an LMP_au_rand PDU which contains a random number (the challenge) to the claimant. The claimant calculates a response, which is a function of the challenge, the claimant's BD_ADDR and a secret key. The response is sent back to the verifier, which checks if the response was correct or not. How the response should be calculated is described in Baseband Specification Section 14.5.1, on page 171. A successful calculation of the authentication response requires that two devices share a secret key. How this key is created is described in Section 3.3 on page 195. Both the master and the slave can be verifiers. The following PDUs are used in the authentication procedure:

M/O	PDU	Contents
M	LMP_au_rand	random number
M	LMP_sres	authentication response

Table 3.2: PDUs used for authentication.

3.2.1 Claimant has link key

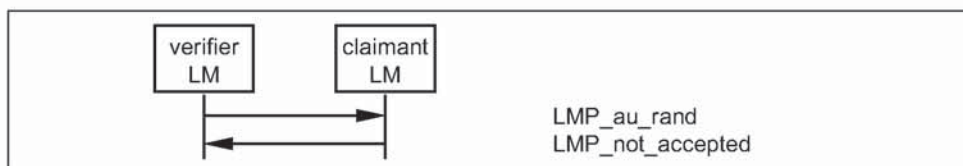
If the claimant has a link key associated with the verifier, it calculates the response and sends it to the verifier with LMP_sres. The verifier checks the response. If the response is not correct, the verifier can end the connection by sending LMP_detach with the reason code *authentication failure*, see Section 3.14 on page 207.



Sequence 1: Authentication. Claimant has link key.

3.2.2 Claimant has no link key

If the claimant does not have a link key associated with the verifier it sends LMP_not_accepted with the reason code *key missing* after receiving LMP_au_rand.



Sequence 2: Authentication fails. Claimant has no link key.

3.2.3 Repeated attempts

The scheme described in Baseband Specification Section 14.4.1, on page 170 shall be applied when an authentication fails. This will prevent an intruder from trying a large number of keys in a relatively short time.

3.3 PAIRING

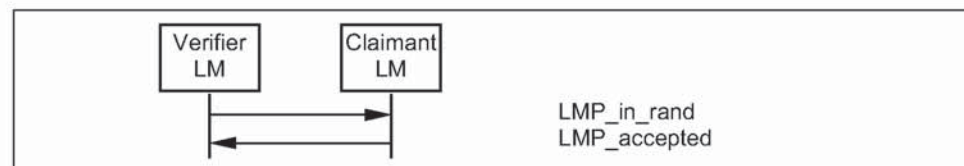
When two devices do not have a common link key an initialization key (K_{init}) is created based on a PIN and a random number. The K_{init} is created when the verifier sends LMP_in_rand to the claimant. How the K_{init} is calculated is described in Baseband Specification Section 14.5.3, on page 175. Authentication then needs to be done, whereby the calculation of the authentication response is based on K_{init} instead of the link key. After a successful authentication, the link key is created. The PDUs used in the pairing procedure are:

M/O	PDU	Contents
M	LMP_in_rand	random number
M	LMP_au_rand	random number
M	LMP_sres	authentication response
M	LMP_comb_key	random number
M	LMP_unit_key	key

Table 3.3: PDUs used for pairing.

3.3.1 Claimant accepts pairing

The verifier sends LMP_in_rand and the claimant replies with LMP_accepted. Both devices calculate K_{init} , and an authentication (see Sequence 1) based on this key needs to be done. The verifier checks the authentication response and if correct, the link key is created; see Section 3.3.4 on page 196. If the authentication response is not correct the verifier can end the connection by sending LMP_detach with the reason code *authentication failure*.

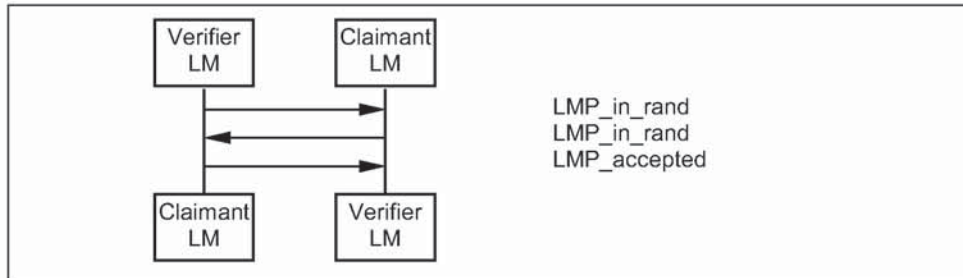


Sequence 3: Claimant accepts pairing.

3.3.2 Claimant requests to become verifier

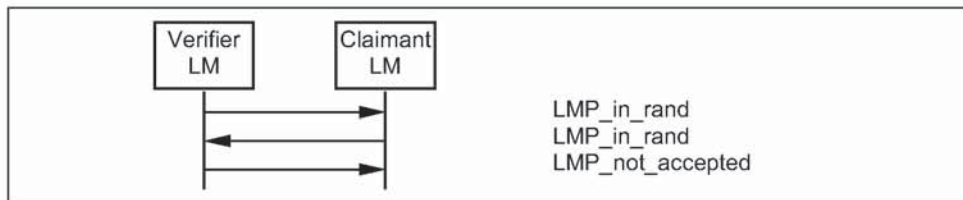
If the claimant has a fixed PIN it may request a switch of the claimant-verifier role in the pairing procedure by generating a new random number and send it

back in LMP_in_rand. If the device that started the pairing procedure has a variable PIN it must accept this and respond with LMP_accepted. The roles are then successfully switched and the pairing procedure continues as described in Section 3.3.1 on page 195.



Sequence 4: Claimant accepts pairing but requests to be verifier.

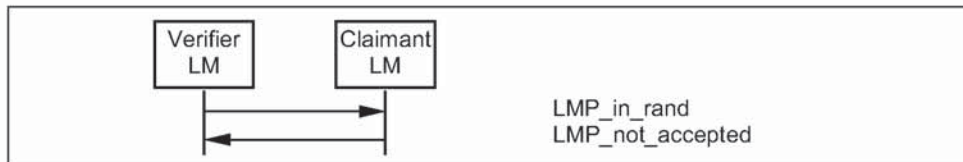
If the device that started the pairing procedure has a fixed PIN and the other device requests a role switch, the switch is rejected by sending LMP_not_accepted with the reason *pairing not allowed*; the pairing procedure is then ended.



Sequence 5: Unsuccessful switch of claimant-verifier role.

3.3.3 Claimant rejects pairing

If the claimant rejects pairing, it sends LMP_not_accepted with the reason *pairing not allowed* after receiving LMP_in_rand.



Sequence 6: Claimant rejects pairing.

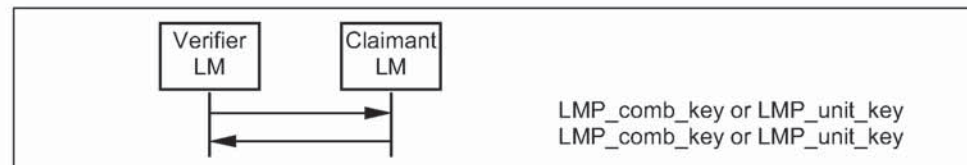
3.3.4 Creation of the link key

When the authentication is finished the link key must be created. This link key will be used in the authentication between the two units for all subsequent connections until it is changed; see Section 3.4 and Section 3.5. The link key cre-

ated in the pairing procedure will either be a combination key or one of the unit's unit keys. The following rules apply to the selection of the link key:

- if one unit sends LMP_unit_key and the other unit sends LMP_comb_key, the unit key will be the link key,
- if both units send LMP_unit_key, the master's unit key will be the link key,
- if both units send LMP_comb_key, the link key is calculated as described in Baseband Specification Section 14.2.2, on page 153.

The content of LMP_unit_key is the unit key bitwise XORed with K_{init} . The content of LMP_comb_key is LK_RAND bitwise XORed with K_{init} . Any device configured to use a combination key will store the link key in non-volatile memory.



Sequence 7: Creation of the link key.

3.3.5 Repeated attempts

When the authentication during pairing fails because of a wrong authentication response, the same scheme is applied as in Section 3.2.3 on page 195. This prevents an intruder from trying a large number of different PINs in a relatively short time.

3.4 CHANGE LINK KEY

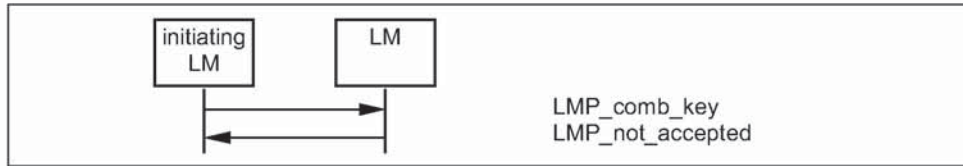
If two devices are paired and the link key is derived from combination keys, the link key can be changed. If the link key is a unit key, the units must go through the pairing procedure in order to change the link key. The contents of the PDU is protected by a bitwise XOR with the current link key.

M/O	PDU	Contents
M	LMP_comb_key	random number
M	LMP_unit_key	key

Table 3.4: PDUs used for change of link key.



Sequence 8: Successful change of the link key.



Sequence 9: Change of the link key not possible since the other unit uses a unit key.

If the change of link key is successful the new link key is stored in non-volatile memory, and the old link key is discarded. The new link key will be used as link key for all the following connections between the two devices until the link key is changed again. The new link key also becomes the current link key. It will remain the current link key until the link key is changed again, or until a temporary link key is created, see Section 3.5 on page 198.

If encryption is used on the link and the current link key is a temporary link key, the procedure of changing link key must be immediately followed by a stop of the encryption by invoking the procedure in Section 3.6.4 on page 202. Encryption can then be started again. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semi-permanent link key is the current link key.

3.5 CHANGE THE CURRENT LINK KEY

The current link key can be a semi-permanent link key or a temporary link key. It can be changed temporarily, but the change is only valid for the session, see Baseband Specification Section 14.2.1, on page 151. Changing to a temporary link key is necessary if the piconet is to support encrypted broadcast.

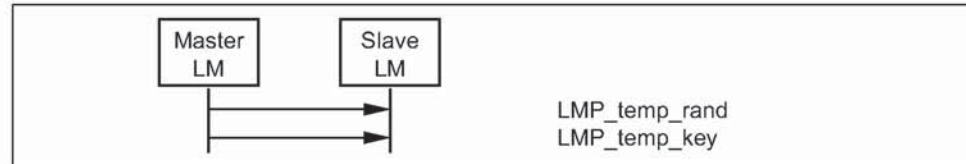
M/O	PDU	Contents
M	LMP_temp_rand	random number
M	LMP_temp_key	key
M	LMP_use_semi_permanent_key	-

Table 3.5: PDUs used to change the current link key.

3.5.1 Change to a temporary link key

In the following, we use the same terms as in Baseband Specification Section 14.2.2.8, on page 158. The master starts by creating the master key K_{master} as described in Baseband Specification (EQ 24), on page 158. Then the master issues a random number RAND and sends it to the slave in LMP_temp_rand. Both sides can then calculate an overlay denoted OVL as $\text{OVL} = E_{22}(\text{current link key}, \text{RAND}, 16)$. Then the master sends K_{master} protected by a modulo-2 addition with OVL to the slave in LMP_temp_key. The slave, who knows OVL,

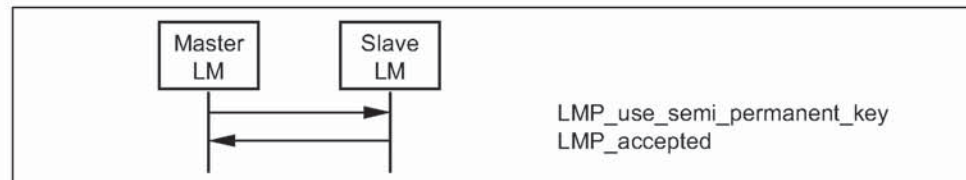
calculates K_{master} . After this, K_{master} becomes the current link key. It will be the current link key until a new temporary key is created or until the link key is changed, see Section 3.4 on page 197.



Sequence 10: Change to a temporary link key.

3.5.2 Make the semi-permanent link key the current link key

After the current link key has been changed to K_{master} , this change can be undone and the semi-permanent link key becomes the current link key again. If encryption is used on the link, the procedure of going back to the semi-permanent link key must be immediately followed by a stop of the encryption by invoking the procedure described in Section 3.6.4 on page 202. Encryption can then be started again. This is to assure that encryption with encryption parameters known by other devices in the piconet is not used when the semi-permanent link key is the current link key.



Sequence 11: Link key changed to the semi-permanent link key.

3.6 ENCRYPTION

If at least one authentication has been performed encryption may be used. If the master wants all slaves in the piconet to use the same encryption parameters it must issue a temporary key (K_{master}) and make this key the current link key for all slaves in the piconet before encryption is started, see Section 3.5 on page 198. This is necessary if broadcast packets should be encrypted.

M/O	PDU	Contents
O	LMP_encryption_mode_req	encryption mode
O	LMP_encryption_key_size_req	key size
O	LMP_start_encryption_req	random number
O	LMP_stop_encryption_req	-

Table 3.6: PDUs used for handling encryption.

3.6.1 Encryption mode

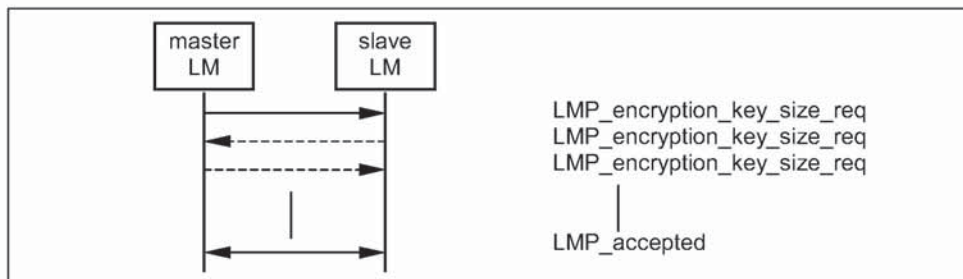
First of all the master and the slave must agree upon whether to use encryption or not and if encryption shall only apply to point-to-point packets or if encryption shall apply to both point-to-point packets and broadcast packets. If master and slave agree on the encryption mode, the master continues to give more detailed information about the encryption.



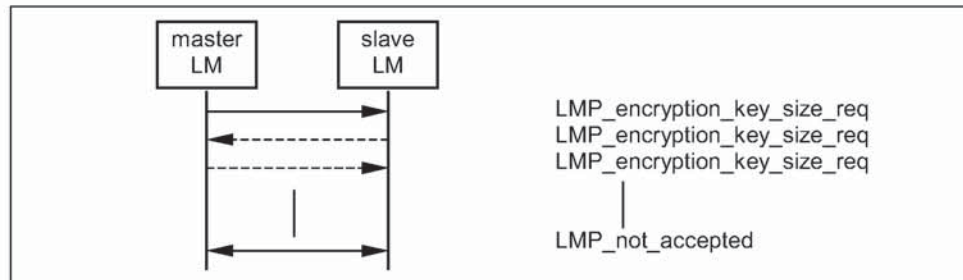
Sequence 12: Negotiation for encryption mode.

3.6.2 Encryption key size

The next step is to determine the size of the encryption key. In the following we use the same terms as in Baseband Specification Section 14.3.1, on page 160. The master sends `LMP_encryption_key_size_req` including the suggested key size $L_{sug, m}$, which is initially equal to $L_{max, m}$. If $L_{min, s} \leq L_{sug, m}$ and the slave supports $L_{sug, m}$ it responds with `LMP_accepted` and $L_{sug, m}$ will be used as the key size. If both conditions are not fulfilled the slave sends back `LMP_encryption_key_size_req` including the slave's suggested key size $L_{sug, s}$. This value is the slave's largest supported key size that is less than $L_{sug, m}$. Then the master performs the corresponding test on the slave's suggestion. This procedure is repeated until a key size agreement is reached or it becomes clear that no such agreement can be reached. If an agreement is reached a unit sends `LMP_accepted` and the key size in the last `LMP_encryption_key_size_req` will be used. After this, the encryption is started; see Section 3.6.3 on page 201. If an agreement is not reached a unit sends `LMP_not_accepted` with the reason code *Unsupported parameter value* and the units are not allowed to communicate using Bluetooth link encryption."



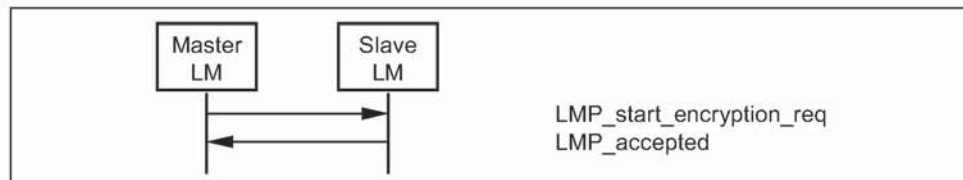
Sequence 13: Encryption key size negotiation successful.



Sequence 14: Encryption key size negotiation failed.

3.6.3 Start encryption

Finally, encryption is started. The master issues the random number EN RAND and calculates the encryption key as $K_c = E_3(\text{current link key, EN_RAND, COF})$. See Baseband Specification Section 14.2.2.5, on page 156 and 14.2.2.2 for the definition of the COF. The random number must be the same for all slaves if the piconet should support encrypted broadcast. Then the master sends LMP_start_encryption_req, which includes EN RAND. The slave calculates K_c when this message is received and acknowledges with LMP_accepted. On both sides, K_c and EN RAND are used as input to the encryption algorithm E_o .



Sequence 15: Start of encryption.

Before starting encryption, higher-layer data traffic must be temporarily stopped to prevent reception of corrupt data. The start of encryption will be done in three steps:

1. Master is configured to transmit unencrypted packets, but to receive encrypted packets.
2. Slave is configured to transmit and receive encrypted packets.
3. Master is configured to transmit and receive encrypted packets.

Between step 1 and step 2, master-to-slave transmission is possible. This is when LMP_start_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3, slave-to-master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

3.6.4 Stop encryption



Sequence 16: Stop of encryption.

Before stopping encryption, higher-layer data traffic must be temporarily stopped to prevent reception of corrupt data. Stopping of encryption is then done in three steps, similar to the procedure for starting encryption.

1. Master is configured to transmit encrypted packets, but to receive unencrypted packets.
2. Slave is configured to transmit and receive unencrypted packets.
3. Master is configured to transmit and receive unencrypted packets.

Between step 1 and step 2 master to slave transmission is possible. This is when LMP_stop_encryption_req is transmitted. Step 2 is triggered when the slave receives this message. Between step 2 and step 3 slave to master transmission is possible. This is when LMP_accepted is transmitted. Step 3 is triggered when the master receives this message.

3.6.5 Change encryption mode, key or random number

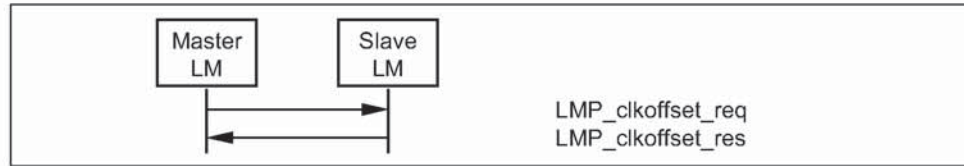
If the encryption mode, encryption key or encryption random number need to be changed, encryption must first be stopped and then re-started with the new parameters.

3.7 CLOCK OFFSET REQUEST

When a slave receives the FHS packet, the difference is computed between its own clock and the master's clock included in the payload of the FHS packet. The clock offset is also updated each time a packet is received from the master. The master can request this clock offset anytime during the connection. By saving this clock offset the master knows on what RF channel the slave wakes up to PAGE SCAN after it has left the piconet. This can be used to speed up the paging time the next time the same device is paged.

M/O	PDU	Contents
M	LMP_clkoffset_req	-
M	LMP_clkoffset_res	clock offset

Table 3.7: PDUs used for clock offset request.



Sequence 17: Clock offset requested.

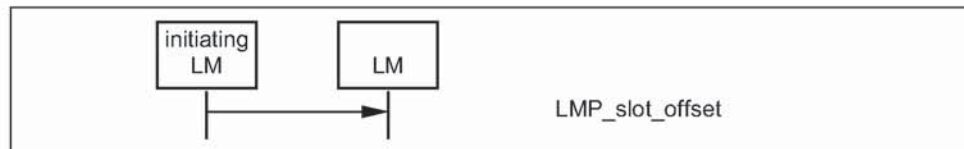
3.8 SLOT OFFSET INFORMATION

With LMP_slot_offset the information about the difference between the slot boundaries in different piconets is transmitted. This PDU carries the parameters slot offset and BD_ADDR. The slot offset is the time in μs between the start of the master's TX slot in the piconet where the PDU is transmitted and the start of the master's TX slot in the piconet where the BD_ADDR device is master.

Before doing a master-slave switch, see Section 3.12 on page 206, this PDU shall be transmitted from the device that becomes master in the switch procedure. If the master initiates the switch procedure, the slave sends LMP_slot_offset before sending LMP_accepted. If the slave initiates the switch procedure, the slave sends LMP_slot_offset before sending LMP_switch_req. The PDU can also be useful in inter-piconet communications.

M/O	PDU	Contents
O	LMP_slot_offset	slot offset BD_ADDR

Table 3.8: PDU used for slot offset information.



Sequence 18: Slot offset information is sent.

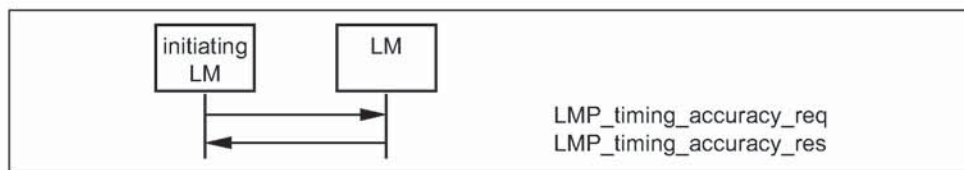
3.9 TIMING ACCURACY INFORMATION REQUEST

LMP supports requests for the timing accuracy. This information can be used to minimize the scan window for a given hold time when returning from hold and to extend the maximum hold time. It can also be used to minimize the scan window when scanning for the sniff mode slots or the park mode beacon packets. The timing accuracy parameters returned are the long term drift measured in ppm and the long term jitter measured in μs of the clock used during hold, sniff and park mode. These parameters are fixed for a certain device and must be identical when requested several times. If a device does not support the tim-

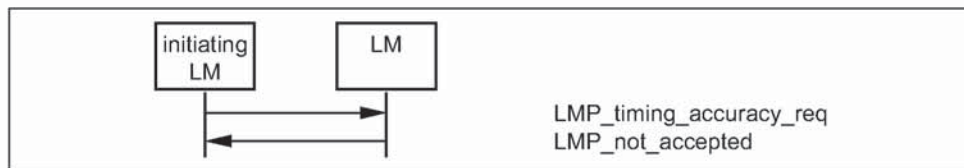
ing accuracy information it sends LMP_not_accepted with the reason code *unsupported LMP feature* when the request is received. The requesting device must in this case assume worst case values (drift=250ppm and jitter=10µs).

M/O	PDU	Contents
O	LMP_timing_accuracy_req	-
O	LMP_timing_accuracy_res	drift jitter

Table 3.9: PDUs used for requesting timing accuracy information.



Sequence 19: The requested device supports timing accuracy information.



Sequence 20: The requested device does not support timing accuracy information.

3.10 LMP VERSION

LMP supports requests for the version of the LM protocol. The requested device will send a response with three parameters: VersNr, Compld and SubVersNr. VersNr specifies the version of the Bluetooth LMP specification that the device supports. Compld is used to track possible problems with the lower Bluetooth layers. All companies that create a unique implementation of the Link Manager shall have their own Compld. The same company is also responsible for the administration and maintenance of the SubVersNr. It is recommended that each company has a unique SubVersNr for each RF/BB/LM implementation. For a given VersNr and Compld, the values of the SubVersNr must increase each time a new implementation is released. For both Compld and SubVersNr the value 0xFFFF means that no valid number applies. There is no ability to negotiate the version of the LMP. The sequence below is only used to exchange the parameters.

M/O	PDU	Contents
M	LMP_version_req	VersNr Compld SubVersNr
M	LMP_version_res	VersNr Compld SubVersNr

Table 3.10: PDUs used for LMP version request.



Sequence 21: Request for LMP version.

3.11 SUPPORTED FEATURES

The Bluetooth radio and link controller may support only a subset of the packet types and features described in Baseband Specification and Radio Specification. The PDU LMP_features_req and LMP_features_res are used to exchange this information. A device may not send any packets other than ID, FHS, NULL, POLL, DM1 or DH1 before it is aware of the supported features of the other device. After the features request has been carried out, the intersection of the supported packet types for both sides may also be transmitted. Whenever a request is issued, it must be compatible with the supported features of the other device. For instance, when establishing an SCO link the initiator may not propose to use HV3 packets if that packet type is not supported by the other device. Exceptions to this rule are LMP switch reg and LMP slot offset, which can be sent as the first LMP messages when two Bluetooth

devices have been connected and before the requesting side is aware of the other side's features (switch is an optional feature).

M/O	PDU	Contents
M	LMP_features_req	features
M	LMP_features_res	features

Table 3.11: PDUs used for features request.



Sequence 22: Request for supported features.

3.12 SWITCH OF MASTER-SLAVE ROLE

Since the paging device always becomes the master of the piconet, a switch of the master-slave role is sometimes needed, see Baseband Specification Section 10.9.3, on page 123. Suppose device A is slave and device B is master. The device that initiates the switch finalizes the transmission of the current L2CAP message and then sends LMP_switch_req.

If the switch is accepted, the other device finalizes the transmission of the current L2CAP message and then responds with LMP_accepted. After this, the procedure described from the 2nd bullet in Baseband Specification Section 10.9.3, on page 123 is carried out.

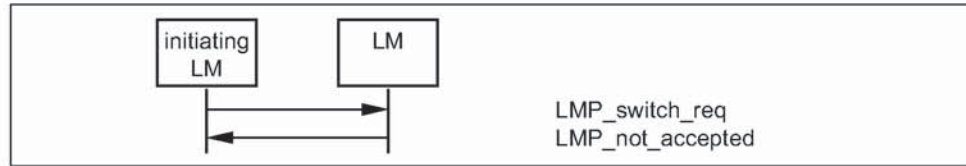
If the switch is rejected, the other device responds with LMP_not_accepted and no switch is performed.

M/O	PDU	Contents
O	LMP_switch_req	-

Table 3.12: PDU used for master slave switch.



Sequence 23: Master-slave switch accepted.



Sequence 24: Master-slave switch not accepted.

3.13 NAME REQUEST

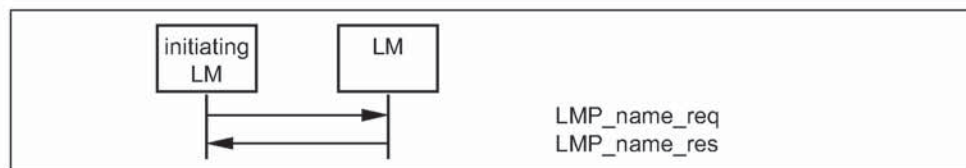
LMP supports name request to another Bluetooth device. The name is a user-friendly name associated with the Bluetooth device and consists of a maximum of 248 bytes coded according to the UTF-8 standard. The name is fragmented over one or more DM1 packets. When the LMP_name_req is sent, a name offset indicates which fragment is expected. The corresponding LMP_name_res carries the same name offset, the name length indicating the total number of bytes in the name of the Bluetooth device and the name fragment, where:

- name fragment(N) = name(N + name offset), if (N + name offset) < name length
- name fragment(N) = 0 , otherwise.

Here $0 \leq N \leq 13$. In the first sent LMP_name_req, name offset=0. Sequence 25 is then repeated until the initiator has collected all fragments of the name.

M/O	PDU	Contents
M	LMP_name_req	name offset
M	LMP_name_res	name offset name length name fragment

Table 3.13: PDUs used for name request.



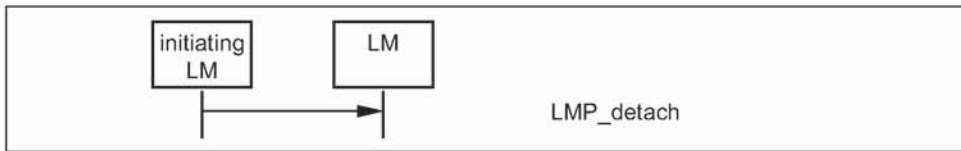
Sequence 25: Device's name requested and it responses.

3.14 DETACH

The connection between two Bluetooth devices can be closed anytime by the master or the slave. A reason parameter is included in the message to inform the other party of why the connection is closed.

M/O	PDU	Contents
M	LMP_detach	reason

Table 3.14: PDU used for detach.



Sequence 26: Connection closed by sending LMP_detach.

3.15 HOLD MODE

The ACL link of a connection between two Bluetooth devices can be placed in hold mode for a specified hold time. During this time no ACL packets will be transmitted from the master. The hold mode is typically entered when there is no need to send data for a relatively long time. The transceiver can then be turned off in order to save power. But the hold mode can also be used if a device wants to discover or be discovered by other Bluetooth devices, or wants to join other piconets. What a device actually does during the hold time is not controlled by the hold message, but it is up to each device to decide.

M/O	PDU	Contents
O	LMP_hold	hold time
O	LMP_hold_req	hold time

Table 3.15: PDUs used for hold mode.

3.15.1 Master forces hold mode

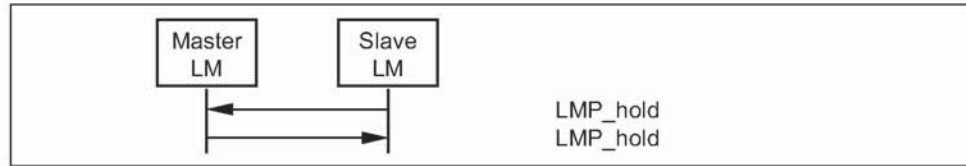
The master can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the master forces hold mode cannot be longer than any hold time the slave has previously accepted when there was a request for hold mode.



Sequence 27: Master forces slave into hold mode.

3.15.2 Slave forces hold mode

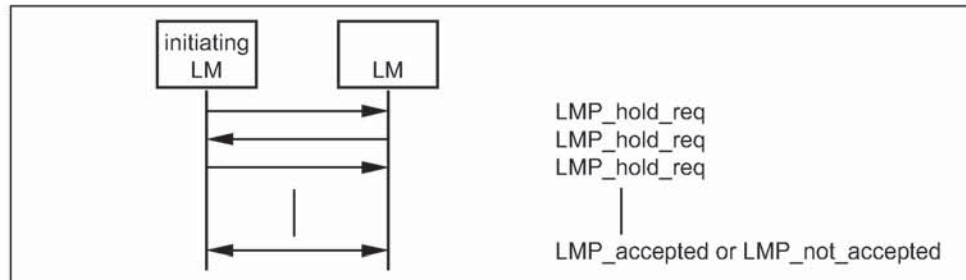
The slave can force hold mode if there has previously been a request for hold mode that has been accepted. The hold time included in the PDU when the slave forces hold mode cannot be longer than any hold time the master has previously accepted when there was a request for hold mode.



Sequence 28: Slave forces master into hold mode.

3.15.3 Master or slave requests hold mode

The master or the slave can request to enter hold mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen LMP_accepted terminates the negotiation and the ACL link is placed in hold mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and hold mode is not entered.



Sequence 29: Negotiation for hold mode.

3.16 SNIFF MODE

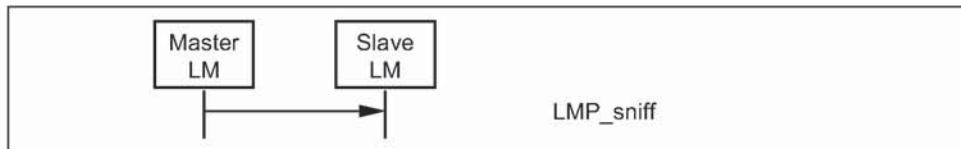
To enter sniff mode, master and slave negotiate a sniff interval T_{sniff} and a sniff offset, D_{sniff} , which specifies the timing of the sniff slots. The offset determines the time of the first sniff slot; after that the sniff slots follows periodically with the sniff interval T_{sniff} . To avoid problems with a clock wrap-around during the initialization, one of two options is chosen for the calculation of the first sniff slot. A timing control flag in the message from the master indicates this. Note: Only bit1 of this field is valid.

When the link is in sniff mode the master can only start a transmission in the sniff slot. Two parameters control the listening activity in the slave. The sniff attempt parameter determines for how many slots the slave must listen, beginning at the sniff slot, even if it does not receive a packet with its own AM address. The sniff timeout parameter determines for how many additional slots the slave must listen if it continues to receive only packets with its own AM address.

M/O	PDU	Contents
O	LMP_sniff	timing control flags D _{sniff} T _{sniff} sniff attempt sniff timeout
O	LMP_sniff_req	timing control flags D _{sniff} T _{sniff} sniff attempt sniff timeout
O	LMP_unsniff_req	-

Table 3.16: PDUs used for sniff mode.

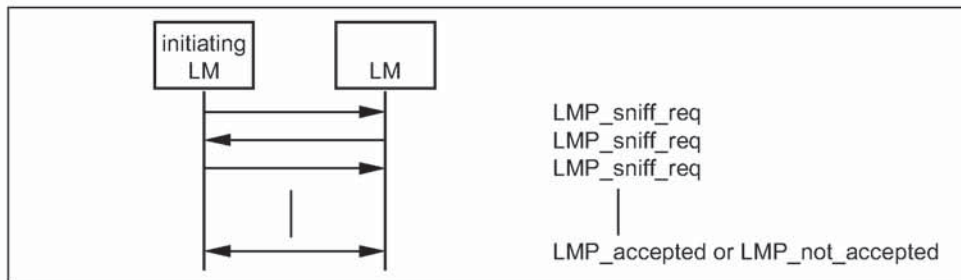
3.16.1 Master forces a slave into sniff mode



Sequence 30: Master forces slave into sniff mode.

3.16.2 Master or slave requests sniff mode

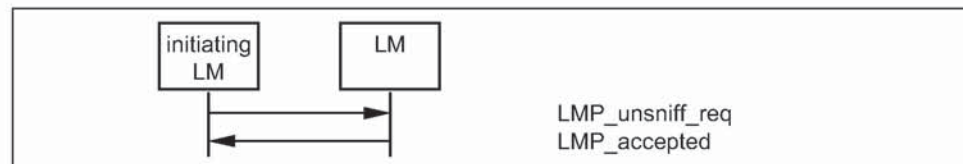
The master or the slave can request to enter sniff mode. Upon receipt of the request, the same request with modified parameters can be returned or the negotiation can be terminated. If an agreement is seen LMP_accepted terminates the negotiation and the ACL link is placed in sniff mode. If no agreement is seen, LMP_not_accepted with the reason code *unsupported parameter value* terminates the negotiation and sniff mode is not entered.



Sequence 31: Negotiation for sniff mode.

3.16.3 Moving a slave from sniff mode to active mode

Sniff mode is ended by sending the PDU LMP_unsniff_req. The requested device must reply with LMP_accepted. If the slave requests it will enter active mode after receiving LMP_accepted. If the master requests, the slave will enter active mode after receiving LMP_unsniff_req.



Sequence 32: Slave moved from sniff mode to active mode.

3.17 PARK MODE

If a slave does not need to participate in the channel, but still should be FH-synchronized, it can be placed in park mode. In this mode the device gives up its AM_ADDR but still re-synchronizes to the channel by waking up at the beacon instants separated by the beacon interval. The beacon interval, a beacon offset and a flag indicating how the first beacon instant is calculated determine the first beacon instant. After this the beacon instants follow periodically at the predetermined beacon interval. At the beacon instant the parked slave can be activated again by the master, the master can change the park mode parameters, transmit broadcast information or let the parked slaves request access to the channel.

All PDUs sent from the master to the parked slaves are broadcast. These PDUs (LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_addr_req and LMP_unpark_PM_addr_req) are the only PDUs that can be sent to a slave in park mode and the only PDUs that can be broadcast. To increase reliability for broadcast, the packets are made as short as possible. Therefore the format for these LMP PDUs are somewhat different. The parameters are not always byte-aligned and the length of the PDUs is variable.

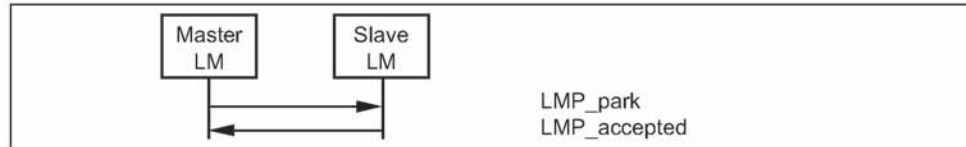
The messages for controlling the park mode include many parameters, which are all defined in Baseband Specification Section 10.8.4, on page 115. When a slave is placed in park mode it is assigned a unique PM_ADDR, which can be used by the master to unpark that slave. The all-zero PM_ADDR has a special meaning; it is not a valid PM_ADDR. If a device is assigned this PM_ADDR, it must be identified with its BD_ADDR when it is unparked by the master.

M/O	PDU	Contents
O	LMP_park_req	-
O	LMP_park	timing control flags D_B T_B N_B Δ_B PM_ADDR AR_ADDR $N_{B\text{sleep}}$ $D_{B\text{sleep}}$ D_{access} T_{access} $N_{\text{acc-slots}}$ N_{poll} M_{access} access scheme
O	LMP_set_broadcast_scan_window	timing control flags D_B (optional) broadcast scan window
O	LMP_modify_beacon	timing control flags D_B (optional) T_B N_B Δ_B D_{access} T_{access} $N_{\text{acc-slots}}$ N_{poll} M_{access} access scheme
O	LMP_unpark_PM_ADDR_req	timing control flags D_B (optional) AM_ADDR PM_ADDR AM_ADDR (optional) PM_ADDR (optional) (totally 1-7 pairs of AM_ADDR, PM_ADDR)
O	LMP_unpark_BD_ADDR_req	timing control flags D_B (optional) AM_ADDR BD_ADDR AM_ADDR (optional) BD_ADDR (optional)

Table 3.17: PDUs used for park mode.

3.17.1 Master forces a slave into park mode

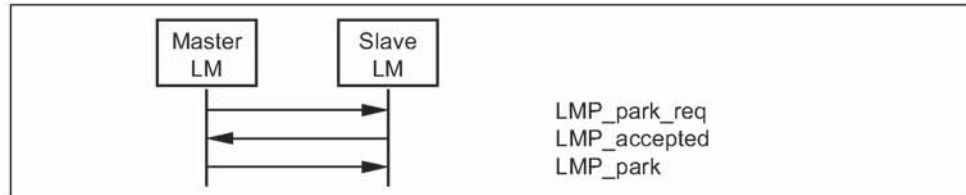
The master can force park mode. The master finalizes the transmission of the current L2CAP message and then sends LMP_park. When this PDU is received by the slave, it finalizes the transmission of the current L2CAP message and then sends LMP_accepted.



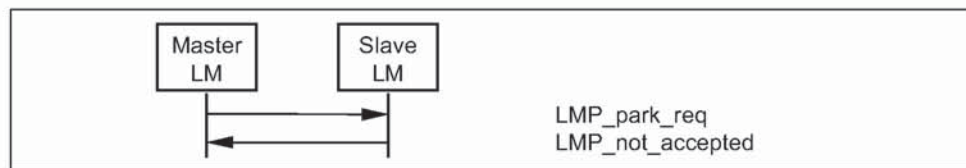
Sequence 33: Slave forced into park mode.

3.17.2 Master requests slave to enter park mode

The master can request park mode. The master finalizes the transmission of the current L2CAP message and then sends LMP_park_req. If the slave accepts to enter park mode it finalizes the transmission of the current L2CAP message and then responds with LMP_accepted. Finally the master sends LMP_park. If the slave rejects park mode it sends LMP_not_accepted.



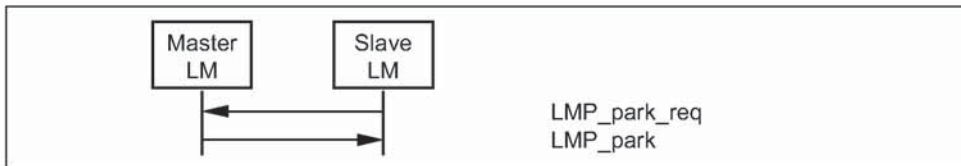
Sequence 34: Slave accepts to be placed in park mode.



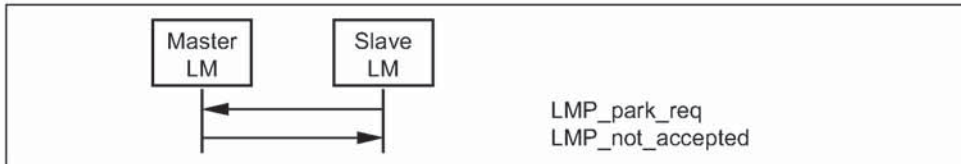
Sequence 35: Slave rejects to be placed in park mode.

3.17.3 Slave requests to be placed in park mode

The slave can request park mode. The slave finalizes the transmission of the current L2CAP message and then sends LMP_park_req. If the master accepts park mode it finalizes the transmission of the current L2CAP message and then sends LMP_park. If the master rejects park mode it sends LMP_not_accepted.



Sequence 36: Master accepts and places slave in park mode.



Sequence 37: Master rejects to place slave in park mode.

3.17.4 Master sets up broadcast scan window

If more broadcast capacity is needed than the beacon train, the master can indicate to the slaves that more broadcast information will follow the beacon train by sending LMP_set_broadcast_scan_window. This message is always sent in a broadcast packet at the beacon slot(s). The scan window starts in the beacon instant and is only valid for the current beacon.



Sequence 38: Master notifies all slaves of increase in broadcast capacity.

3.17.5 Master modifies beacon parameters

When the beacon parameters change the master notifies the parked slaves of this by sending LMP_modify_beacon. This message is always sent in a broadcast packet at the beacon slot(s).



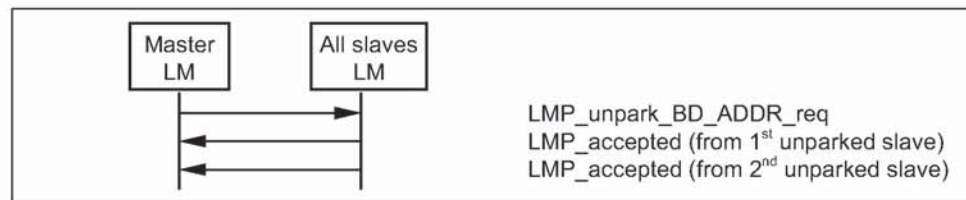
Sequence 39: Master modifies beacon parameters.

3.17.6 Unparking slaves

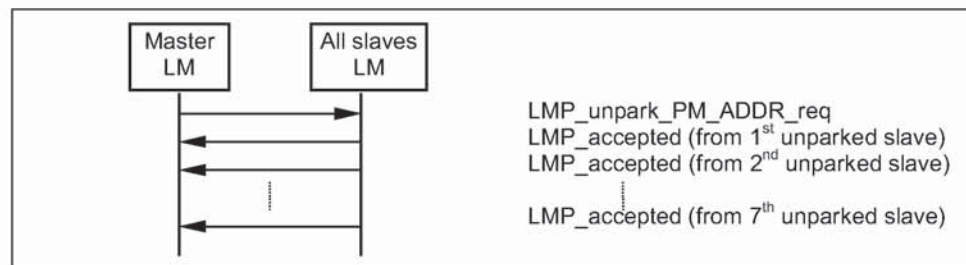
The master can unpark one or many slaves by sending a broadcast LMP message including the PM_ADDR or the BD_ADDR of the device(s) it wishes to

unpark at the beacon slot(s). This message also includes the AM_ADDR that the master assigns to the slave(s). After sending this message, the master must check the success of the unpark by polling each unparked slave, i.e. sending POLL packets, so that the slave is granted access to the channel. The unparked slave must then send a response with LMP_accepted. If this message is not received from the slave within a certain time after the master sent the unpark message, the unpark failed and the master must consider the slave as still being in park mode.

One message is used where the parked device is identified with the PM_ADDR, and another message is used where it is identified with the BD_ADDR. Both messages have variable length depending on the number of slaves the master unparks. For each slave the master wishes to unpark an AM_ADDR followed by the PM/BD_ADDR of the device that is assigned this AM_ADDR is included in the payload. If the slaves are identified with the PM_ADDR a maximum of 7 slaves can be unparked with the same message. If they are identified with the BD_ADDR a maximum of 2 slaves can be unparked with the same message.



Sequence 40: Master unparks slaves addressed with their BD_ADDR.



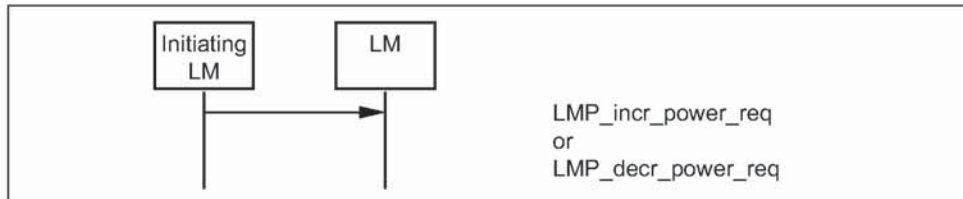
Sequence 41: Master unparks slaves addressed with their PM_ADDR.

3.18 POWER CONTROL

If the RSSI value differs too much from the preferred value of a Bluetooth device, it can request an increase or a decrease of the other device's TX power. Upon receipt of this message, the output power is increased or decreased one step. See Radio Specification Section 3.1, on page 21 for the definition of the step size. At the master side the TX power is completely independent for different slaves; a request from one slave can only effect the master's TX power for that same slave.

M/O	PDU	Contents
O	LMP_incr_power_req	for future use (1 Byte)
O	LMP_decr_power_req	for future use (1 Byte)
O	LMP_max_power	-
O	LMP_min_power	-

Table 3.18: PDUs used for power control.

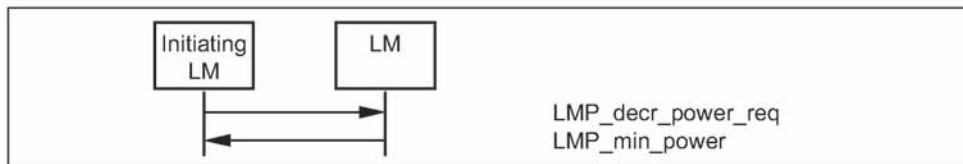


Sequence 42: A device requests a change of the other device's TX power.

If the receiver of LMP_incr_power_req already transmits at maximum power LMP_max_power is returned. The device may then only request an increase again after having requested a decrease at least once. Similarly, if the receiver of LMP_decr_power_req already transmits at minimum power then LMP_min_power is returned and the device may only request a decrease again after having requested an increase at least once.



Sequence 43: The TX power cannot be increased.



Sequence 44: The TX power cannot be decreased.

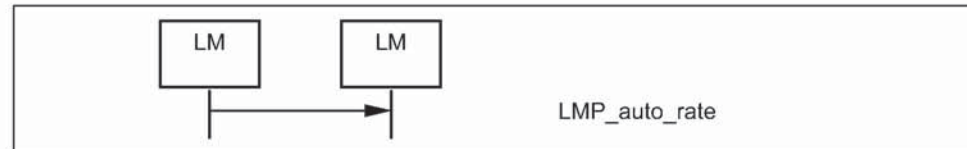
One byte is reserved in LMP_incr/decr_power_req for future use. It could, for example, be the mismatch between preferred and measured RSSI. The receiver of LMP_incr/decr_power_req could then use this value to adjust to the correct power at once, instead of only changing it one step for each request. The parameter value must be 0x00 for all versions of LMP where this parameter is not yet defined.

3.19 CHANNEL QUALITY-DRIVEN CHANGE BETWEEN DM AND DH

A device is configured to always use DM packets or to always use DH packets or to automatically adjust its packet type according to the quality of the channel. Nevertheless, all devices are capable of transmitting either DM or DH packets. The difference between DM and DH is that the payload in a DM packet is protected with a 2/3 FEC code, whereas the payload of a DH is not protected with any FEC. If a device wants to automatically adjust between DM and DH it sends LMP_auto_rate to the other device. Based upon quality measures in LC, the device determines if throughput will be increased by a change of packet type. If so, LMP_preferred_rate is sent to the other device. The PDUs used for this are:

M/O	PDU	Contents
O	LMP_auto_rate	-
O	LMP_preferred_rate	data rate

Table 3.19: PDUs used for quality driven change of the data rate.



Sequence 45: The left-hand unit is configured to automatically change between DM and DH.



Sequence 46: The right-hand device orders the left-hand device to change data rate.

3.20 QUALITY OF SERVICE (QoS)

The Link Manager provides Quality of Service capabilities. A poll interval, which is defined as the maximum time between subsequent transmissions from the master to a particular slave, is used to support bandwidth allocation and latency control. The poll interval is guaranteed except when there are collisions with page, page scan, inquiry and inquiry scan.

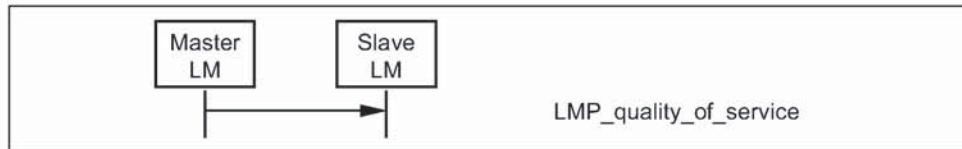
In addition, master and slave negotiate the number of repetitions for broadcast packets (NBC), see Baseband Specification Section 5.3, on page 68.

M/O	PDU	Contents
M	LMP_quality_of_service	poll interval N _{BC}
M	LMP_quality_of_service_req	poll interval N _{BC}

Table 3.20: PDUs used for quality of service.

3.20.1 Master notifies slave of the quality of service

In this case the master notifies the slave of the new poll interval and N_{BC}. The slave cannot reject the notification.



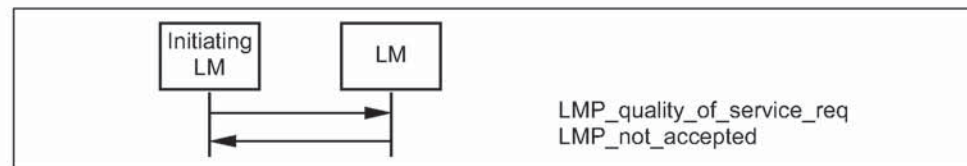
Sequence 47: Master notifies slave of new quality of service.

3.20.2 Device requests new quality of service

In this case the master or slave requests a new poll interval and N_{BC} . The parameter N_{BC} is meaningful only when it is sent by a master to a slave. For transmission of LMP_quality_of_service_req PDUs from a slave, this parameter is ignored by the master. The request can be accepted or rejected. This will allow the master and slave to dynamically negotiate the quality of service as needed.



Sequence 48: Device accepts new quality of service



Sequence 49: Device rejects new quality of service.

3.21 SCO LINKS

When a connection has been established between two Bluetooth devices the connection consists of an ACL link. One or more SCO links can then be established. The SCO link reserves slots separated by the SCO interval, T_{SCO} . The first slot reserved for the SCO link is defined by T_{SCO} and the SCO delay, D_{SCO} . After that the SCO slots follows periodically with the SCO interval. To avoid problems with a wrap-around of the clock during initialization of the SCO link, a flag indicating how the first SCO slot should be calculated is included in a message from the master. Note: Only bit0 and bit1 of this field is valid. Each SCO link is distinguished from all other SCO links by an SCO handle. The SCO handle zero is never used.

M/O	PDU	Contents
O	LMP_SCO_link_req	SCO handle timing control flags D_{SCO} T_{SCO} SCO packet air mode
O	LMP_remove_SCO_link_req	SCO handle reason

Table 3.21: PDUs used for managing the SCO links.

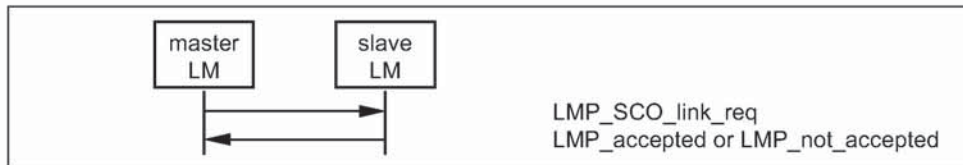
3.21.1 Master initiates an SCO link

When establishing an SCO link the master sends a request with parameters that specify the timing, packet type and coding that will be used on the SCO link. For each of the SCO packets Bluetooth supports three different voice coding formats on the air-interface: μ -law log PCM, A-law log PCM and CVSD.

The slots used for the SCO links are determined by three parameters controlled by the master: T_{SCO} , D_{SCO} and a flag indicating how the first SCO slot should be calculated. After the first slot, the SCO slots follows periodically with the T_{SCO} .

If the slave does not accept the SCO link, but is willing to consider another possible set of SCO parameters, it can indicate what it does not accept in the error reason field of LMP_not_accepted. The master then has the possibility to issue a new request with modified parameters.

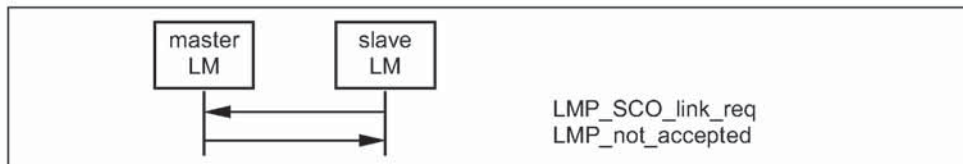
The SCO handle in the message must be different from any already existing SCO link(s).



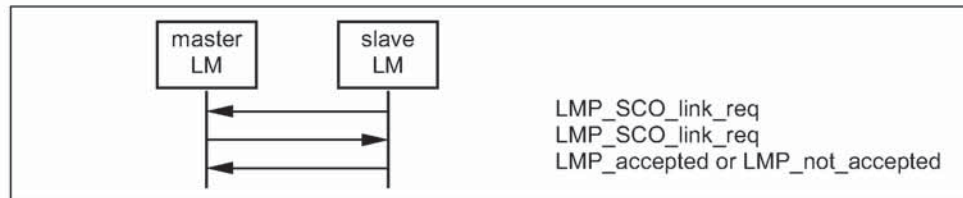
Sequence 50: Master requests an SCO link.

3.21.2 Slave initiates an SCO link

The slave can also initiate the establishment of an SCO link. The slave sends LMP_SCO_link_req, but the parameters timing control flags and D_{SCO} are invalid as well as the SCO handle, which must be zero. If the master is not capable of establishing an SCO link, it replies with LMP_not_accepted. Otherwise it sends back LMP_SCO_link_req. This message includes the assigned SCO handle, D_{SCO} and the timing control flags. For the other parameters, the master should try to use the same parameters as in the slave request; if the master cannot meet that request, it is allowed to use other values. The slave must then reply with LMP_accepted or LMP_not_accepted.



Sequence 51: Master rejects slave's request for an SCO link.



Sequence 52: Master accepts slave's request for an SCO link.

3.21.3 Master requests change of SCO parameters

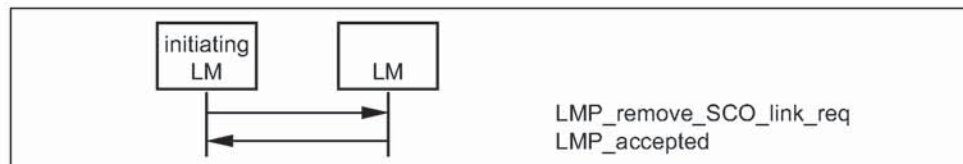
The master sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the master wishes to change parameters for. If the slave accepts the new parameters, it replies with LMP_accepted and the SCO link will change to the new parameters. If the slave does not accept the new parameters, it replies with LMP_not_accepted and the SCO link is left unchanged. When the slave replies with LMP_not_accepted it shall indicate in the error reason parameter what it does not accept. The master can then try to change the SCO link again with modified parameters. The sequence is the same as in Section 3.21.1 on page 220.

3.21.4 Slave requests change of SCO parameters

The slave sends LMP_SCO_link_req, where the SCO handle is the handle of the SCO link the slave wishes to change parameters for. The parameters timing control flags and D_{SCO} are not valid in this message. If the master does not accept the new parameters it replies with LMP_not_accepted and the SCO link is left unchanged. If the master accepts the new parameters it replies with LMP_SCO_link_req, where it must use the same parameters as in the slave request. When receiving this message the slave replies with LMP_accepted if it does not accept the new parameters. The SCO link is then left unchanged. If the slave accepts the new parameters it replies with LMP_accepted and the SCO link will change to the new parameters. The sequence is the same as in Section 3.21.2 on page 220.

3.21.5 Remove an SCO link

Master or slave can remove the SCO link by sending a request including the SCO handle of the SCO link to be removed and a reason indicating why the SCO link is removed. The receiving party must respond with LMP_accepted.



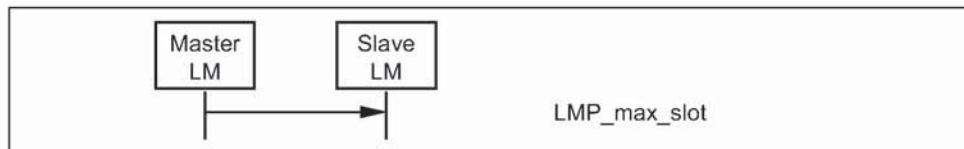
Sequence 53: SCO link removed.

3.22 CONTROL OF MULTI-SLOT PACKETS

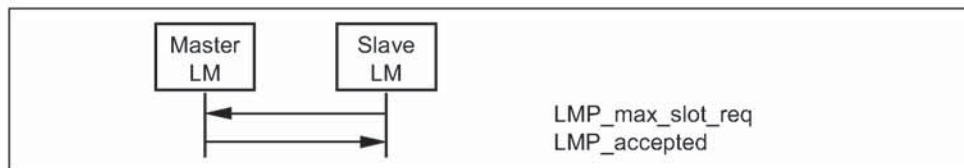
The number of slots used by a slave in its return packet can be limited. The master allows the slave to use a maximal number of slots by sending the PDU LMP_max_slots providing max slots as parameter. Each slave can request to use a maximal number of slots by sending the PDU LMP_max_slot_req providing max slots as parameter. The default value is 1 slot, i.e. if the slave has not been informed about the number of slots, it may only use 1-slot packets. Two PDUs are used for the control of multi-slot packets.

M/O	PDU	Contents
M	LMP_max_slot	max slots
M	LMP_max_slot_req	max slots

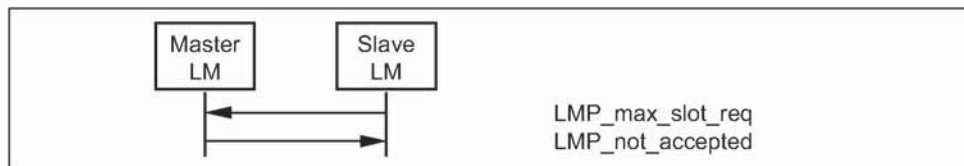
Table 3.22: PDUs used to control the use of multi-slot packets.



Sequence 54: Master allows slave to use a maximal number of slots.



Sequence 55: Slave requests to use a maximal number of slots. Master accepts.



Sequence 56: Slave requests to use a maximal number of slots. Master rejects.

3.23 PAGING SCHEME

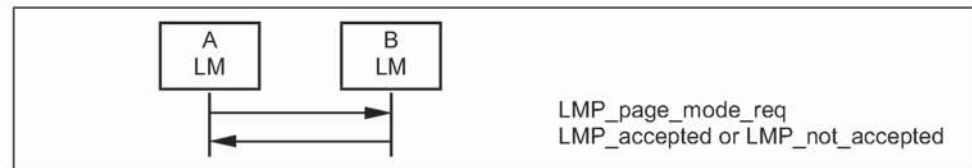
In addition to the mandatory paging scheme, Bluetooth defines optional paging schemes; see "Appendix VII" on page 999. LMP provides a means to negotiate the paging scheme, which is to be used the next time a unit is paged.

M/O	PDU	Contents
O	LMP_page_mode_req	paging scheme paging scheme settings
O	LMP_page_scan_mode_req	paging scheme paging scheme settings

Table 3.23: PDUs used to request paging scheme.

3.23.1 Page mode

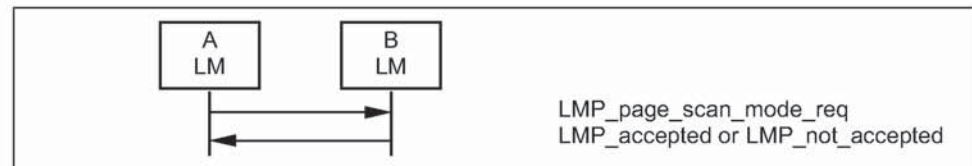
This procedure is initiated from device A and negotiates the paging scheme used when device A pages device B. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch back to the mandatory scheme may be rejected.



Sequence 57: Negotiation for page mode.

3.23.2 Page scan mode

This procedure is initiated from device A and negotiates the paging scheme used when device B pages device A. Device A proposes a paging scheme including the parameters for this scheme and device B can accept or reject. On rejection the old setting is not changed. A request to switch to the mandatory scheme must be accepted.



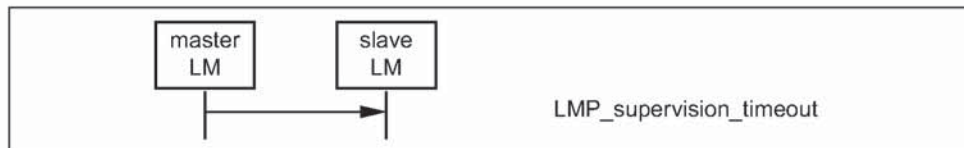
Sequence 58: Negotiation for page scan mode

3.24 LINK SUPERVISION

Each Bluetooth link has a timer that is used for link supervision. This timer is used to detect link loss caused by devices moving out of range, a device's power-down, or other similar failure cases. The scheme for link supervision is described in Baseband Specification Section 10.11, on page 126. An LMP procedure is used to set the value of the supervision timeout.

M/O	PDU	Contents
M	LMP_supervision_timeout	supervision timeout

Table 3.24: PDU used to set the supervision timeout.



Sequence 59: Setting the link supervision timeout.

4 CONNECTION ESTABLISHMENT

After the paging procedure, the master must poll the slave by sending POLL or NULL packets, with a max poll interval as defined in Table 5.5 on page 236. LMP procedures that do not require any interactions between the LM and the host at the paged unit's side can then be carried out.

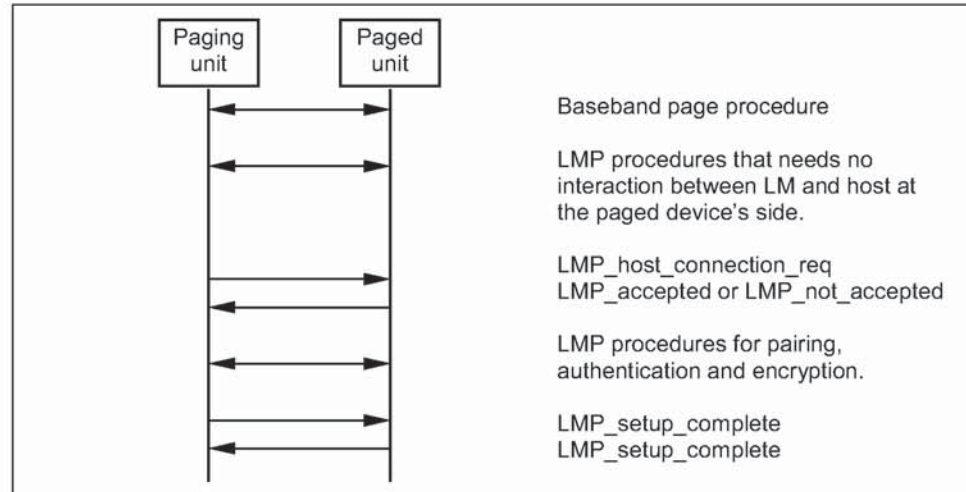


Figure 4.1: Connection establishment.

When the paging device wishes to create a connection involving layers above LM, it sends LMP_host_connection_req. When the other side receives this message, the host is informed about the incoming connection. The remote device can accept or reject the connection request by sending LMP_accepted or LMP_not_accepted.

When a device does not require any further link set-up procedures, it will send LMP_setup_complete. The device will still respond to requests from the other device. When the other device is also ready with link set-up, it will send LMP_setup_complete. After this, the first packet on a logical channel different from LMP can then be transmitted.

M/O	PDU	Contents
M	LMP_host_connection_req	-
M	LMP_setup_complete	-

Table 4.1: PDUs used for connection establishment.

5 SUMMARY OF PDUs

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_accepted	2	3	DM1/DV	m ↔ s	op code	2
LMP_au_rand	17	11	DM1	m ↔ s	random number	2-17
LMP_auto_rate	1	35	DM1/DV	m ↔ s	-	
LMP_clkoffset_req	1	5	DM1/DV	m → s	-	
LMP_clkoffset_res	3	6	DM1/DV	m ← s	clock offset	2-3
LMP_comb_key	17	9	DM1	m ↔ s	random number	2-17
LMP_decr_power_req	2	32	DM1/DV	m ↔ s	for future use	2
LMP_detach	2	7	DM1/DV	m ↔ s	reason	2
LMP_encryption_key_size_req	2	16	DM1/DV	m ↔ s	key size	2
LMP_encryption_mode_req	2	15	DM1/DV	m ↔ s	encryption mode	2
LMP_features_req	9	39	DM1/DV	m ↔ s	features	2-9
LMP_features_res	9	40	DM1/DV	m ↔ s	features	2-9
LMP_host_connection_req	1	51	DM1/DV	m ↔ s	-	
LMP_hold	3	20	DM1/DV	m ↔ s	hold time	2-3
LMP_hold_req	3	21	DM1/DV	m ↔ s	hold time	2-3
LMP_incr_power_req	2	31	DM1/DV	m ↔ s	for future use	2
LMP_in_rand	17	8	DM1	m ↔ s	random number	2-17
LMP_max_power	1	33	DM1/DV	m ↔ s	-	

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_max_slot	2	45	DM1/DV	m → s	max slots	2
LMP_max_slot_req	2	46	DM1/DV	m ← s	max slots	2
LMP_min_power	1	34	DM1/DV	m ↔ s	-	
LMP_modify_beacon	11 or 13	28	DM1	m → s	timing control flags	2
					D _B	3-4
					T _B	5-6
					N _B	7
					Δ _B	8
					D _{access}	9
					T _{access}	10
					N _{acc-slots}	11
					N _{poll}	12
					M _{access}	13:0-3
access scheme	13:4-7					
LMP_name_req	2	1	DM1/DV	m ↔ s	name offset	2
LMP_name_res	17	2	DM1	m ↔ s	name offset	2
					name length	3
					name fragment	4-17
LMP_not_accepted	3	4	DM1/DV	m ↔ s	op code	2
					reason	3
LMP_page_mode_req	3	53	DM1/DV	m ↔ s	paging scheme	2
					paging scheme settings	3
LMP_page_scan_mode_req	3	54	DM1/DV	m ↔ s	paging scheme	2
					paging scheme settings	3

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_park	17	26	DM	m → s	timing control flags D_B T_B N_B Δ_B PM_ADDR AR_ADDR $N_{B\text{sleep}}$ $D_{B\text{sleep}}$ D_{access} T_{access} $N_{\text{acc-slots}}$ N_{poll} M_{access} access scheme	2 3-4 5-6 7 8 9 10 11 12 13 14 15 16 17:0-3 17:4-7
LMP_park_req	1	25	DM1/DV	m ↔ s	-	
LMP_preferred_rate	2	36	DM1/DV	m ↔ s	data rate	2
LMP_quality_of_service	4	41	DM1/DV	m → s	poll interval N_{BC}	2-3 4
LMP_quality_of_service_req	4	42	DM1/DV	m ↔ s	poll interval N_{BC}	2-3 4
LMP_remove_SCO_link_req	3	44	DM1/DV	m ↔ s	SCO handle reason	2 3

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_SCO_link_req	7	43	DM1/DV	m ↔ s	SCO handle timing control flags D _{sco} T _{sco} SCO packet air mode	2 3 4 5 6 7
LMP_set_broadcast_scan_window	4 or 6	27	DM1	m → s	timing control flags D _B broadcast scan window	2 3-4 5-6
LMP_setup_complete	1	49	DM1	m ↔ s	-	
LMP_slot_offset	9	52	DM1/DV	m ↔ s	slot offset BD_ADDR	2-3 4-9
LMP_sniff	10	22	DM1	m → s	timing control flags D _{sniff} T _{sniff} sniff attempt sniff timeout	2 3-4 5-6 7-8 9-10
LMP_sniff_req	10	23	DM1	m ↔ s	timing control flags D _{sniff} T _{sniff} sniff attempt sniff timeout	2 3-4 5-6 7-8 9-10
LMP_sres	5	12	DM1/DV	m ↔ s	authentication response	2-5
LMP_start_encryption_req	17	17	DM1	m → s	random number	2-17
LMP_stop_encryption_req	1	18	DM1/DV	m → s	-	
LMP_supervision_timeout	3	55	DM1/DV	m ↔ s	supervision timeout	2-3
LMP_switch_req	1	19	DM1/DV	m ↔ s	-	

Table 5.1: Coding of the different LM PDUs.

LMP PDU	Length (bytes)	op code	Packet type	Possible direction	Contents	Position in payload
LMP_temp_rand	17	13	DM1	m → s	random number	2-17
LMP_temp_key	17	14	DM1	m → s	key	2-17
LMP_timing_accuracy_req	1	47	DM1/ DV	m ↔ s	-	
LMP_timing_accuracy_res	3	48	DM1/ DV	m ↔ s	drift	2
					jitter	3
LMP_unit_key	17	10	DM1	m ↔ s	key	2-17
LMP_unpark_BD_ADDR_req	variable	29	DM1	m → s	timing control flags	2
					D _B	3-4
					AM_ADDR 1 st unpark	5:0-3
					AM_ADDR 2 nd unpark	5:4-7
					BD_ADDR 1 st unpark	6-11
LMP_unpark_PM_ADDR_req	variable	30	DM1	m → s	timing control flags	2
					D _B	3-4
					AM_ADDR 1 st unpark	5:0-3
					AM_ADDR 2 nd unpark	5:4-7
					PM_ADDR 1 st unpark	6
LMP_unsniff_req	1	24	DM1/ DV	m ↔ s	-	
LMP_use_semi_permanent_key	1	50	DM1/ DV	m → s	-	
LMP_version_req	6	37	DM1/ DV	m ↔ s	VersNr	2
					Compld	3-4
					SubVersNr	5-6
LMP_version_res	6	38	DM1/ DV	m ↔ s	VersNr	2
					Compld	3-4
					SubVersNr	5-6

Table 5.1: Coding of the different LM PDUs.

Note1: For LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_ADDR_req and LMP_unpark_PM_ADDR_req the parameter

D_B is optional. This parameter is only present if bit0 of *timing control flags* is 0. If the parameter is not included, the position in payload for all parameters following D_B are decreased by 2.

Note2: For LMP_unpark_BD_ADDR the AM_ADDR and the BD_ADDR of the 2nd unparked slave are optional. If only one slave is unparked AM_ADDR 2nd unpark should be zero and BD_ADDR 2nd unpark is left out.

Note3: For LMP_unpark_PM_ADDR the AM_ADDR and the PM_ADDR of the 2nd – 7th unparked slaves are optional. If N slaves are unparked, the fields up to and including the Nth unparked slave are present. If N is odd, the AM_ADDR (N+1)th unpark must be zero. The length of the message is $x + 3N/2$ if N is even and $x + 3(N+1)/2 - 1$ if N is odd, where $x = 2$ or 4 depending on if the D_B is included Or Not (See Note1).

5.1 DESCRIPTION OF PARAMETERS

Name	Length (bytes)	Type	Unit	Detailed
access scheme	1	u_int4		0: polling technique 1-15: Reserved
air mode	1	u_int8		0: μ -law log 1: A-law log 2: CVSD 3-255: Reserved
AM_ADDR	1	u_int4		
AR_ADDR	1	u_int8		
authentication response	4	multiple bytes		
BD_ADDR	6	multiple bytes		
broadcast scan window	2	u_int16	slots	
clock offset	2	u_int16	1.25ms	(CLKN ₁₆₋₂ slave - CLKN ₁₆₋₂ master) mod 2 ¹⁵ MSbit of second byte not used.
Compld	2	u_int16		see BT Assigned Numbers Section 2.1 on page 1018
D_{access}	1	u_int8	slots	
D_B	2	u_int16	slots	

Table 5.2: Parameters in LM PDUs.

Name	Length (bytes)	Type	Unit	Detailed
D _{Bsleep}	1	u_int8	slots	
data rate	1	u_int8		0: medium rate 1: high rate 2-255: Reserved
drift	1	u_int8	ppm	
D _{SCO}	1	u_int8	slots	
D _{Sniff}	2	u_int16	slots	
encryption mode	1	u_int8		0: no encryption 1: point-to-point encryption 2: point-to-point and broadcast encryption 3 -255: Reserved
features	8	multiple bytes		See Table 5.3 on page 234
hold time	2	u_int16	slots	
jitter	1	u_int8	µs	
key	16	multiple bytes		
key size	1	u_int8	byte	
M _{access}	1	u_int4	slots	
max slots	1	u_int8	slots	
N _{acc-slots}	1	u_int8	slots	
name fragment	14	multiple bytes		UTF-8 characters.
name length	1	u_int8	bytes	
name offset	1	u_int8	bytes	
N _B	1	u_int8		
N _{BC}	1	u_int8		
N _{Bsleep}	1	u_int8	slots	
N _{poll}	1	u_int8	slots	
op code	1	u_int8		
paging scheme	1	u_int8		0: mandatory scheme 1: optional scheme 1 2-255: Reserved

Table 5.2: Parameters in LM PDUs.

Name	Length (bytes)	Type	Unit	Detailed
paging scheme settings	1	u_int8		For mandatory scheme: 0: R0 1: R1 2: R2 3-255: Reserved For optional scheme 1: 0: Reserved 1: R1 2: R2 3-255: Reserved
PM_ADDR	1	u_int8		
poll interval	2	u_int16	slots	
random number	16	multiple bytes		
reason	1	u_int8		See Table 5.4 on page 235.
SCO handle	1	u_int8		
SCO packet	1	u_int8		0: HV1 1: HV2 2: HV3 3-255: Reserved
slot offset	2	u_int16	μs	$0 \leq \text{slot offset} < 1250$
sniff attempt	2	u_int16	slots	
sniff timeout	2	u_int16	slots	
SubVersNr	2	u_int16		Defined by each company
supervision time-out	2	u_int16	slots	
T _{access}	1	u_int8	slots	
T _B	2	u_int16	slots	
timing control flags	1	u_int8		bit0 = 0: no timing change bit0 = 1: timing change bit1 = 0: use initialization 1 bit1 = 1: use initialization 2 bit2 = 0: access window bit2 = 1: no access window bit3-7: Reserved

Table 5.2: Parameters in LM PDUs.

Name	Length (bytes)	Type	Unit	Detailed
T _{sco}	1	u_int8	slots	
T _{sniff}	2	u_int16	slots	
VersNr	1	u_int8		0: Bluetooth LMP 1.0 1-255: Reserved
Δ _B	1	u_int8	slots	

Table 5.2: Parameters in LM PDUs.

5.1.1 Coding of features

This parameter is a bitmap with information about the Bluetooth radio-, base-band- and LMP features which a device supports. The bit shall be one if the feature is supported. The feature parameter bits that are not defined in Table 5.3 shall be zero.

Byte	Bit	Supported feature
0	0	3-slot packets
	1	5-slot packets
	2	encryption
	3	slot offset
	4	timing accuracy
	5	switch
	6	hold mode
	7	sniff mode
1	0	park mode
	1	RSSI
	2	channel quality driven data rate
	3	SCO link
	4	HV2 packets
	5	HV3 packets
	6	u-law log
	7	A-law log
2	0	CVSD
	1	paging scheme
	2	power control

Table 5.3: Coding of the parameter features.

5.1.2 List of error reasons

The following table contains the codes of the different error reasons used in LMP.

Reason	Description
0x05	Authentication Failure
0x06	Key Missing
0x0A	Max Number Of SCO Connections To A Device (The maximum number of SCO connections to a particle device has been reached. All allowed SCO connection handles to that device are used.)
0x0D	Host Rejected due to limited resources (The host at the remote side has rejected the connection because the remote host did not have enough additional resources to accept the connection.)
0x0E	Host Rejected due to security reasons (The host at the remote side has rejected the connection because the remote host determined that the local host did not meet its security criteria.)
0x0F	Host Rejected due to remote device is only a personal device (The host at the remote side has rejected the connection because the remote host is a personal device and will only accept the connection from one particle remote host.)
0x10	Host Timeout (Used at connection accept timeout, the host did not respond to an incoming connection attempt before the connection accept timer expired.)
0x13	Other End Terminated Connection: User Ended Connection
0x14	Other End Terminated Connection: Low Resources
0x15	Other End Terminated Connection: About to Power Off
0x16	Connection Terminated by Local Host
0x17	Repeated Attempts (An authentication or pairing attempt is made too soon after a previously failed authentication or pairing attempt.)
0x18	Pairing Not Allowed
0x19	Unknown LMP PDU
0x1A	Unsupported LMP Feature
0x1B	SCO Offset Rejected
0x1C	SCO Interval Rejected
0x1D	SCO Air Mode Rejected
0x1E	Invalid LMP Parameters
0x1F	Unspecified Error
0x20	Unsupported parameter value
0x21	Switch not allowed
0x23	LMP Error Transaction Collision
0x24	PDU not allowed

Table 5.4: List of error reasons.

5.2 DEFAULT VALUES

The Bluetooth device must use these values before anything else has been negotiated:

Parameter	Value
drift	250
jitter	10
max slots	1
poll interval	40

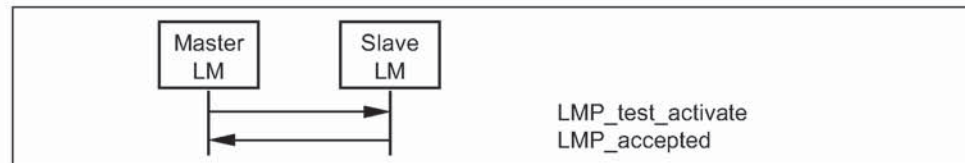
Table 5.5: Default values.

6 TEST MODES

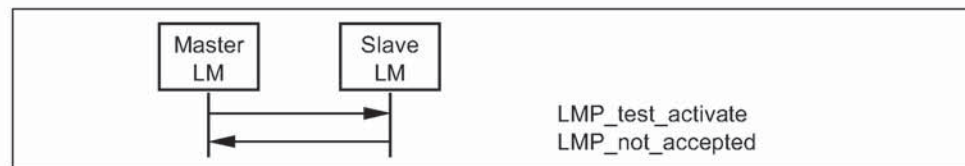
LMP has PDUs to support different Bluetooth test modes, which are used for certification and compliance testing of the Bluetooth radio and baseband. See “Bluetooth Test Mode” on page 803 for a detailed description of these test modes.

6.1 ACTIVATION AND DEACTIVATION OF TEST MODE

The test mode is activated by sending LMP_test_activate to the device under test (DUT). The DUT is always the slave. The link manager must be able to receive this message anytime. If entering test mode is locally enabled in the DUT it responds with LMP_accepted and test mode is entered. Otherwise the DUT responds with LMP_not_accepted and the DUT remains in normal operation. The reason code in LMP_not_accepted shall be *PDU not allowed*.



Sequence 60: Activation of test mode successful.

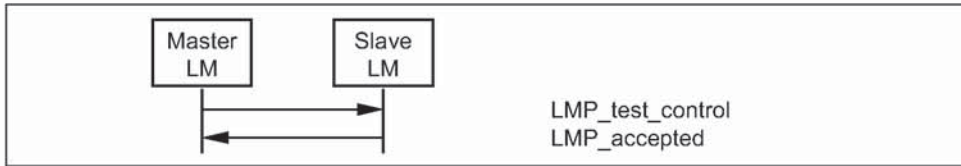


Sequence 61: Activation of test mode fails. Slave is not allowed to enter test mode.

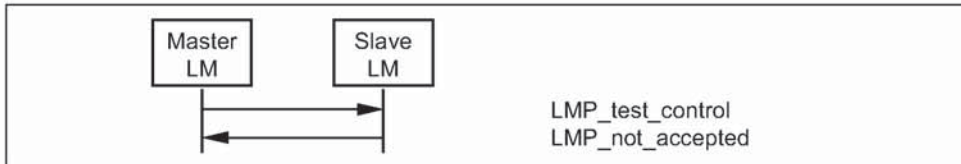
The test mode can be deactivated in two ways. Sending LMP_test_control with the test scenario set to "exit test mode" exits the test mode and the slave returns to normal operation still connected to the master. Sending LMP_detach to the DUT ends the test mode and the connection.

6.2 CONTROL OF TEST MODE

When the DUT has entered test mode, the PDU LMP_test_control can be sent to the DUT to start a specific test. This PDU is acknowledged with LMP_accepted. If a device that is not in test mode receives LMP_test_control it responds with LMP_not_accepted, where the reason code shall be *PDU not allowed*.



Sequence 62: Control of test mode successful.



Sequence 63: Control of test mode rejected since slave is not in test mode.

6.3 SUMMARY OF TEST MODE PDUs

The PDUs used for test purposes are summarized in the following table. For a detailed description of the parameters, see Bluetooth Test Mode Table 3.2 on page 817.

M/O	LMP PDU	Length	op code	Packet type	Possible direction	Contents	Position in payload
M	LMP_test_activate	1	56	DM1/DV	m → s	-	
M	LMP_test_control	10	57	DM1	m → s	test scenario hopping mode TX frequency RX frequency power control mode poll period packet type length of test data	2 3 4 5 6 7 8 9-10

Table 6.1: Test mode PDUs.

7 ERROR HANDLING

If the Link Manager receives a PDU with unrecognized opcode, it responds with `LMP_not_accepted` with the reason code *unknown LMP PDU*. The opcode parameter that is echoed back is the unrecognized opcode.

If the Link Manager receives a PDU with invalid parameters, it responds with `LMP_not_accepted` with the reason code *invalid LMP parameters*.

If the maximum response time, see Section 1 on page 191, is exceeded or if a link loss is detected (see Baseband Specification Section 10.11, on page 126), the party that waits for the response shall conclude that the procedure has terminated unsuccessfully.

Erroneous LMP messages can be caused by errors on the channel or systematic errors at the transmit side. To detect the latter case, the LM should monitor the number of erroneous messages and disconnect if it exceeds a threshold, which is implementation-dependent.

Since LMP PDUs are not interpreted in real time, collision situations can occur where both LMs initiate the same procedure and both cannot be completed. In this situation, the master shall reject the slave-initiated procedure by sending `LMP_not_accepted` with the reason code 'LMP Error Transaction Collision'. The master-initiated procedure shall then be completed.



8 LIST OF FIGURES

Figure 1.1:	Link Manager's place on the global scene.....	191
Figure 2.1:	Payload body when LM PDUs are sent.	192
Figure 3.1:	Symbols used in sequence diagrams.	193
Sequence 1:	Authentication. Claimant has link key.	194
Sequence 2:	Authentication fails. Claimant has no link key.	194
Sequence 3:	Claimant accepts pairing.	195
Sequence 4:	Claimant accepts pairing but requests to be verifier.	196
Sequence 5:	Unsuccessful switch of claimant-verifier role.	196
Sequence 6:	Claimant rejects pairing.	196
Sequence 7:	Creation of the link key.	197
Sequence 8:	Successful change of the link key.	197
Sequence 9:	Change of the link key not possible since the other unit uses a unit key.	198
Sequence 10:	Change to a temporary link key.	199
Sequence 11:	Link key changed to the semi-permanent link key.	199
Sequence 12:	Negotiation for encryption mode.	200
Sequence 13:	Encryption key size negotiation successful.	200
Sequence 14:	Encryption key size negotiation failed.	201
Sequence 15:	Start of encryption.	201
Sequence 16:	Stop of encryption.	202
Sequence 17:	Clock offset requested.	203
Sequence 18:	Slot offset information is sent.	203
Sequence 19:	The requested device supports timing accuracy information.	204
Sequence 20:	The requested device does not support timing accuracy information.	204
Sequence 21:	Request for LMP version.	205
Sequence 22:	Request for supported features.	206
Sequence 23:	Master-slave switch accepted.	206
Sequence 24:	Master-slave switch not accepted.	207
Sequence 25:	Device's name requested and it responds.	207
Sequence 26:	Connection closed by sending LMP_detach.	208
Sequence 27:	Master forces slave into hold mode.	208
Sequence 28:	Slave forces master into hold mode.	209
Sequence 29:	Negotiation for hold mode.	209
Sequence 30:	Master forces slave into sniff mode.	210
Sequence 31:	Negotiation for sniff mode.	210
Sequence 32:	Slave moved from sniff mode to active mode.	211
Sequence 33:	Slave forced into park mode.	213
Sequence 34:	Slave accepts to be placed in park mode.	213

Link Manager Protocol

Bluetooth.

Sequence 35: Slave rejects to be placed in park mode.213

Sequence 36: Master accepts and places slave in park mode.214

Sequence 37: Master rejects to place slave in park mode.214

Sequence 38: Master notifies all slaves of increase in broadcast capacity.214

Sequence 39: Master modifies beacon parameters.214

Sequence 40: Master un parks slaves addressed with their BD_ADDR. ...215

Sequence 41: Master un parks slaves addressed with their PM_ADDR. ...215

Sequence 42: A device requests a change of the other device's TX power.216

Sequence 43: The TX power cannot be increased.216

Sequence 44: The TX power cannot be decreased.216

Sequence 45: The left-hand unit is configured to automatically change between DM and DH.217

Sequence 46: The right-hand device orders the left-hand device to change data rate.217

Sequence 47: Master notifies slave of new quality of service.218

Sequence 48: Device accepts new quality of service219

Sequence 49: Device rejects new quality of service.219

Sequence 50: Master requests an SCO link.220

Sequence 51: Master rejects slave's request for an SCO link.220

Sequence 52: Master accepts slave's request for an SCO link.221

Sequence 53: SCO link removed.221

Sequence 54: Master allows slave to use a maximal number of slots.222

Sequence 55: Slave requests to use a maximal number of slots. Master accepts.222

Sequence 56: Slave requests to use a maximal number of slots. Master rejects.222

Sequence 57: Negotiation for page mode.223

Sequence 58: Negotiation for page scan mode223

Sequence 59: Setting the link supervision timeout.224

Figure 4.1: Connection establishment.225

Sequence 60: Activation of test mode successful.237

Sequence 61: Activation of test mode fails. Slave is not allowed to enter test mode.237

Sequence 62: Control of test mode successful.238

Sequence 63: Control of test mode rejected since slave is not in test mode.238

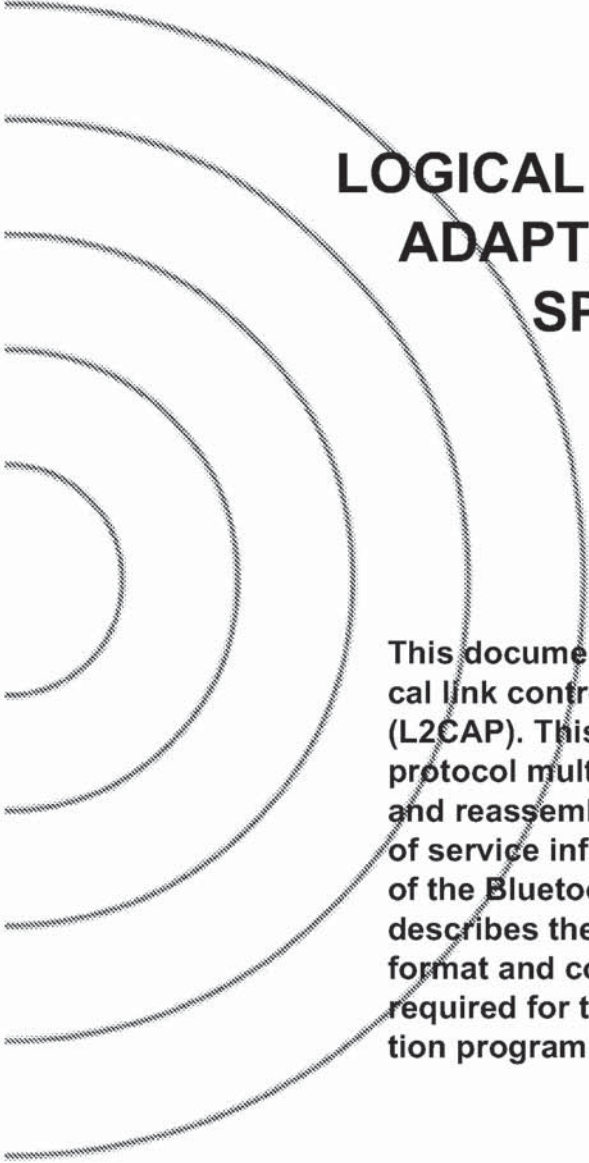
9 LIST OF TABLES

Table 2.1:	Logical channel L_CH field contents.....	192
Table 3.1:	General response messages.....	193
Table 3.2:	PDU used for authentication.....	194
Table 3.3:	PDU used for pairing.....	195
Table 3.4:	PDU used for change of link key.....	197
Table 3.5:	PDU used to change the current link key.....	198
Table 3.6:	PDU used for handling encryption.....	199
Table 3.7:	PDU used for clock offset request.....	202
Table 3.8:	PDU used for slot offset information.....	203
Table 3.9:	PDU used for requesting timing accuracy information.....	204
Table 3.10:	PDU used for LMP version request.....	205
Table 3.11:	PDU used for features request.....	206
Table 3.12:	PDU used for master slave switch.....	206
Table 3.13:	PDU used for name request.....	207
Table 3.14:	PDU used for detach.....	207
Table 3.15:	PDU used for hold mode.....	208
Table 3.16:	PDU used for sniff mode.....	210
Table 3.17:	PDU used for park mode.....	212
Table 3.18:	PDU used for power control.....	216
Table 3.19:	PDU used for quality driven change of the data rate.....	217
Table 3.20:	PDU used for quality of service.....	218
Table 3.21:	PDU used for managing the SCO links.....	219
Table 3.22:	PDU used to control the use of multi-slot packets.....	222
Table 3.23:	PDU used to request paging scheme.....	223
Table 3.24:	PDU used to set the supervision timeout.....	224
Table 4.1:	PDU used for connection establishment.....	225
Table 5.1:	Coding of the different LM PDUs.....	226
Table 5.2:	Parameters in LM PDUs.....	231
Table 5.3:	Coding of the parameter features.....	234
Table 5.4:	List of error reasons.....	235
Table 5.5:	Default values.....	236
Table 6.1:	Test mode PDUs.....	238



Part D

LOGICAL LINK CONTROL AND ADAPTATION PROTOCOL SPECIFICATION



This document describes the Bluetooth logical link control and adaptation protocol (L2CAP). This protocol supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. This document is part of the Bluetooth Specification. This document describes the protocol state machine, packet format and composition, and a test interface required for the Bluetooth test and certification program.

CONTENTS

1	Introduction	249
1.1	L2CAP Functional Requirements.....	250
1.2	Assumptions	252
1.3	Scope.....	252
2	General Operation.....	253
2.1	Channel Identifiers	253
2.2	Operation Between Devices.....	253
2.3	Operation Between Layers.....	254
2.4	Segmentation and Reassembly	255
2.4.1	Segmentation Procedures.....	256
2.4.2	Reassembly Procedures	256
3	State Machine	258
3.1	Events	259
3.1.1	Lower-Layer Protocol (LP) to L2CAP events	259
3.1.2	L2CAP to L2CAP Signalling events	260
3.1.3	L2CAP to L2CAP Data events	261
3.1.4	Upper-Layer to L2CAP events	261
3.1.5	Timer events.....	262
3.2	Actions	263
3.2.1	L2CAP to Lower Layer actions.....	263
3.2.2	L2CAP to L2CAP Signalling actions.....	264
3.2.3	L2CAP to L2CAP Data actions.....	264
3.2.4	L2CAP to Upper Layer actions.....	264
3.3	Channel Operational States	265
3.4	Mapping Events to Actions.....	266
4	Data Packet Format.....	272
4.1	Connection-oriented Channel	272
4.2	Connectionless Data Channel.....	273
5	Signalling	275
5.1	Command Reject (code 0x01)	277
5.2	Connection Request (code 0x02).....	278
5.3	Connection Response (code 0x03).....	279
5.4	Configuration Request (code 0x04)	280
5.5	Configure Response (code 0x05)	283
5.6	Disconnection Request (code 0x06).....	285
5.7	Disconnection Response (code 0x07)	286
5.8	Echo Request (code 0x08).....	286
5.9	Echo Response (code 0x09).....	287
5.10	Information Request (CODE 0x0A).....	287
5.11	Information Response (CODE 0x0B).....	288

6	Configuration Parameter Options	289
6.1	Maximum Transmission Unit (MTU)	289
6.2	Flush Timeout Option.....	290
6.3	Quality of Service (QoS) Option	291
6.4	Configuration Process	293
6.4.1	Request Path	293
6.4.2	Response Path	294
6.4.3	Configuration State Machine.....	294
7	Service Primitives	295
7.1	Event Indication	295
7.1.1	L2CA_ConnectInd Callback.....	296
7.1.2	L2CA_ConfigInd Callback.....	296
7.1.3	L2CA_DisconnectInd Callback.....	296
7.1.4	L2CA_QoSViolationInd Callback	296
7.2	Connect	296
7.3	Connect Response	298
7.4	Configure	299
7.5	Configuration Response	301
7.6	Disconnect	302
7.7	Write.....	303
7.8	Read	304
7.9	Group Create	305
7.10	Group Close	305
7.11	Group Add Member	306
7.12	Group Remove Member	307
7.13	Get Group Membership	308
7.14	Ping	309
7.15	GetInfo	310
7.16	Disable Connectionless Traffic	311
7.17	Enable Connectionless Traffic	312
8	Summary	313
9	References	314
10	List of Figures	315
11	List of Tables	316
	Terms and Abbreviations	317
	Appendix A: Configuration MSCs	318
	Appendix B: Implementation Guidelines	321

1 INTRODUCTION

This section of the Bluetooth Specification defines the Logical Link Control and Adaptation Layer Protocol, referred to as L2CAP. L2CAP is layered over the Baseband Protocol and resides in the data link layer as shown in Figure 1.1. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

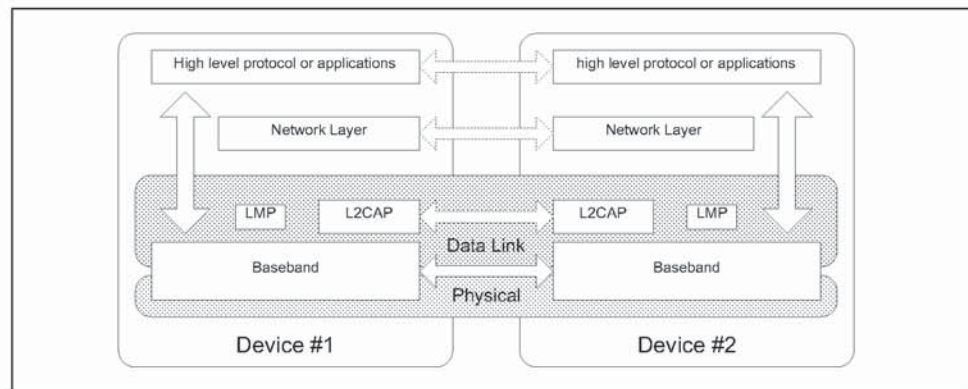


Figure 1.1: L2CAP within protocol layers

The “Baseband Specification” on page 33 defines two link types: Synchronous Connection-Oriented (SCO) links and Asynchronous Connection-Less (ACL) links. SCO links support real-time voice traffic using reserved bandwidth. ACL links support best effort traffic. The L2CAP Specification is defined for only ACL links and no support for SCO links is planned.

For ACL links, use of the AUX1 packet on the ACL link is prohibited. This packet type supports no data integrity checks (no CRC). Because L2CAP depends on integrity checks in the Baseband to protect the transmitted information, AUX1 packets must never be used to transport L2CAP packets.

The format of the ACL payload header is shown below. Figure 1.2 on page 250 displays the payload header used for single-slot packets and Figure 1.3 displays the header used in multi-slot packets. The only difference is the size of the length field. The packet type (a field in the Baseband header) distinguishes single-slot packets from multi-slot packets.



Figure 1.2: ACL Payload Header for single-slot packets

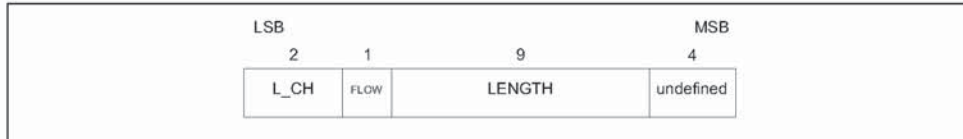


Figure 1.3: ACL Payload Header for multi-slot packets

The 2-bit logical channel (L_CH) field, defined in Table 1.1, distinguishes L2CAP packets from Link Manager Protocol (page 185) packets. The remaining code is reserved for future use.

L_CH code	Logical Channel	Information
00	RESERVED	Reserved for future use
01	L2CAP	Continuation of L2CAP packet
10	L2CAP	Start of L2CAP packet
11	LMP	Link Manager Protocol

Table 1.1: Logical channel L_CH field contents

The FLOW bit in the ACL header is managed by the Link Controller (LC), a Baseband implementation entity, and is normally set to 1 ('flow on'). It is set to 0 ('flow off') when no further L2CAP traffic shall be sent over the ACL link. Sending an L2CAP packet with the FLOW bit set to 1 resumes the flow of incoming L2CAP packets. This is described in more detail in "Baseband Specification" on page 33.

1.1 L2CAP FUNCTIONAL REQUIREMENTS

The functional requirements for L2CAP include protocol multiplexing, segmentation and reassembly (SAR), and group management. Figure 1.4 illustrates how L2CAP fits into the Bluetooth Protocol Stack. L2CAP lies above the Baseband Protocol (page 33) and interfaces with other communication protocols such as the Bluetooth Service Discovery Protocol (SDP, page 323), RFCOMM (page 385), and Telephony Control (TCS, page 429). Voice-quality channels for audio and telephony applications are usually run over Baseband SCO links. Packetized audio data, such as IP Telephony, may be sent using communication protocols running over L2CAP.

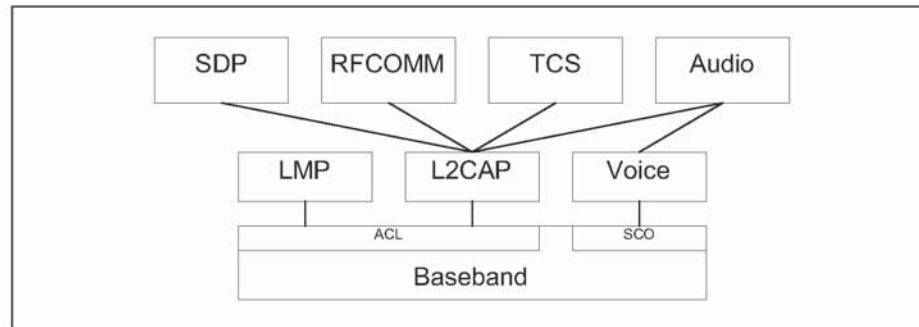


Figure 1.4: L2CAP in Bluetooth Protocol Architecture

Essential protocol requirements for L2CAP include simplicity and low overhead. Implementations of L2CAP must be applicable for devices with limited computational resources. L2CAP should not consume excessive power since that significantly sacrifices power efficiency achieved by the Bluetooth Radio. Memory requirements for protocol implementation should also be kept to a minimum.

The protocol complexity should be acceptable to personal computers, PDAs, digital cellular phones, wireless headsets, joysticks and other wireless devices supported by Bluetooth. Furthermore, the protocol should be designed to achieve reasonably high bandwidth efficiency.

- *Protocol Multiplexing*

L2CAP must support protocol multiplexing because the Baseband Protocol does not support any 'type' field identifying the higher layer protocol being multiplexed above it. L2CAP must be able to distinguish between upper layer protocols such as the Service Discovery Protocol (page 323), RFCOMM (page 385), and Telephony Control (page 429).

- *Segmentation and Reassembly*

Compared to other wired physical media, the data packets defined by the Baseband Protocol (page 33) are limited in size. Exporting a maximum transmission unit (MTU) associated with the largest Baseband payload (341 bytes for DH5 packets) limits the efficient use of bandwidth for higher layer protocols that are designed to use larger packets. Large L2CAP packets must be segmented into multiple smaller Baseband packets prior to their transmission over the air. Similarly, multiple received Baseband packets may be reassembled into a single larger L2CAP packet following a simple integrity check (described in Section 2.4.2 on page 256). The Segmentation and Reassembly (SAR) functionality is absolutely necessary to support protocols using packets larger than those supported by the Baseband.

- *Quality of Service*

The L2CAP connection establishment process allows the exchange of information regarding the quality of service (QoS) expected between two Blue-

tooth units. Each L2CAP implementation must monitor the resources used by the protocol and ensure that QoS contracts are honoured.

- *Groups*

Many protocols include the concept of a group of addresses. The Baseband Protocol supports the concept of a piconet, a group of devices synchronously hopping together using the same clock. The L2CAP group abstraction permits implementations to efficiently map protocol groups on to piconets. Without a group abstraction, higher level protocols would need to be exposed to the Baseband Protocol and Link Manager functionality in order to manage groups efficiently.

1.2 ASSUMPTIONS

The protocol is designed based on the following assumptions:

1. The ACL link between two units is set up using the Link Manager Protocol (page 185). The Baseband provides orderly delivery of data packets, although there might be individual packet corruption and duplicates. No more than 1 ACL link exists between any two devices.
2. The Baseband always provides the impression of full-duplex communication channels. This does not imply that all L2CAP communications are bi-directional. Multicasts and unidirectional traffic (e.g., video) do not require duplex channels.
3. L2CAP provides a reliable channel using the mechanisms available at the Baseband layer. The Baseband always performs data integrity checks when requested and resends data until it has been successfully acknowledged or a timeout occurs. Because acknowledgements may be lost, timeouts may occur even after the data has been successfully sent. The Baseband protocol uses a 1-bit sequence number that removes duplicates. Note that the use of Baseband broadcast packets is prohibited if reliability is required since all broadcasts start the first segment of an L2CAP packet with the same sequence bit.

1.3 SCOPE

The following features are outside the scope of L2CAP's responsibilities:

- L2CAP does not transport audio designated for SCO links.
- L2CAP does not enforce a reliable channel or ensure data integrity, that is, L2CAP performs no retransmissions or checksum calculations.
- L2CAP does not support a reliable multicast channel. See Section 4.2.
- L2CAP does not support the concept of a global group name.

2 GENERAL OPERATION

The Logical Link Control and Adaptation Protocol (L2CAP) is based around the concept of 'channels'. Each one of the end-points of an L2CAP channel is referred to by a *channel identifier*.

2.1 CHANNEL IDENTIFIERS

Channel identifiers (CIDs) are local names representing a logical channel end-point on the device. Identifiers from 0x0001 to 0x003F are reserved for specific L2CAP functions. The null identifier (0x0000) is defined as an illegal identifier and must never be used as a destination end-point. Implementations are free to manage the remaining CIDs in a manner best suited for that particular implementation, with the provision that the same CID is not reused as a local L2CAP channel endpoint for multiple simultaneous L2CAP channels between a local device and some remote device. Table 2.1 summarizes the definition and partitioning of the CID name space.

CID assignment is relative to a particular device and a device can assign CIDs independently from other devices (unless it needs to use any of the reserved CIDs shown in the table below). Thus, even if the same CID value has been assigned to (remote) channel endpoints by several remote devices connected to a single local device, the local device can still uniquely associate each remote CID with a different device.

CID	Description
0x0000	Null identifier
0x0001	Signalling channel
0x0002	Connectionless reception channel
0x0003-0x003F	Reserved
0x0040-0xFFFF	Dynamically allocated

Table 2.1: CID Definitions

2.2 OPERATION BETWEEN DEVICES

Figure 2.1 on page 254 illustrates the use of CIDs in a communication between corresponding peer L2CAP entities in separate devices. The connection-oriented data channels represent a connection between two devices, where a CID identifies each endpoint of the channel. The connectionless channels restrict data flow to a single direction. These channels are used to support a channel 'group' where the CID on the source represents one or more remote devices. There are also a number of CIDs reserved for special purposes. The signalling channel is one example of a reserved channel. This channel is used to create and establish connection-oriented data channels and to negotiate changes in the characteristics of these channels. Support for a signalling chan-

nel within an L2CAP entity is mandatory. Another CID is reserved for all incoming connectionless data traffic. In the example below, a CID is used to represent a group consisting of device #3 and #4. Traffic sent from this channel ID is directed to the remote channel reserved for connectionless data traffic.

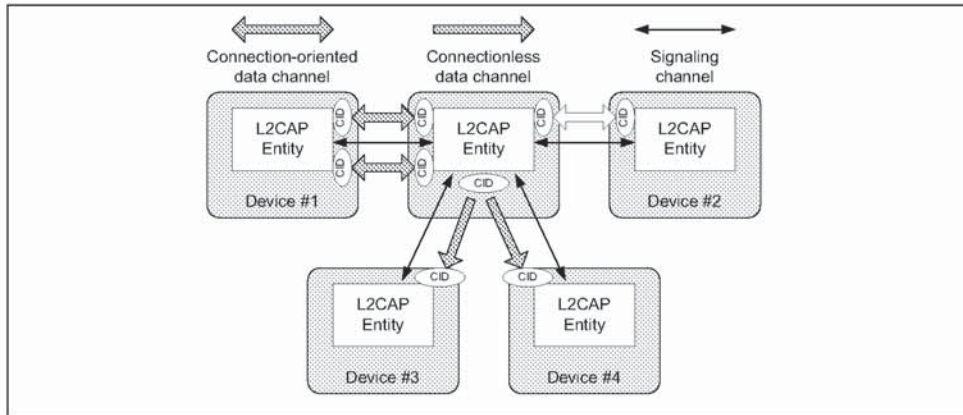


Figure 2.1: Channels between devices

Table 2.2 describes the various channels and their source and destination identifiers. An 'allocated' channel is created to represent the local endpoint and should be in the range 0x0040 to 0xFFFF. Section 3 on page 258 describes the state machine associated with each connectionless channel. Section 4.1 on page 272 describes the packet format associated with bi-directional channels and Section 4.2 on page 273 describes the packet format associated with uni-directional channels.

Channel Type	Local CID	Remote CID
Connection-oriented	Dynamically allocated	Dynamically allocated
Connectionless data	Dynamically allocated	0x0002 (fixed)
Signalling	0x0001 (fixed)	0x0001 (fixed)

Table 2.2: Types of Channel Identifiers

2.3 OPERATION BETWEEN LAYERS

L2CAP implementations should follow the general architecture described below. L2CAP implementations must transfer data between higher layer protocols and the lower layer protocol. This document lists a number of services that should be exported by any L2CAP implementation. Each implementation must also support a set of signalling commands for use between L2CAP implementations. L2CAP implementations should also be prepared to accept certain types of events from lower layers and generate events to upper layers. How these events are passed between layers is an implementation-dependent process.

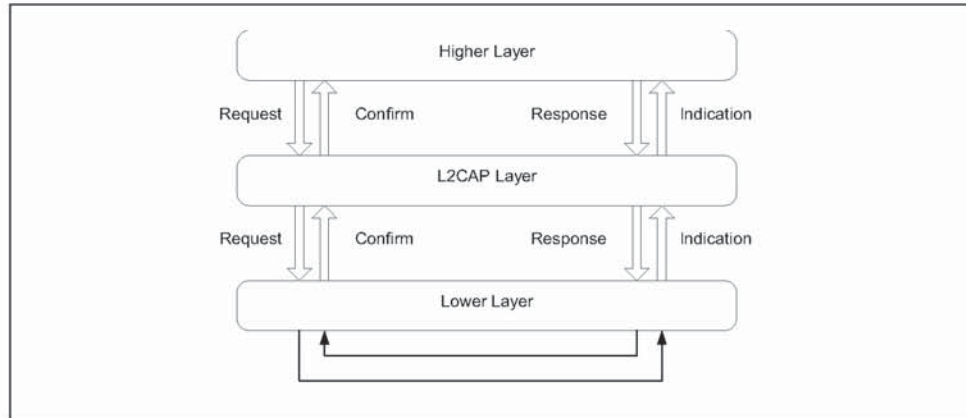


Figure 2.2: L2CAP Architecture

2.4 SEGMENTATION AND REASSEMBLY

Segmentation and reassembly (SAR) operations are used to improve efficiency by supporting a maximum transmission unit (MTU) size larger than the largest Baseband packet. This reduces overhead by spreading the network and transport packets used by higher layer protocols over several Baseband packets. All L2CAP packets may be segmented for transfer over Baseband packets. The protocol does not perform any segmentation and reassembly operations but the packet format supports adaptation to smaller physical frame sizes. An L2CAP implementation exposes the outgoing (i.e., the remote host's receiving) MTU and segments higher layer packets into 'chunks' that can be passed to the Link Manager via the Host Controller Interface (HCI), whenever one exists. On the receiving side, an L2CAP implementation receives 'chunks' from the HCI and reassembles those chunks into L2CAP packets using information provided through the HCI and from the packet header.

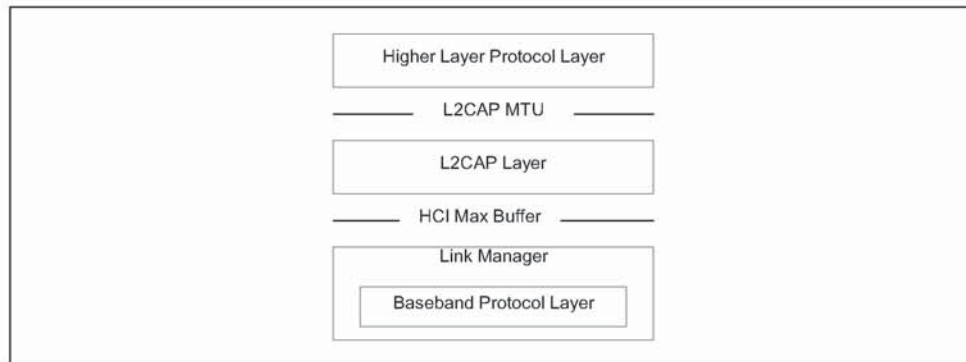


Figure 2.3: L2CAP SAR Variables

Segmentation and Reassembly is implemented using very little overhead in Baseband packets. The two L_CH bits defined in the first byte of Baseband

payload (also called the frame header) are used to signal the start and continuation of L2CAP packets. L_CH shall be '10' for the first segment in an L2CAP packet and '01' for a continuation segment. An example use of SAR is shown in Figure 2.4.

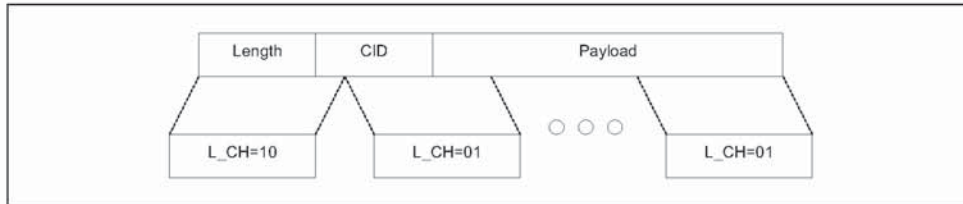


Figure 2.4: L2CAP segmentation

2.4.1 Segmentation Procedures

The L2CAP maximum transmission unit (MTU) will be exported using an implementation specific service interface. It is the responsibility of the higher layer protocol to limit the size of packets sent to the L2CAP layer below the MTU limit. An L2CAP implementation will segment the packet into protocol data units (PDUs) to send to the lower layer. If L2CAP runs directly over the Baseband Protocol, an implementation may segment the packet into Baseband packets for transmission over the air. If L2CAP runs above the host controller interface (typical scenario), an implementation may send block-sized chunks to the host controller where they will be converted into Baseband packets. All L2CAP segments associated with an L2CAP packet must be passed through to the Baseband before any other L2CAP packet destined to the same unit may be sent.

2.4.2 Reassembly Procedures

The Baseband Protocol delivers ACL packets in sequence and protects the integrity of the data using a 16-bit CRC. The Baseband also supports reliable connections using an automatic repeat request (ARQ) mechanism. As the Baseband controller receives ACL packets, it either signals the L2CAP layer on the arrival of each Baseband packets, or accumulates a number of packets before the receive buffer fills up or a timer expires before signalling the L2CAP layer.

L2CAP implementations must use the length field in the header of L2CAP packets, see Section 4 on page 272, as a consistency check and discard any L2CAP packets that fail to match the length field. If channel reliability is not needed, packets with improper lengths may be silently discarded. For reliable channels, L2CAP implementations must indicate to the upper layer that the channel has become unreliable. Reliable channels are defined by having an infinite flush timeout value as specified in Section 6.2 on page 290.

Figure 2.5 on page 257 illustrates the use of segmentation and reassembly operations to transmit a single higher layer PDU. Note that while there is a one-to-one mapping between a high layer PDU and an L2CAP packet, the segment

size used by the segmentation and reassembly routines is left to the implementation and may differ from the sender to the receiver.

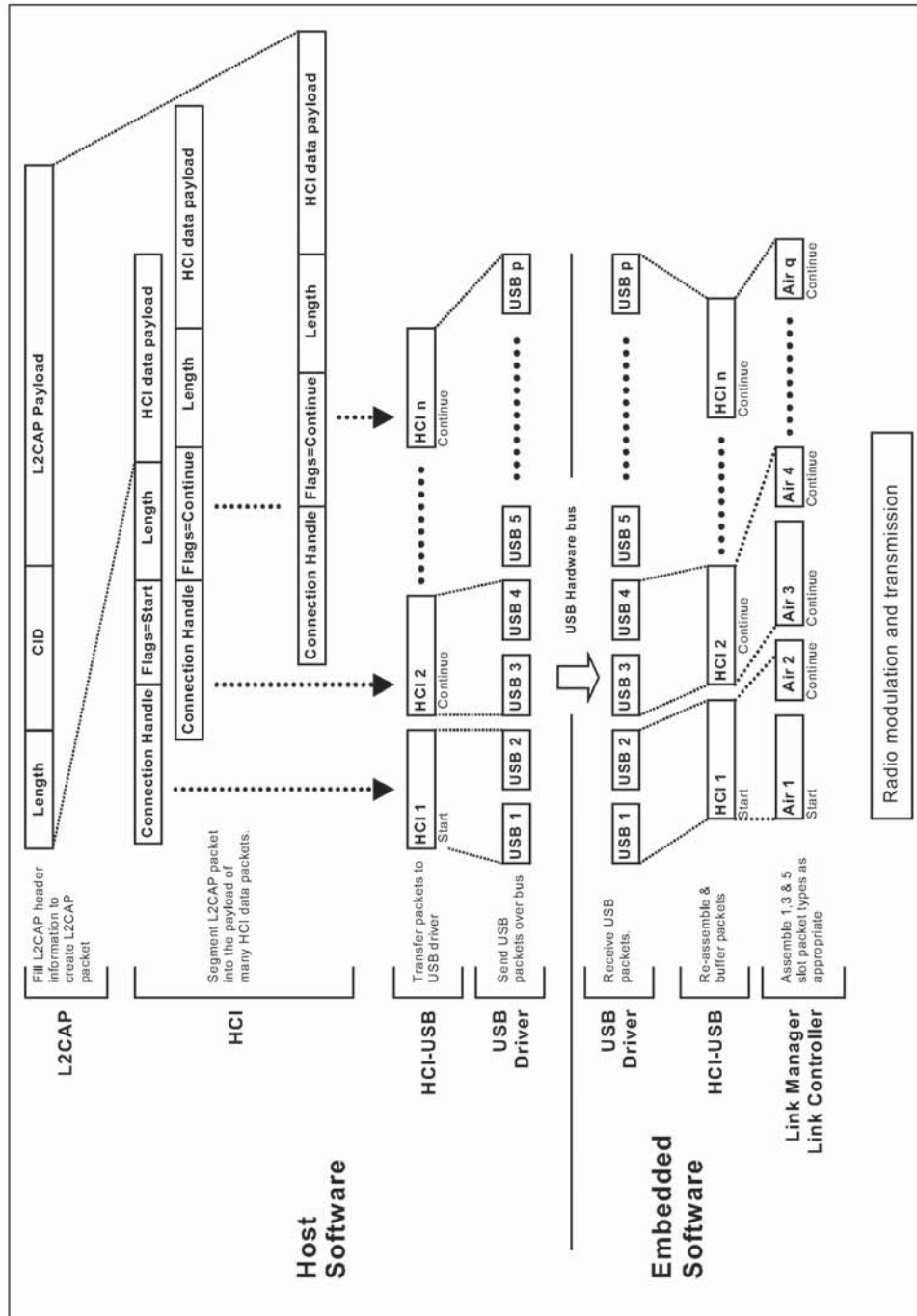


Figure 2.5: Segmentation and Reassembly Services in a unit with an HCI¹

3 STATE MACHINE

This section describes the L2CAP connection-oriented channel state machine. The section defines the states, the events causing state transitions, and the actions to be performed in response to events. This state machine is only pertinent to bi-directional CIDs and is not representative of the signalling channel or the uni-directional channel.

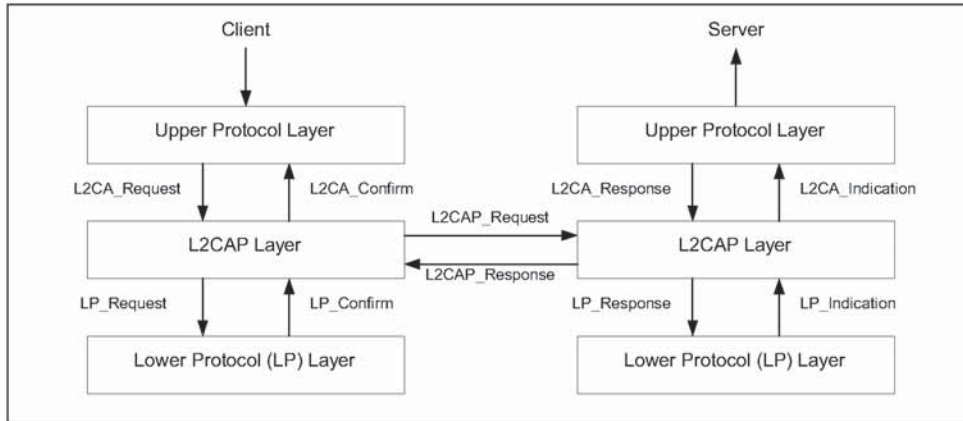


Figure 3.1: L2CAP Layer Interactions

Figure 3.1 illustrates the events and actions performed by an implementation of the L2CAP layer. Client and Server simply represent the initiator of the request and the acceptor of the request respectively. An application-level Client would both initiate and accept requests. The naming convention is as follows. The interface between two layers (vertical interface) uses the prefix of the lower layer offering the service to the higher layer, e.g., L2CA. The interface between two entities of the same layer (horizontal interface) uses the prefix of the protocol (adding a P to the layer identification), e.g., L2CAP. Events coming from above are called Requests (Req) and the corresponding replies are called Confirms (Cfm). Events coming from below are called Indications (Ind) and the corresponding replies are called Responses (Rsp). Responses requiring further processing are called Pending (Pnd). The notation for Confirms and Responses assumes positive replies. Negative replies are denoted by a 'Neg' suffix such as L2CAP_ConnectCfmNeg.

While Requests for an action always result in a corresponding Confirmation (for the successful or unsuccessful satisfaction of the action), Indications do not always result into corresponding Responses. The latter is especially true, if the Indications are informative about locally triggered events, e.g., seeing the

1. For simplicity, the stripping of any additional HCI and USB specific information fields prior to the creation of the baseband packets (Air_1, Air_2, etc.) is not shown in the figure.

LP_QoSViolationInd in Section 3.1.1 on page 259, or *L2CA_TimeOutInd* in Section 3.2.4 on page 264.

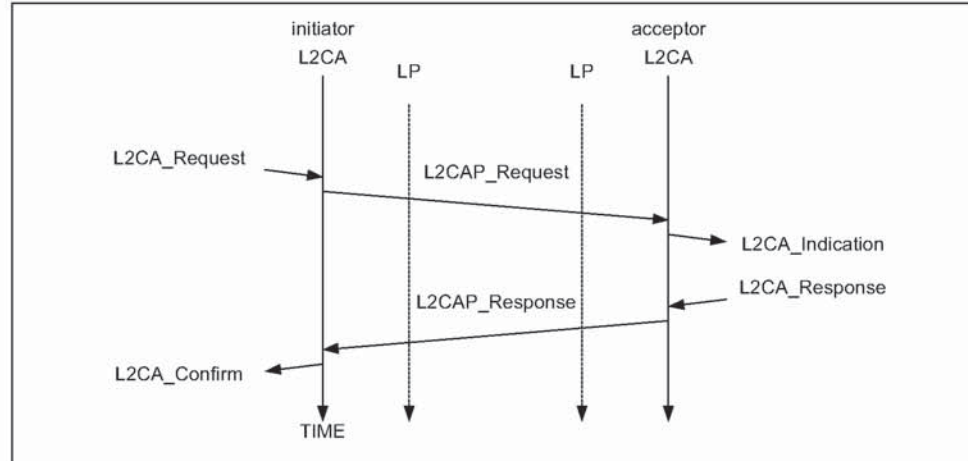


Figure 3.2: MSC of Layer Interactions

Figure 3.2 uses a message sequence chart (MSC) to illustrate the normal sequence of events. The two outer vertical lines represent the L2CA interface on the initiator (the device issuing a request) and the acceptor (the device responding to the initiator's request). Request commands at the L2CA interface result in Requests defined by the protocol. When the protocol communicates the request to the acceptor, the remote L2CA entity presents the upper protocol with an Indication. When the acceptor's upper protocol responds, the response is packaged by the protocol and communicated back to the initiator. The result is passed back to the initiator's upper protocol using a Confirm message.

3.1 EVENTS

Events are all incoming messages to the L2CA layer along with timeouts. Events are partitioned into five categories: Indications and Confirms from lower layers, Requests and Responses from higher layers, data from peers, signal Requests and Responses from peers, and events caused by timer expirations.

3.1.1 Lower-Layer Protocol (LP) to L2CAP events

- *LP_ConnectCfm*
Confirms the request (see *LP_ConnectReq* in Section 3.2.1) to establish a lower layer (Baseband) connection. This includes passing the authentication challenge if authentication is required to establish the physical link.
- *LP_ConnectCfmNeg*
Confirms the failure of the request (see *LP_ConnectReq* in Section 3.2.1) to establish a lower layer (Baseband) connection failed. This could be because

the device could not be contacted, refused the request, or the LMP authentication challenge failed.

- *LP_ConnectInd*

Indicates the lower protocol has successfully established connection. In the case of the Baseband, this will be an ACL link. An L2CAP entity may use to information to keep track of what physical links exist.

- *LP_DisconnectInd*

Indicates the lower protocol (Baseband) has been shut down by LMP commands or a timeout event.

- *LP_QoSConf*

Confirms the request (see *LP_QoSReq* in Section 3.2.1) for a given quality of service.

- *LP_QoSConfNeg*

Confirms the failure of the request (see *LP_QoSReq* in Section 3.2.1) for a given quality of service.

- *LP_QoSViolationInd*

Indicates the lower protocol has detected a violation of the QoS agreement specified in the previous *LP_QoSReq* (see Section 3.2.1).

3.1.2 L2CAP to L2CAP Signalling events

L2CAP to L2CAP signalling events are generated by each L2CAP entity following the exchange of the corresponding L2CAP signalling PDUs, see Section 5. L2CAP signalling PDUs, like any other L2CAP PDUs, are received from a lower layer via a lower protocol indication event. For simplicity of the presentation, we avoid a detailed description of this process, and we assume that signalling events are exchanged directly between the L2CAP peer entities as shown in Figure 3.1 on page 258.

- *L2CAP_ConnectReq*

A Connection Request packet has been received.

- *L2CAP_ConnectRsp*

A Connection Response packet has been received with a positive result indicating that the connection has been established.

- *L2CAP_ConnectRspPnd*

A Connection Response packet has been received indicating the remote endpoint has received the request and is processing it.

- *L2CAP_ConnectRspNeg*

A Connection Response packet has been received, indicating that the connection could not be established.

- *L2CAP_ConfigReq*

A Configuration Request packet has been received indicating the remote endpoint wishes to engage in negotiations concerning channel parameters.

- *L2CAP_ConfigRsp*

A Configuration Response packet has been received indicating the remote endpoint agrees with all the parameters being negotiated.

- *L2CAP_ConfigRspNeg*

A Configuration Response packet has been received indicating the remote endpoint does not agree to the parameters received in the response packet.

- *L2CAP_DisconnectReq*

A Disconnection Request packet has been received and the channel must initiate the disconnection process. Following the completion of an L2CAP channel disconnection process, an L2CAP entity should return the corresponding local CID to the pool of 'unassigned' CIDs.

- *L2CAP_DisconnectRsp*

A Disconnection Response packet has been received. Following the receipt of this signal, the receiving L2CAP entity may return the corresponding local CID to the pool of unassigned CIDs. There is no corresponding negative response because the Disconnect Request must succeed.

3.1.3 L2CAP to L2CAP Data events

- *L2CAP_Data*

A Data packet has been received.

3.1.4 Upper-Layer to L2CAP events

- *L2CA_ConnectReq*

Request from upper layer for the creation of a channel to a remote device.

- *L2CA_ConnectRsp*

Response from upper layer to the indication of a connection request from a remote device (see *L2CA_ConnectInd* in Section 3.2.4).

- *L2CA_ConnectRspNeg*

Negative response (rejection) from upper layer to the indication of a connection request from a remote device (see *L2CA_ConnectInd* in Section 3.2.4).

- *L2CA_ConfigReq*

Request from upper layer to (re)configure the channel.

- *L2CA_ConfigRsp*

Response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in Section 3.2.4).

- *L2CA_ConfigRspNeg*

A negative response from upper layer to the indication of a (re) configuration request (see *L2CA_ConfigInd* in Section 3.2.4).

- *L2CA_DisconnectReq*
Request from upper layer for the immediate disconnection of a channel.
- *L2CA_DisconnectRsp*
Response from upper layer to the indication of a disconnection request (see *L2CA_DisconnectInd* in Section 3.2.4). There is no corresponding negative response, the disconnect indication must always be accepted.
- *L2CA_DataRead*
Request from upper layer for the transfer of received data from L2CAP entity to upper layer.
- *L2CA_DataWrite*
Request from upper layer for the transfer of data from the upper layer to L2CAP entity for transmission over an open channel.

3.1.5 Timer events

- *RTX*

The Response Timeout eXpired (RTX) timer is used to terminate the channel when the remote endpoint is unresponsive to signalling requests. This timer is started when a signalling request (see Section 5 on page 275) is sent to the remote device. This timer is disabled when the response is received. If the initial timer expires, a duplicate Request message may be sent or the channel identified in the request may be disconnected. If a duplicate Request message is sent, the RTX timeout value must be reset to a new value at least double the previous value.

Implementations have the responsibility to decide on the maximum number of Request retransmissions performed at the L2CAP level before disconnecting the channel. The decision should be based on the flush timeout of the signalling link. The longer the flush timeout, the more retransmissions may be performed at the physical layer and the reliability of the channel improves, requiring fewer retransmissions at the L2CAP level. For example, if the flush timeout is infinite, no retransmissions should be performed at the L2CAP level.

The value of this timer is implementation-dependent but the minimum initial value is 1 second and the maximum initial value is 60 seconds. One RTX timer MUST exist for each outstanding signalling request, including each Echo Request. The timer disappears on the final expiration, when the response is received, or the physical link is lost. The maximum elapsed time between the initial start of this timer and the initiation of channel disconnection (if no response is received) is 60 seconds.

- *ERTX*

The Extended Response Timeout eXpired (ERTX) timer is used in place of the RTX timer when it is suspected the remote endpoint is performing addi-

tional processing of a request signal. This timer is started when the remote endpoint responds that a request is pending, e.g., when an *L2CAP_ConnectRspPnd* event is received. This timer is disabled when the formal response is received or the physical link is lost. If the initial timer expires, a duplicate Request may be sent or the channel may be disconnected. If a duplicate Request is sent, the particular ERTX timer disappears, replaced by a new RTX timer and the whole timing procedure restarts as described previously for the RTX timer.

The value of this timer is implementation-dependent but the minimum initial value is 60 seconds and the maximum initial value is 300 seconds. Similar to RTX, there MUST be at least one ERTX timer for each outstanding request that received a Pending response. There should be at most one (RTX or ERTX) associated with each outstanding request. The maximum elapsed time between the initial start of this timer and the initiation of channel disconnection (if no response is received) is 300 seconds.

3.2 ACTIONS

Actions are partitioned into five categories: Confirms and Indications to higher layers, Request and Responses to lower layers, Requests and Responses to peers, data transmission to peers, and setting timers.

3.2.1 L2CAP to Lower Layer actions

- *LP_ConnectReq*

L2CAP requests the lower protocol to create a connection. If a physical link to the remote device does not exist, this message must be sent to the lower protocol to establish the physical connection. Since no more than a single ACL link between two devices is assumed, see Section 1.2 on page 252, additional L2CAP channels between these two devices must share the same baseband ACL link.

Following the processing of the request, the lower layer returns with an *LP_ConnectCfm* or an *LP_ConnectCfmNeg* to indicate whether the request has been satisfied or not, respectively.

- *LP_QoSReq*

L2CAP requests the lower protocol to accommodate a particular QoS parameter set. Following the processing of the request, the lower layer returns with an *LP_QoSReqCfm* or an *LP_QoSReqCfmNeg* to indicate whether the request has been satisfied or not, respectively.

- *LP_ConnectRsp*

A positive response accepting the previous connection indication request (see *LP_ConnectInd* in Section 3.1.1).

- *LP_ConnectRspNeg*

A negative response denying the previous connection indication request (see *LP_ConnectInd* in Section 3.1.1).

3.2.2 L2CAP to L2CAP Signalling actions

This section contains the same names identified in Section 3.1.2 except the actions refer to the transmission, rather than reception, of these messages.

3.2.3 L2CAP to L2CAP Data actions

This section is the counterpart of Section 3.1.3. Data transmission is the action performed here.

3.2.4 L2CAP to Upper Layer actions

- *L2CA_ConnectInd*
Indicates a Connection Request has been received from a remote device (see *L2CA_ConnectReq* in Section 3.1.4).
- *L2CA_ConnectCfm*
Confirms that a Connection Request has been accepted (see (*L2CAP_ConnectReq* in Section 3.1.4) following the receipt of a Connection message from the remote device.
- *L2CA_ConnectCfmNeg*
Negative confirmation (failure) of a Connection Request (see *L2CA_ConnectReq* in Section 3.1.4). An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.
- *L2CA_ConnectPnd*
Confirms that a Connection Response (pending) has been received from the remote device.
- *L2CA_ConfigInd*
Indicates a Configuration Request has been received from a remote device.
- *L2CA_ConfigCfm*
Confirms that a Configuration Request has been accepted (see *L2CA_ConfigReq* in Section 3.1.4) following the receipt of a Configuration Response from the remote device.
- *L2CA_ConfigCfmNeg*
Negative confirmation (failure) of a Configuration Request (see *L2CA_ConfigReq* in Section 3.1.4). An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Connect Request can substitute for a negative Connect Response and result in this action.

- *L2CA_DisconnectInd*
Indicates a Disconnection Request has been received from a remote device or the remote device has been disconnected because it has failed to respond to a signalling request. See Section 3.1.5
- *L2CA_DisconnectCfm*
Confirms that a Disconnect Request has been processed by the remote device (see *L2CA_DisconnectReq* in Section 3.1.4) following the receipt of a Disconnection Response from the remote device. An RTX timer expiration (see Section 3.1.5 and *L2CA_TimeOutInd* below) for an outstanding Disconnect Request can substitute for a Disconnect Response and result in this action. Upon receiving this event the upper layer knows the L2CAP channel has been terminated. There is no corresponding negative confirm.
- *L2CA_TimeOutInd*
Indicates that a RTX or ERTX timer has expired. This indication will occur an implementation-dependant number of times before the L2CAP implementation will give up and send a *L2CA_DisconnectInd*.
- *L2CA_QoSViolationInd*
Indicates that the quality of service agreement has been violated.

3.3 CHANNEL OPERATIONAL STATES

- *CLOSED*
In this state, there is no channel associated with this CID. This is the only state when a link level connection (Baseband) may not exist. Link disconnection forces all other states into the *CLOSED* state.
- *W4_L2CAP_CONNECT_RSP*
In this state, the CID represents a local end-point and an *L2CAP_ConnectReq* message has been sent referencing this endpoint and it is now waiting for the corresponding *L2CAP_ConnectRsp* message.
- *W4_L2CA_CONNECT_RSP*
In this state, the remote end-point exists and an *L2CAP_ConnectReq* has been received by the local L2CAP entity. An *L2CA_ConnectInd* has been sent to the upper layer and the part of the local L2CAP entity processing the received *L2CAP_ConnectReq* waits for the corresponding response. The response may require a security check to be performed.
- *CONFIG*
In this state, the connection has been established but both sides are still negotiating the channel parameters. The Configuration state may also be entered when the channel parameters are being renegotiated. Prior to entering the *CONFIG* state, all outgoing data traffic should be suspended since the traffic parameters of the data traffic are to be renegotiated. Incoming data traffic must be accepted until the remote channel endpoint has entered the *CONFIG* state.

In the CONFIG state, both sides must issue L2CAP_ConfigReq messages – if only defaults are being used, a null message should be sent, see Section 5.4 on page 280. If a large amount of parameters need to be negotiated, multiple messages may be sent to avoid any MTU limitations and negotiate incrementally – see Section 6 on page 289 for more details.

Moving from the CONFIG state to the OPEN state requires both sides to be ready. An L2CAP entity is ready when it has received a positive response to its final request and it has positively responded to the final request from the remote device.

- *OPEN*

In this state, the connection has been established and configured, and data flow may proceed.

- *W4_L2CAP_DISCONNECT_RSP*

In this state, the connection is shutting down and an L2CAP_DisconnectReq message has been sent. This state is now waiting for the corresponding response.

- *W4_L2CA_DISCONNECT_RSP*

In this state, the connection on the remote endpoint is shutting down and an L2CAP_DisconnectReq message has been received. An L2CA_DisconnectInd has been sent to the upper layer to notify the owner of the CID that the remote endpoint is being closed. This state is now waiting for the corresponding response from the upper layer before responding to the remote endpoint.

3.4 MAPPING EVENTS TO ACTIONS

Table 3.1 defines the actions taken in response to events that occur in a particular state. Events that are not listed in the table, nor have actions marked N/C (for no change), are assumed to be errors and silently discarded.

Data input and output events are only defined for the Open and Configuration states. Data may not be received during the initial Configuration state, but may be received when the Configuration state is re-entered due to a reconfiguration process. Data received during any other state should be silently discarded.

Event	Current State	Action	New State
LP_ConnectCfm	CLOSED	Flag physical link as up and initiate the L2CAP connection.	CLOSED
LP_ConnectCfmNeg	CLOSED	Flag physical link as down and fail any outstanding service connection requests by sending an L2CA_ConnectCfmNeg message to the upper layer.	CLOSED
LP_ConnectInd	CLOSED	Flag link as up.	CLOSED
LP_DisconnectInd	CLOSED	Flag link as down.	CLOSED
LP_DisconnectInd	Any except CLOSED	Send upper layer L2CA_DisconnectInd message.	CLOSED
LP_QoSViolationInd	Any but OPEN	Discard	N/C
LP_QoSViolationInd	OPEN	Send upper layer L2CA_QoSViolationInd message. If service level is guaranteed, terminate the channel.	OPEN or W4_L2CA_DISCONNECT_RSP
L2CAP_ConnectReq	CLOSED. (CID dynamically allocated from free pool.)	Send upper layer L2CA_ConnectInd. Optionally: Send peer L2CAP_ConnectRspPnd	W4_L2CA_CONNECT_RSP
L2CAP_ConnectRsp	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfm message. Disable RTX timer.	CONFIG
L2CAP_ConnectRspPnd	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectPnd message. Disable RTX timer and start ERTX timer.	N/C
L2CAP_ConnectRspNeg	W4_L2CAP_CONNECT_RSP	Send upper layer L2CA_ConnectCfmNeg message. Return CID to free pool. Disable RTX/ERTX timers.	CLOSED
L2CAP_ConfigReq	CLOSED	Send peer L2CAP_ConfigRspNeg message.	N/C
L2CAP_ConfigReq	CONFIG	Send upper layer L2CA_ConfigInd message.	N/C

Table 3.1: L2CAP Channel State Machine

Event	Current State	Action	New State
L2CAP_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send upper layer L2CA_ConfigInd message.	CONFIG
L2CAP_ConfigRsp	CONFIG	Send upper layer L2CA_ConfigCfm message. Disable RTX timer. If an L2CAP_ConfigReq message has been received and positively responded to, then enter OPEN state, otherwise remain in CONFIG state.	N/C or OPEN
L2CAP_ConfigRsp Neg	CONFIG	Send upper layer L2CA_ConfigCfmNeg message. Disable RTX timer.	N/C
L2CAP_DisconnectReq	CLOSED	Send peer L2CAP_DisconnectRsp message.	N/C
L2CAP_DisconnectReq	Any except CLOSED	Send upper layer L2CA_DisconnectInd message.	W4_L2CA_DISCONNECT_RSP
L2CAP_DisconnectRsp	W4_L2CAP_DISCONNECT_RSP	Send upper layer L2CA_DisconnectCfm message. Disable RTX timer.	CLOSED
L2CAP_Data	OPEN or CONFIG	If complete L2CAP packet received, send upper layer L2CA_Read confirm.	N/C
L2CA_ConnectReq	CLOSED (CID dynamically allocated from free pool)	Send peer LP2CAP_ConnectReq message. Start RTX timer.	W4_L2CAP_CONNECT_RSP
L2CA_ConnectRsp	W4_L2CA_CONNECT_RSP	Send peer L2CAP_ConnectRsp message.	CONFIG
L2CA_ConnectRsp Neg	W4_L2CA_CONNECT_RSP	Send peer L2CAP_ConnectRspNeg message. Return CID to free pool.	CLOSED
L2CA_ConfigReq	CLOSED	Send upper layer L2CA_ConfigCfmNeg message.	N/C
L2CA_ConfigReq	CONFIG	Send peer L2CAP_ConfigReq message. Start RTX timer.	N/C

Table 3.1: L2CAP Channel State Machine

Event	Current State	Action	New State
L2CA_ConfigReq	OPEN	Suspend data transmission at a convenient point. Send peer L2CAP_ConfigReq message. Start RTX timer.	CONFIG
L2CA_ConfigRsp	CONFIG	Send peer L2CAP_ConfigRsp message. If all outstanding L2CAP_ConfigReq messages have received positive responses then move in OPEN state. Otherwise, remain in CONFIG state.	N/C or OPEN
L2CA_ConfigRspNeg	CONFIG	Send peer L2CAP_ConfigRspNeg message.	N/C
L2CA_DisconnectReq	OPEN or CONFIG	Send peer L2CAP_DisconnectReq message. Start RTX timer.	W4_L2CAP_DISCONNECT_RSP
L2CA_DisconnectRsp	W4_L2CAP_DISCONNECT_RSP	Send peer L2CAP_DisconnectRsp message. Return CID to free pool.	CLOSED
L2CA_DataRead	OPEN	If payload complete, transfer payload to InBuffer.	OPEN
L2CA_DataWrite	OPEN	Send peer L2CAP_Data message.	OPEN
Timer_RTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool else re-send Request.	CLOSED
Timer_ERTX	Any	Send upper layer L2CA_TimeOutInd message. If final expiration, return CID to free pool else re-send Request.	CLOSED

Table 3.1: L2CAP Channel State Machine

Figure 3.3 illustrates a simplified state machine and typical transition path taken by an initiator and acceptor. The state machine shows what events cause state transitions and what actions are also taken while the transitions occur. Not all the events listed in Table 3.1 are included in the simplified State Machine to avoid cluttering the figure.

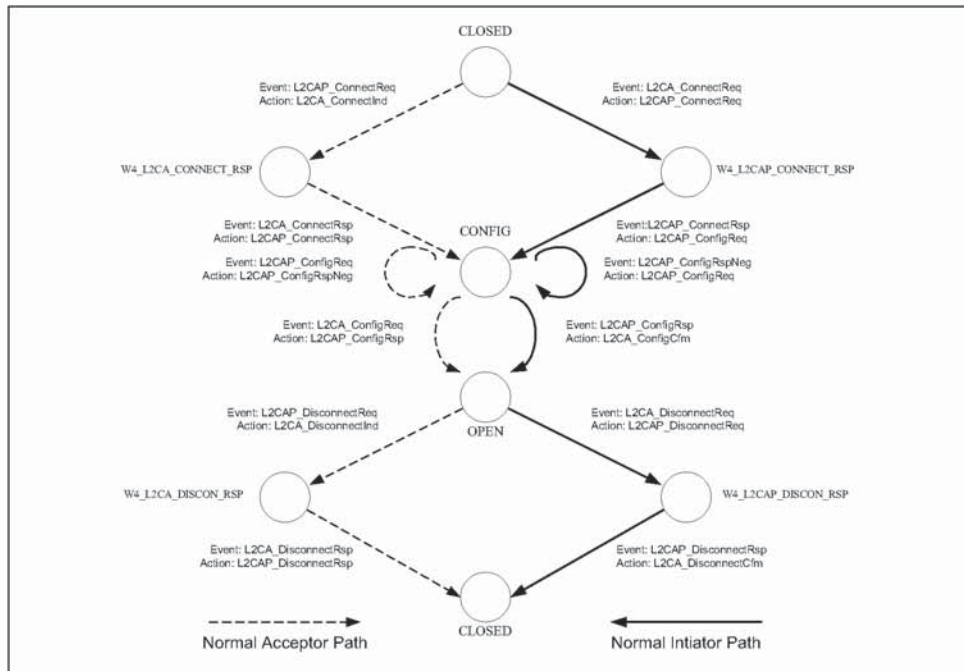


Figure 3.3: State Machine Example

Figure 3.4 presents another illustration of the events and actions based around the messages sequences being communicated between two devices. In this example, the initiator is creating the first L2CAP channel between two devices. Both sides start in the CLOSED state. After receiving the request from the upper layer, the entity requests the lower layer to establish a physical link. If no physical link exists, LMP commands are used to create the physical link between the devices. Once the physical link is established, L2CAP signals may be sent over it.

Figure 3.4 is an example and not all setup sequences will be identical to the one illustrated below.

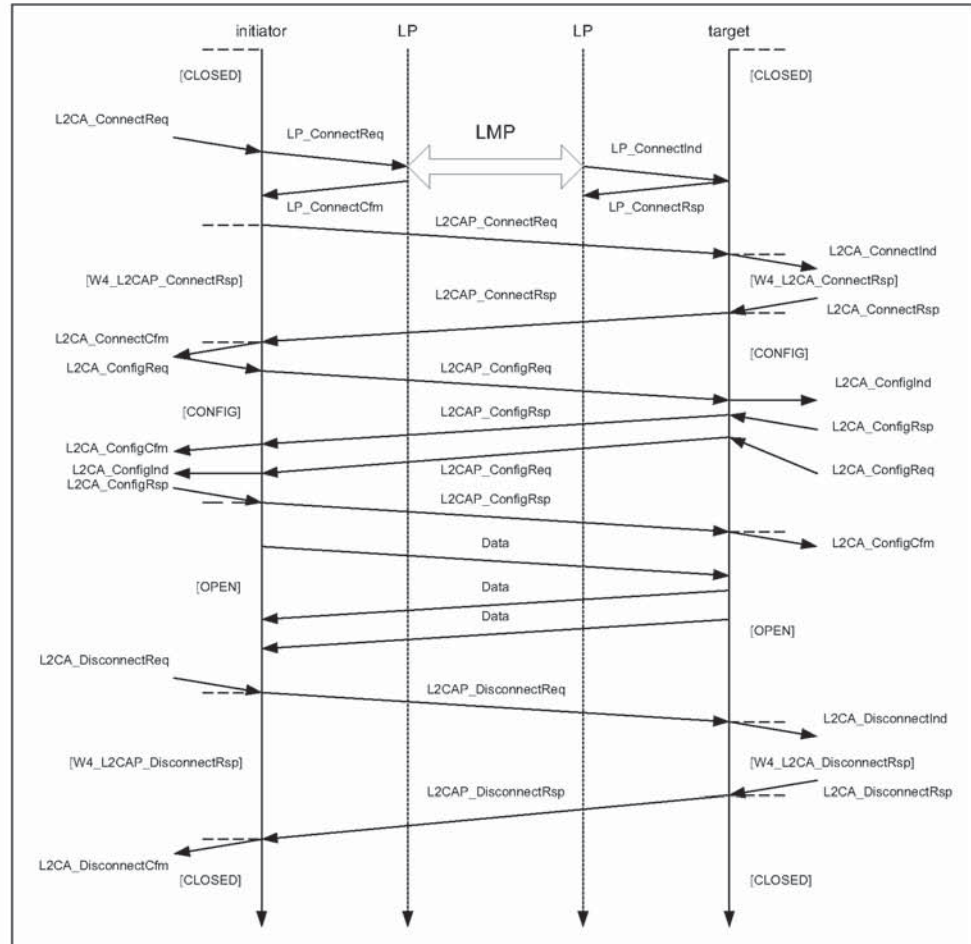


Figure 3.4: Message Sequence Chart of Basic Operation

4 DATA PACKET FORMAT

L2CAP is packet-based but follows a communication model based on *channels*. A channel represents a data flow between L2CAP entities in remote devices. Channels may be connection-oriented or connectionless. All packet fields use Little Endian byte order.

4.1 CONNECTION-ORIENTED CHANNEL

Figure 4.1 illustrates the format of the L2CAP packet (also referred to as the L2CAP PDU) within a connection-oriented channel.

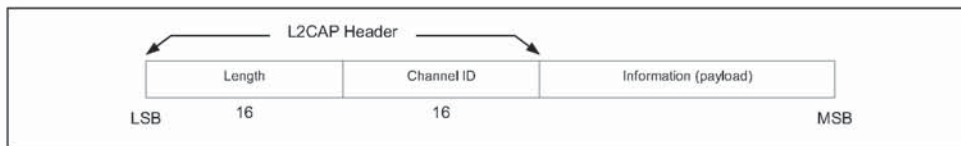


Figure 4.1: L2CAP Packet (field sizes in bits)

The fields shown are:

- *Length: 2 octets (16 bits)*

Length indicates the size of information payload in bytes, excluding the length of the L2CAP header. The length of an information payload can be up to 65535 bytes. The Length field serves as a simple integrity check of the reassembled L2CAP packet on the receiving end.

- *Channel ID: 2 octets*

The channel ID identifies the destination channel endpoint of the packet. The scope of the channel ID is relative to the device the packet is being sent to.

- *Information: 0 to 65535 octets*

This contains the payload received from the upper layer protocol (outgoing packet), or delivered to the upper layer protocol (incoming packet). The minimum supported MTU for connection-oriented packets (MTU_{cno}) is negotiated during channel configuration (see Section 6.1 on page 289). The minimum supported MTU for the signalling packet (MTU_{sig}) is 48 bytes (see Section 5 on page 275).

4.2 CONNECTIONLESS DATA CHANNEL

In addition to connection-oriented channels, L2CAP also exports the concept of a group-oriented channel. Data sent to the 'group' channel is sent to all members of the group in a best-effort manner. Groups have no quality of service associated with them. Group channels are unreliable; L2CAP makes no guarantee that data sent to the group successfully reaches all members of the group. If reliable group transmission is required, it must be implemented at a higher layer.

Transmissions to a group must be non-exclusively sent to all members of that group. The local device cannot be a member of the group, and higher layer protocols are expected to loopback any data traffic being sent to the local device. Non-exclusive implies non-group members may receive group transmissions and higher level (or link level) encryption can be used to support private communication.

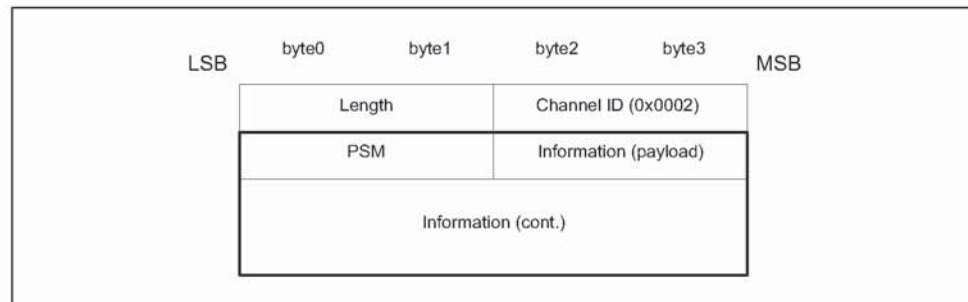


Figure 4.2: Connectionless Packet

The fields shown are:

- **Length: 2 octets**
Length indicates the size of information payload plus the PSM field in bytes, excluding the length of the L2CAP header.
- **Channel ID: 2 octets**
Channel ID (0x0002) reserved for connectionless traffic.
- **Protocol/Service Multiplexer (PSM): 2 octets (minimum)**
The PSM field is based on the ISO 3309 extension mechanism for address fields. All content of the PSM field, referred to as the PSM value, must be ODD, that is, the least significant bit of the least significant octet must be '1'. Also, all PSM values must be assigned such that the least significant bit of the most significant octet equals '0'. This allows the PSM field to be extended beyond 16 bits. The PSM value definitions are specific to L2CAP and assigned by the Bluetooth SIG. For more information on the PSM field see Section 5.2 on page 278.

| • *Information: 0 to 65533 octets*

The payload information to be distributed to all members of the group. Implementations must support a minimum connectionless MTU (MTU_{cni}) of 670 octets, unless explicitly agreed upon otherwise, e.g., for single operation devices that are built to comply to a specific Bluetooth profile that dictates the use of a specific MTU for connectionless traffic that is less than MTU_{cni} .

| The L2CAP group service interface provides basic group management mechanisms including creating a group, adding members to a group, and removing members from a group. There are no pre-defined groups such as 'all radios in range'.

5 SIGNALLING

This section describes the signalling commands passed between two L2CAP entities on remote devices. All signalling commands are sent to CID 0x0001. The L2CAP implementation must be able to determine the Bluetooth address (BD_ADDR) of the device that sent the commands. Figure 5.1 illustrates the general format of all L2CAP packets containing signalling commands. Multiple commands may be sent in a single (L2CAP) packet and packets are sent to CID 0x0001. MTU Commands take the form of Requests and Responses. All L2CAP implementations must support the reception of signalling packets whose MTU (MTU_{sig}) does not exceed 48 bytes. L2CAP implementations should not use signalling packets beyond this size without first testing whether the implementation can support larger signalling packets.

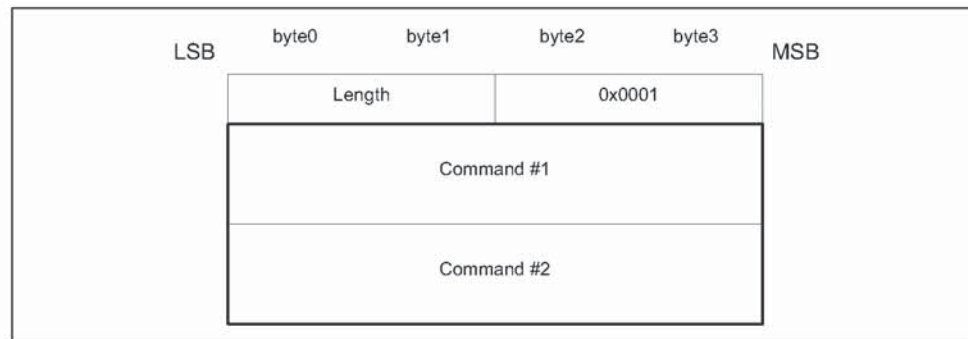


Figure 5.1: Signalling Command Packet Format

Figure 5.2 displays the general format of all signalling commands.

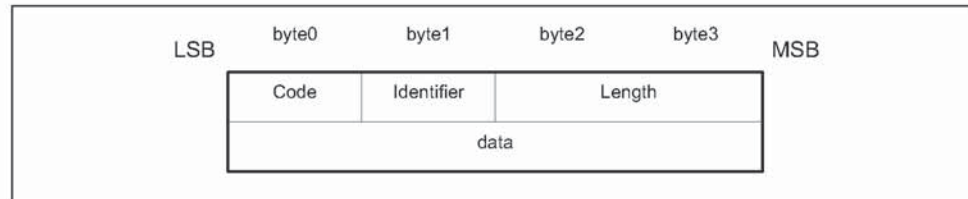


Figure 5.2: Command format

The fields shown are:

- *Code: 1 octet*

The Code field is one octet long and identifies the type of command. When a packet is received with an unknown Code field, a Command Reject packet (defined in Section 5.1 on page 277) is sent in response.

Up-to-date values of assigned Codes are specified in the latest Bluetooth 'Assigned Numbers' document (page 1009). Table 5.1 on page 276 lists the codes defined by this document. All codes are specified with the most significant bit in the left-most position.

Code	Description
0x00	RESERVED
0x01	Command reject
0x02	Connection request
0x03	Connection response
0x04	Configure request
0x05	Configure response
0x06	Disconnection request
0x07	Disconnection response
0x08	Echo request
0x09	Echo response
0x0A	Information request
0x0B	Information response

Table 5.1: Signalling Command Codes

- *Identifier: 1 octet*

The Identifier field is one octet long and helps matching a request with the reply. The requesting device sets this field and the responding device uses the same value in its response. A different Identifier must be used for each original command. Identifiers should not be recycled until a period of 360 seconds has elapsed from the initial transmission of the command using the identifier. On the expiration of a RTX or ERTX timer, the same identifier should be used if a duplicate Request is re-sent as stated in Section 3.1.5 on page 262. A device receiving a duplicate request should reply with a duplicate response. A command response with an invalid identifier is silently discarded. Signalling identifier 0x0000 is defined to be an illegal identifier and shall never be used in any command.

- *Length: 2 octets*

The Length field is two octets long and indicates the size in octets of the data field of the command only, i.e., it does not cover the Code, Identifier, and Length fields.

- *Data: 0 or more octets*

The Data field is variable in length and discovered using the Length field. The Code field determines the format of the Data field.

5.1 COMMAND REJECT (CODE 0x01)

A Command Reject packet is sent in response to a command packet with an unknown command code or when sending the corresponding Response is inappropriate. Figure 5.3 displays the format of the packet. The Identifier should match the Identifier of the packet containing the unidentified code field. Implementations must always send these packets in response to unidentified signalling packets.

When multiple commands are included in an L2CAP packet and the packet exceeds the MTU of the receiver, a single Command Reject packet is sent in response. The identifier should match the first Request command in the L2CAP packet. If only Responses are recognized, the packet shall be silently discarded.

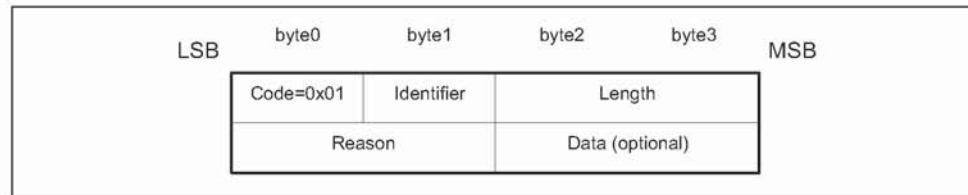


Figure 5.3: Command Reject Packet

- Length = 0x0002 or more octets
- Reason: 2 octets

The Reason field describes why the Request packet was rejected.

Reason value	Description
0x0000	Command not understood
0x0001	Signalling MTU exceeded
0x0002	Invalid CID in request
Other	Reserved

Table 5.2: Reason Code Descriptions

- Data: 0 or more octets

The length and content of the Data field depends on the Reason code. If the Reason code is 0x0000, "Command not understood", no Data field is used. If the Reason code is 0x0001, "Signalling MTU Exceeded", the 2-octet Data field represents the maximum signalling MTU the sender of this packet can accept.

If a command refers to an invalid channel then the Reason code 0x0002 will be returned. Typically a channel is invalid because it does not exist. A 4-octet data field on the command reject will contain the local (first) and remote (second) channel endpoints (relative to the sender of the Command Reject) of the disputed channel. The latter endpoints are obtained from the corresponding rejected command. If the rejected command contains only one of the channel endpoints, the other one is replaced by the null CID 0x0000.

Reason value	Data Length	Data value
0x0000	0 octets	N/A
0x0001	2 octets	Actual MTU
0x0002	4 octets	Requested CID

Table 5.3: Reason Data values

5.2 CONNECTION REQUEST (CODE 0x02)

Connection request packets are sent to create a channel between two devices. The channel connection must be established before configuration may begin. Figure 5.4 illustrates a Connection Request packet.

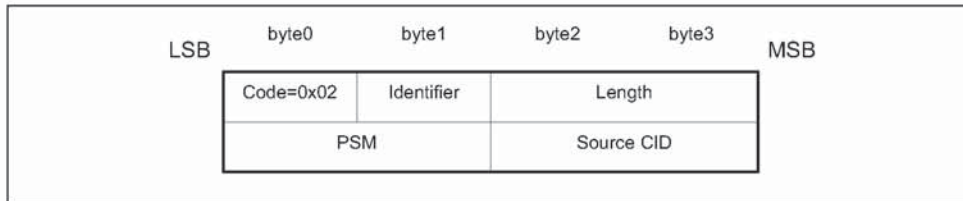


Figure 5.4: Connection Request Packet

- Length = 0x0004 or more octets
- Protocol/Service Multiplexor (PSM): 2 octets (minimum)

The PSM field is two octets (minimum) in length. The structure of the PSM field is based on the ISO 3309 extension mechanism for address fields. All PSM values must be ODD, that is, the least significant bit of the least significant octet must be '1'. Also, all PSM values must be assigned such that the least significant bit of the most significant octet equals '0'. This allows the PSM field to be extended beyond 16 bits. PSM values are separated into two ranges. Values in the first range are assigned by the Bluetooth SIG and indicate protocols. The second range of values are dynamically allocated and used in conjunction with the Service Discovery Protocol (SDP). The dynamically assigned values may be used to support multiple implementations of a particular protocol, e.g., RFCOMM, residing on top of L2CAP or prototyping an experimental protocol.

PSM value	Description
0x0001	Service Discovery Protocol
0x0003	RFCOMM
0x0005	Telephony Control Protocol
<0x1000	RESERVED
[0x1001-0xFFFF]	DYNAMICALLY ASSIGNED

Table 5.4: Defined PSM Values

- *Source CID (SCID): 2 octets*

The source local CID is two octets in length and represents a channel end-point on the device sending the request. Once the channel has been configured, data packets flowing from the sender of the request must be sent to this CID. In this section, the Source CID represents the channel endpoint on the device sending the request and receiving the response, while the Destination CID represents the channel endpoint on the device receiving the request and sending the response.

5.3 CONNECTION RESPONSE (CODE 0x03)

When a unit receives a Connection Request packet, it must send a Connection Response packet. The format of the connection response packet is shown in Figure 5.5.

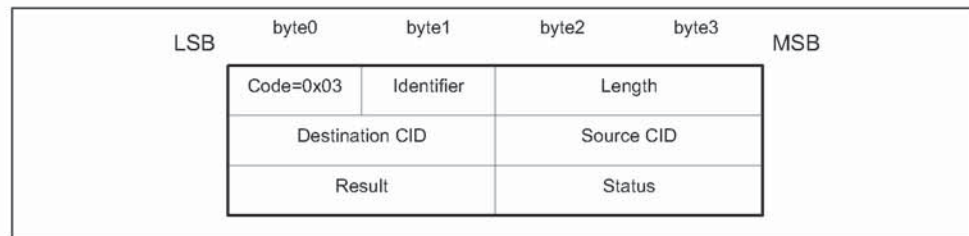


Figure 5.5: Connection Response Packet

- *Length = 0x0008 octets*
- *Destination Channel Identifier (DCID): 2 octets*
The field contains the channel end-point on the device sending this Response packet.
- *Source Channel Identifier (SCID): 2 octets*
The field contains the channel end-point on the device receiving this Response packet.

• *Result: 2 octets*

The result field indicates the outcome of the connection request. The result value of 0x0000 indicates success while a non-zero value indicates the connection request failed. A logical channel is established on the receipt of a successful result. Table 5.5 defines values for this field. If the result field is not zero, the DCID and SCID fields should be ignored.

Value	Description
0x0000	Connection successful.
0x0001	Connection pending
0x0002	Connection refused – PSM not supported.
0x0003	Connection refused – security block.
0x0004	Connection refused – no resources available.
Other	Reserved.

Table 5.5: Result values

• *Status: 2 octets*

Only defined for Result = Pending. Indicates the status of the connection.

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
Other	Reserved

Table 5.6: Status values

5.4 CONFIGURATION REQUEST (CODE 0x04)

Configuration Request packets are sent to establish an initial logical link transmission contract between two L2CAP entities and also to re-negotiate this contract whenever appropriate. During a re-negotiation session, all data traffic on the channel should be suspended pending the outcome of the negotiation. Each configuration parameter in a Configuration Request is related exclusively either with the outgoing or the incoming data traffic but not both of them. In Section 6 on page 289, the various configuration parameters and their relation to the outgoing or incoming data traffic are presented. If an L2CAP entity receives a Configuration Request while it is waiting for a response it must not block sending the Configuration Response, otherwise the configuration process may deadlock.

If no parameters need to be negotiated, no options need to be inserted and the C-bit should be cleared. L2CAP entities in remote devices MUST negotiate all parameters defined in this document whenever the default values are not

acceptable. Any missing configuration parameters are assumed to have their most recently (mutually) explicitly or implicitly accepted values. Even if all default values are acceptable, a Configuration Request packet with no options MUST be sent. Implicitly accepted values are any default values for the configuration parameters specified in this document that have not been explicitly negotiated for the specific channel under configuration.

Each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. If a device needs to establish the value of a configuration parameter in the opposite direction than the one implied by a Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Connection Request.

The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is left to the implementation but it shall not last more than 120 seconds.

Figure 5.6 defines the format of the Configuration Request packet.

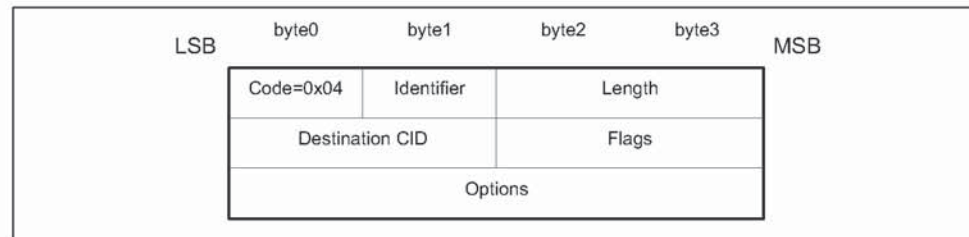


Figure 5.6: Configuration Request Packet

- Length = 0x0004 or more octets
- Destination CID (DCID): 2 octets

The field contains the channel end-point on the device receiving this Request packet.

- Flags: 2 octets

Figure 5.7 display the two-octet Flags field. Note the most significant bit is shown on the left.



Figure 5.7: Configuration Request Flags field format

C – more configuration requests will follow when set to 1. This flag indicates that the remote device should not enter OPEN state after agreeing to these parameters because more parameter negotiations are being sent. Segment-

ing the Configuration Request packet is necessary if the parameters exceed the MTU_{sig} .

Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

- *Configuration Options*

The list of the parameters and their values to be negotiated. These are defined in Section 6 on page 289. Configuration Requests may contain no options (referred to as an empty or null configuration request) and can be used to request a response. For an empty configuration request the length field is set to 0x0004.

5.5 CONFIGURE RESPONSE (CODE 0X05)

Configure Response packets MUST be sent in reply to Configuration Request packets. Each configuration parameter value (if any is present) in a Configuration Response reflects an 'adjustment' to a configuration parameter value that has been sent (or, in case of default values, implied) in the corresponding Configuration Request. Thus, for example, if a configuration parameter in a Configuration Request relates to traffic flowing from device A to device B, the sender of the Configuration Response will only adjust (if needed) this value again for the same traffic flowing from device A to device B. The options sent in the Response depend on the value in the Result field. Figure 5.8 defines the format of the Configuration Response packet.

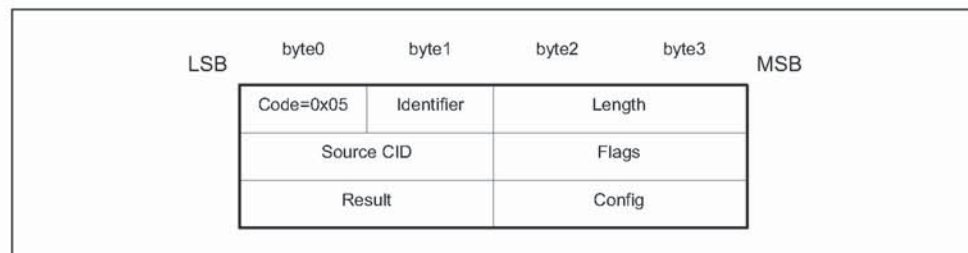


Figure 5.8: Configuration Response Packet

- Length = 0x0006 or more octets
- Source CID (SCID): 2 octets

The field contains the channel end-point on the device receiving this Response packet. The device receiving the Response must check that the Identifier field matches the same field in the corresponding configuration request command and the SCID matches its local CID paired with the original DCID.

- Flags: 2 octets

Figure 5.9 displays the two-octet Flags field. Note the most significant bit is shown on the left.



Figure 5.9: Configuration Response Flags field format

C – more configuration responses will follow when set to 1. This flag indicates that the parameters included in the response are a partial subset of parameters being sent by the device sending the Response packet.

Other flags are reserved and should be cleared. L2CAP implementations should ignore these bits.

- *Result: 2 octets*

The Result field indicates whether or not the Request was acceptable. See Table 5.7 for possible result codes.

Result	Description
0x0000	Success
0x0001	Failure – unacceptable parameters
0x0002	Failure – rejected (no reason provided)
0x0003	Failure – unknown options
Other	RESERVED

Table 5.7: Configuration Response Result codes

- *Configuration Options*

This field contains the list of parameters being negotiated. These are defined in Section 6 on page 289. On a successful result, these parameters contain the return values for any wild card parameters (see Section 6.3 on page 291) contained in the request.

On an unacceptable parameters failure (Result=0x0001) the rejected parameters should be sent in the response with the values that would have been accepted if sent in the original request. Any missing configuration parameters are assumed to have their most recently (mutually) accepted values and they too can be included in the Configuration Response if need to be changed. Recall that, each configuration parameter is one-directional and relative to the direction implied by the sender of a Configuration Request. Thus, if the sender of the Configuration Response needs to establish the value of a configuration parameter in the opposite direction than the one implied by an original Configuration Request, a new Configuration Request with the desired value of the configuration parameter in it needs to be sent in the direction opposite the one used for the original Connection Request.

On an unknown option failure (Result=0x0003), the option types not understood by the recipient of the Request must be included in the Response. Note that hints (defined in Section 6 on page 289), those options in the Request that are skipped if not understood, must not be included in the Response and must not be the sole cause for rejecting the Request.

The decision on the amount of time (or messages) spent arbitrating the channel parameters before terminating the negotiation is left to the implementation.

5.6 DISCONNECTION REQUEST (CODE 0x06)

Terminating an L2CAP channel requires that a disconnection request packet be sent and acknowledged by a disconnection response packet. Disconnection is requested using the signalling channel since all other L2CAP packets sent to the destination channel automatically get passed up to the next protocol layer. Figure 5.10 displays a disconnection packet request. The receiver must ensure both source and destination CIDs match before initiating a connection disconnection. Once a Disconnection Request is issued, all incoming data in transit on this L2CAP channel will be discarded and any new additional outgoing data is not allowed. Once a disconnection request for a channel has been received, all data queued to be sent out on that channel may be discarded.

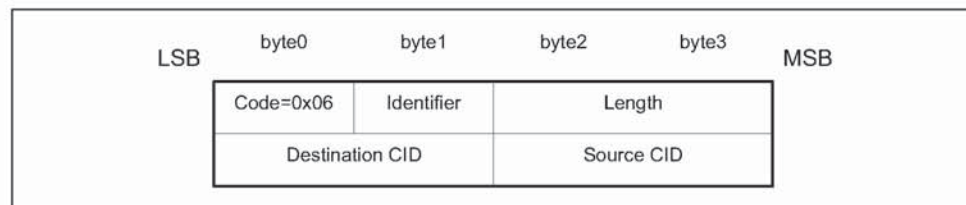


Figure 5.10: Disconnection Request Packet

- *Length = 0x0004 octets*
- *Destination CID (DCID): 2 octets*

This field specifies the end-point of the channel to be shutdown on the device receiving this request.

- *Source CID (SCID): 2 octets*

This field specifies the end-point of the channel to be shutdown on the device sending this request.

The SCID and DCID are relative to the sender of this request and must match those of the channel to be disconnected. If the DCID is not recognized by the receiver of this message, a CommandReject message with 'invalid CID' result code must be sent in response. If the receiver finds a DCID match but the SCID fails to find the same match, the request should be silently discarded.

5.7 DISCONNECTION RESPONSE (CODE 0x07)

Disconnection responses should be sent in response to each disconnection request.

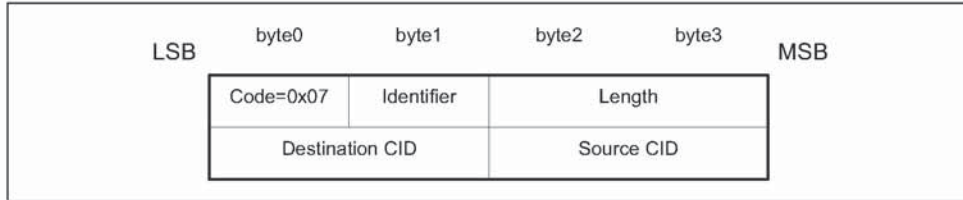


Figure 5.11: Disconnection Response Packet

- *Length = 0x0004 octets*
- *Destination CID (DCID): 2 octets*
This field identifies the channel end-point on the device sending the response.
- *Source CID (SCID): 2 octets*
This field identifies the channel end-point on the device receiving the response.
The DCID and the SCID (which are relative to the sender of the request), and the Identifier fields must match those of the corresponding disconnection request command. If the CIDs do not match, the response should be silently discarded at the receiver.

5.8 ECHO REQUEST (CODE 0x08)

Echo requests are used to solicit a response from a remote L2CAP entity. These requests may be used for testing the link or passing vendor specific information using the optional data field. L2CAP entities MUST respond to well-formed Echo Request packets with an Echo Response packet. The Data field is optional and implementation-dependent. L2CAP entities should ignore the contents of this field.

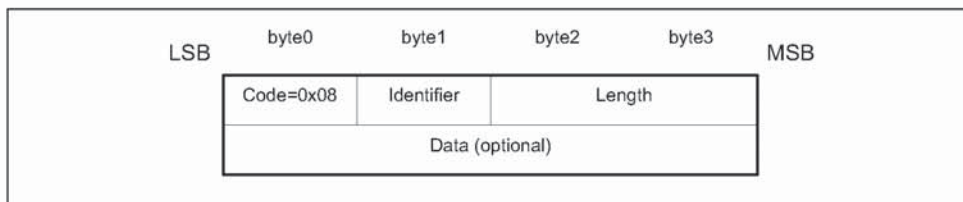


Figure 5.12: Echo Request Packet

5.9 ECHO RESPONSE (CODE 0x09)

Echo responses are sent upon receiving Echo Request packets. The identifier in the response **MUST** match the identifier sent in the Request. The optional and implementation-dependent data field may contain the contents of the data field in the Request, different data, or no data at all.

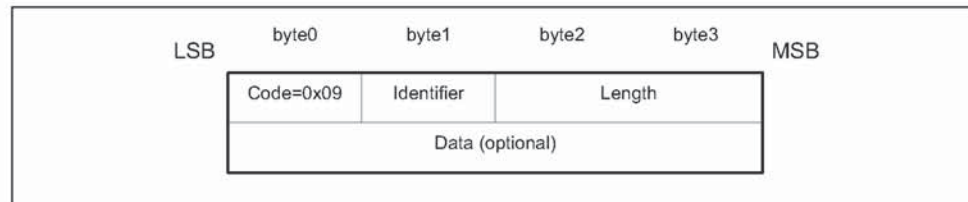


Figure 5.13: Echo Response Packet

5.10 INFORMATION REQUEST (CODE 0x0A)

Information requests are used to solicit implementation-specific information from a remote L2CAP entity. L2CAP entities **MUST** respond to well-formed Information Request packets with an Information Response packet.

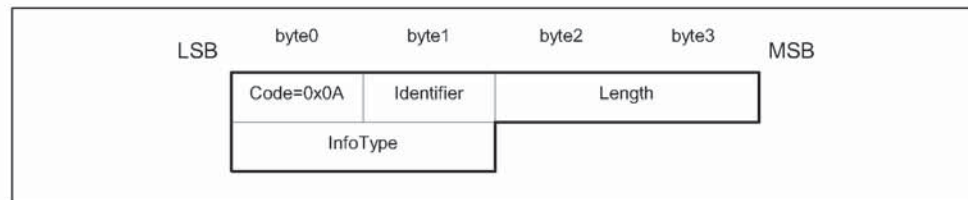


Figure 5.14: Information Request Packet

- *Length = 0x0002 octets*
- *InfoType: 2 octets*

The InfoType defines the type of implementation-specific information being solicited.

Value	Description
0x0001	Connectionless MTU
Other	Reserved

Table 5.8: InfoType definitions

5.11 INFORMATION RESPONSE (CODE 0X0B)

Information responses are sent upon receiving Information Request packets. The identifier in the response **MUST** match the identifier sent in the Request. The optional data field may contain the contents of the data field in the Request, different data, or no data at all.

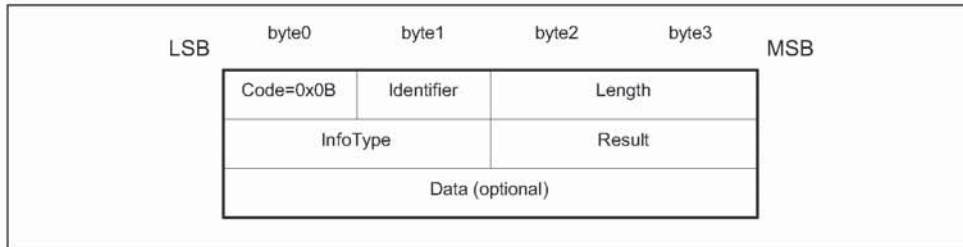


Figure 5.15: Information Response Packet

- **InfoType: 2 octets**

Same value sent in the request.

- **Result: 2 octets**

The Result contains information about the success of the request. If result is "Success", the data field contains the information as specified in Table 5.10. If result is "Not supported", no data should be returned.

Value	Description
0x0000	Success
0x0001	Not supported
Other	Reserved

Table 5.9: Information Response Result values

- **Data: 0 or more octets**

The contents of the Data field depends on the InfoType. For the Connection MTU request, the data field contains the remote entity's 2-octet acceptable connectionless MTU.

InfoType	Data	Data Length (in octets)
0x0001	Connectionless MTU	2

Table 5.10: Information Response Data fields

6 CONFIGURATION PARAMETER OPTIONS

Options are a mechanism to extend the ability to negotiate different connection requirements. Options are transmitted in the form of information elements comprised an option type, an option length, and one or more option data fields. Figure 6.1 illustrates the format of an option.



Figure 6.1: Configuration option format

- *Type: 1 octet*

The option type field defines the parameters being configured. The most significant bit of the type determines the action taken if the option is not recognized. The semantics assigned to the bit are defined below.

0 - option must be recognized; refuse the configuration request

1 - option is a hint; skip the option and continue processing

- *Length: 1 octet*

The length field defines the number of octets in the option payload. So an option type with no payload has a length of 0.

- *Option data*

The contents of this field are dependent on the option type.

6.1 MAXIMUM TRANSMISSION UNIT (MTU)

This option specifies the payload size the sender is capable of accepting. The type is 0x01, and the payload length is 2 bytes, carrying the two-octet MTU size value as the only information element (see Figure 6.2 on page 290).

Since all L2CAP implementations are capable to support a minimum L2CAP packet size, see Section 4 on page 272, MTU is not really a negotiated value but rather an informational parameter to the remote device that the local device can accommodate in this channel an MTU larger than the minimum required. In the unlikely case that the remote device is only willing to send L2CAP packets in this channel that are larger than the MTU announced by the local device, then this Configuration Request will receive a negative response in which the remote device will include the value of MTU that is indented to transmit. In this case, it is implementation specific on whether the local device will continue the configuration process or even maintain this channel.

The remote device in its positive Configuration Response will include the actual MTU to be used on this channel for traffic flowing into the local device which is

minimum{ MTU in configReq, outgoing MTU capability of remote device }. The MTU to be used on this channel but for the traffic flowing in the opposite direction will be established when the remote device (with respect to this discussion) sends its own Configuration Request as explained in Section 5.4 on page 280.

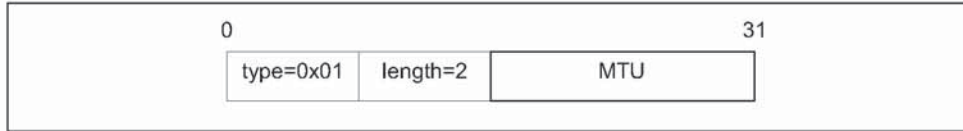


Figure 6.2: MTU Option Format

- **Maximum Transmission Unit (MTU) Size: 2 octets**
 The MTU field represents the largest L2CAP packet payload, in bytes, that the originator of the Request can accept for that channel. The MTU is asymmetric and the sender of the Request shall specify the MTU it can receive on this channel if it differs from the default value. L2CAP implementations must support a minimum MTU size of 48 bytes. The default value is 672 bytes¹.

6.2 FLUSH TIMEOUT OPTION

This option is used to inform the recipient of the amount of time the originator's link controller / link manager will attempt to successfully transmit an L2CAP segment before giving up and flushing the packet. The type is 0x02 and the payload size is 2 octets.

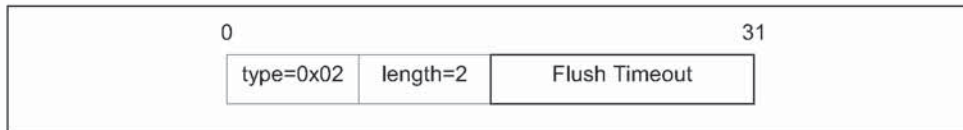


Figure 6.3: Flush Timeout

- **Flush Timeout**
 This value represents units of time measured in milliseconds. The value of 1 implies no retransmissions at the Baseband level should be performed since the minimum polling interval is 1.25 ms. The value of all 1's indicates an infinite amount of retransmissions. This is also referred to as 'reliable channel'. In this case, the link manager shall continue retransmitting a segment until physical link loss occurs. This is an asymmetric value and the sender of the Request shall specify its flush timeout value if it differs from the default value of 0xFFFF.

1. The default MTU was selected based on the payload carried by two Baseband DH5 packets (2*341=682) minus the Baseband ACL headers (2*2=4) and L2CAP header (6).

6.3 QUALITY OF SERVICE (QOS) OPTION

This option specifies a flow specification (flowSpec) similar to RFC 1363 [11]. If no QoS configuration parameter is negotiated the link should assume the default parameters discussed below. The QoS option is type 0x03.

When included in a Configuration Request, this option describes the outgoing traffic flow from the device sending the request to the device receiving it. When included in a positive Configuration Response, this option describes the incoming traffic flow agreement as seen from the device sending the response. When included in a negative Configuration Response, this option describes the preferred incoming traffic flow from the perspective of the device sending the response.

L2CAP implementations are only required to support 'Best Effort' service, support for any other service type is optional. Best Effort does not require any guarantees. If no QoS option is placed in the request, Best Effort must be assumed. If any QoS guarantees are required then a QoS configuration request must be sent.

The remote device places information that depends on the value of the result field, see Section 5.5 on page 283, in its Configuration Response. If the request was for Guaranteed Service, the response shall include specific values for any wild card parameters (see Token Rate and Token Bucket Size descriptions) contained in the request. If the result is "Failure – unacceptable parameters", the response may include a list of outgoing flowspec parameters and parameter values that would make a new Connection Request from the local device acceptable by the remote device. Both explicitly referenced in a Configuration Request or implied configuration parameters can be included in a Configuration Response. Recall that any missing configuration parameters from a Configuration Request are assumed to have their most recently (mutually) accepted values. For both Best effort and Guaranteed service, when the QoS option appears in the Configuration Response, "do not cares" shall be present where they appeared in the Configuration Request.

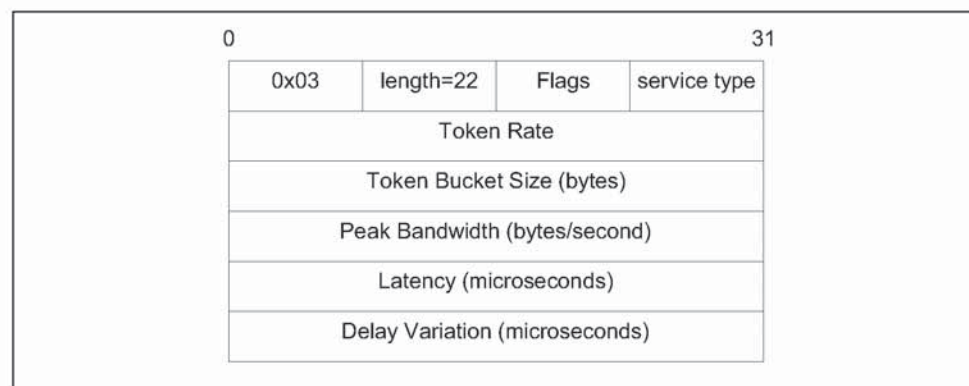


Figure 6.4: Quality of Service Flow Specification

- *Flags: 1 octet*

Reserved for future use and must be set to 0.

- *Service Type: 1 octet*

This field indicates the level of service required. Table 6.1 defines the different services available. If 'No traffic' is selected, the remainder of the fields may be ignored because there is no data being sent across the channel in the outgoing direction.

If 'Best effort', the default value, is selected, the remaining fields should be treated as hints by the remote device. The remote device may choose to ignore the fields, try to satisfy the hint but provide no response (QoS option omitted in the Response message), or respond with the settings it will try to meet.

Value	Description
0x00	No traffic
0x01	Best effort (Default)
0x02	Guaranteed
Other	Reserved

Table 6.1: Service type definitions

- *Token Rate: 4 octets*

The value of this field represents the rate at which traffic credits are granted in bytes per second. An application may send data at this rate continuously. Burst data may be sent up to the token bucket size (see below). Until that data burst has been drained, an application must limit itself to the token rate. The value 0x00000000 indicates no token rate is specified. This is the default value and implies indifference to token rate. The value 0xFFFFFFFF represents a wild card matching the maximum token rate available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value is a hint that the application wants as much bandwidth as possible. For Guaranteed service the value represents the maximum bandwidth available at the time of the request.

- *Token Bucket Size: 4 octets*

The value of this field represents the size of the token bucket in bytes. If the bucket is full, then applications must either wait or discard data. The value of 0x00000000 represents no token bucket is needed; this is the default value. The value 0xFFFFFFFF represents a wild card matching the maximum token bucket available. The meaning of this value depends on the semantics associated with the service type. For best effort, the value indicates the application wants a bucket as big as possible. For Guaranteed service the value represents the maximum buffer space available at the time of the request.

- *Peak Bandwidth: 4 octets*

The value of this field, expressed in bytes per second, limits how fast packets may be sent back-to-back from applications. Some intermediate systems can take advantage of this information, resulting in more efficient resource allocation. The value of 0x00000000 states that the maximum bandwidth is unknown, which is the default value.

- *Latency: 4 octets*

The value of this field represents the maximum acceptable delay between transmission of a bit by the sender and its initial transmission over the air, expressed in microseconds. The precise interpretation of this number depends on the level of guarantee specified in the Class of Service. The value 0xFFFFFFFF represents a do not care and is the default value.

- *Delay Variation: 4 octets*

The value of this field is the difference, in microseconds, between the maximum and minimum possible delay that a packet will experience. This value is used by applications to determine the amount of buffer space needed at the receiving side in order to restore the original data transmission pattern. The value 0xFFFFFFFF represents a do not care and is the default value.

6.4 CONFIGURATION PROCESS

Negotiating the channel parameters involves three steps:

1. Informing the remote side of the non-default parameters that the local side will accept
2. Having the remote side agreeing or disagreeing to these values (including the default ones); steps (1) and (2) may iterate as needed
3. Repeat steps (1) and (2) for the reverse direction from the (previous) remote side to the (previous) local side.

This process can be abstracted into a Request negotiation path and a Response negotiation path.

6.4.1 Request Path

The Request Path negotiates the incoming MTU, flush timeout, and outgoing flowspec. Table 6.2 defines the configuration options that may be placed in the Configuration Request message and their semantics.

Parameter	Description
MTU	Incoming MTU information
FlushTO	Outgoing flush timeout
OutFlow	Outgoing flow information.

Table 6.2: Parameters allowed in Request

6.4.2 Response Path

The Response Path negotiates the outgoing MTU (remote side's incoming MTU), the remote side's flush timeout, and incoming flowspec (remote side's outgoing flowspec). If a request-oriented parameter is not present in the Request message (reverts to default value), the remote side may negotiate for a non-default value by including the proposed value in a negative Response message.

Parameter	Description
MTU	Outgoing MTU information
FlushTO	Incoming flush timeout
InFlow	Incoming flow information

Table 6.3: Parameters allowed in Response

6.4.3 Configuration State Machine

The configuration state machine shown below depicts two paths. Before leaving the CONFIG state and moving into the OPEN state, both paths must reach closure. The request path requires the local device to receive a positive response to reach closure while the response path requires the local device to send a positive response to reach closure.

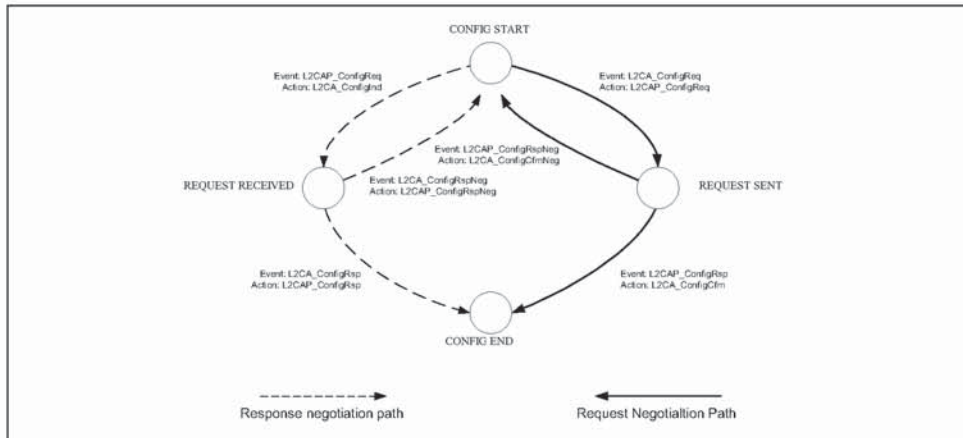


Figure 6.5: Configuration State Machine

"Appendix A: Configuration MSCs" on page 318 provides some configuration examples.

7 SERVICE PRIMITIVES

This section presents an abstract description of the services offered by L2CAP in terms of service primitives and parameters. The service interface is required for testing. The interface is described independently of any platform specific implementation. All data values use Little Endian byte ordering.

7.1 EVENT INDICATION

Service	Input Parameters	Output Parameters
EventIndication	Event, Callback	Result

Description:

The use of this primitive requests a callback when the selected indication Event occurs.

Input Parameters:

Event *Type: uint* *Size: 2 octets*

Value	Description
0x00	Reserved
0x01	L2CA_ConnectInd
0x02	L2CA_ConfigInd
0x03	L2CA_DisconnectInd
0x04	L2CA_QoSViolationInd
other	Reserved for future use

Callback *Type: function* *Size: N/A*

Event	Callback Function Input Parameters
L2CA_ConnectInd	BD_ADDR, CID, PSM, Identifier
L2CA_ConfigInd	CID, OutMTU, InFlow, InFlushTO
L2CA_DisconnectInd	CID
L2CA_QoSViolationInd	BD_ADDR

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Event successfully registered
0x0001	Event registration failed

7.1.1 L2CA_ConnectInd Callback

This callback function includes the parameters for the address of the remote device that issued the connection request, the local CID representing the channel being requested, the Identifier contained in the request, and the PSM value the request is targeting.

7.1.2 L2CA_ConfigInd Callback

This callback function includes the parameters indicating the local CID of the channel the request has been sent to, the outgoing MTU size (maximum packet that can be sent across the channel) and the flowspec describing the characteristics of the incoming data. All other channel parameters are set to their default values if not provided by the remote device.

7.1.3 L2CA_DisconnectInd Callback

This callback function includes the parameter indicating the local CID the request has been sent to.

7.1.4 L2CA_QoSViolationInd Callback

This callback function includes the parameter indicating the address of the remote Bluetooth device where the QoS contract has been violated.

7.2 CONNECT

Service	Input Parameters	Output Parameters
L2CA_ConnectReq	PSM, BD_ADDR	LCID, Result, Status

Description:

This primitive initiates the sending of an L2CA_ConnectReq message and blocks until a corresponding L2CA_ConnectCfm(Neg) or L2CA_TimeOutInd message is received.

The use of this primitive requests the creation of a channel representing a logical connection to a physical address. Input parameters are the target protocol (*PSM*) and remote device's 48-bit address (*BD_ADDR*). Output parameters are the local CID (*LCID*) allocated by the local L2CAP entity, and *Result* of the request. If the *Result* indicates success, the *LCID* value contains the identification of the local endpoint. Otherwise the *LCID* returned should be set to 0. If *Result* indicates a pending notification, the *Status* value may contain more information of what processing is delaying the establishment of the connection. Otherwise the *Status* value should be ignored.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Target PSM provided for the connection

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Unique Bluetooth address of target device

Output Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel if Result = 0x0000, otherwise set to 0.

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful and the CID identifies the local endpoint. Ignore Status parameter
0x0001	Connection pending. Check Status parameter for more information
0x0002	Connection refused because no service for the PSM has been registered
0x0003	Connection refused because the security architecture on the remote side has denied the request
0xEEEE	Connection timeout occurred. This is a result of a timer expiration indication being included in the connection confirm message

Status *Type: uint* *Size: 2 octets*

Value	Description
0x0000	No further information
0x0001	Authentication pending
0x0002	Authorization pending

7.3 CONNECT RESPONSE

Service	Input Parameters	Output Parameters
L2CA_ConnectRsp	BD_ADDR, Identifier, LCID, Response, Status	Result

Description:

This primitive represents the L2CA_ConnectRsp.

The use of this primitive issues a response to a connection request event indication. Input parameters are the remote device's 48-bit address, Identifier sent in the request, local CID, the Response code, and the Status attached to the Response code. The output parameter is the Result of the service request.

This primitive must be called no more than once after receiving the callback indication. This primitive returns once the local L2CAP entity has validated the request. A successful return does indicate the response has been sent over the air interface.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Unique Bluetooth address of target device

Identifier *Type: uint* *Size: 1 octets*

Value	Description
0xXX.	This value must match the value received in the L2CA_ConnectInd event described in Section 7.1.1 on page 296

LCID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

Response *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Connection successful
0x0001	Connection pending
0x0002	Connection refused – PSM not supported
0x0003	Connection refused – security block
0x0004	Connection refused – no resources available
0XXXXX	Other connection response code

Status *Type: uint* *Size: 2 octets*

Value	Description
0x0000	No further information available
0x0001	Authentication pending
0x0002	Authorization pending
0xXXXX	Other status code

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response successfully sent
0x0001	Failure to match any outstanding connection request

7.4 CONFIGURE

Service	Input Parameters	Output Parameters
L2CA_ConfigReq	CID, InMTU, OutFlow, OutFlushTO, LinkTO	Result, InMTU, OutFlow, OutFlushTO

Description:

This primitive initiates the sending of an L2CA_ConfigReq message and blocks until a corresponding L2CA_ConfigCfm(Neg) or L2CA_TimeOutInd message is received.

The use of this primitive requests the initial configuration (or reconfiguration) of a channel to a new set of channel parameters. Input parameters are the local CID endpoint, new incoming receivable MTU (InMTU), new outgoing flow specification, and flush and link timeouts. Output parameters composing the L2CA_ConfigCfm(Neg) message are the Result, accepted incoming MTU(InMTU), the remote side's flow requests, and flush and link timeouts. Note that the output results are returned only after the local L2CAP entity transitions out of the CONFIG state (even if this transition is back to the CONFIG state).

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local CID

InMTU

Type: *uint*

Size: 2 octets

Value	Description
0xXXXX	Maximum transmission unit this channel can accept

OutFlow

Type: *Flow*

Size: *x* octets

Value	Description
flowspec	Quality of service parameters dealing with the traffic characteristics of the outgoing data flow

OutFlushTO

Size 2 octets

Value	Description
0xXXXX	Number of milliseconds to wait before an L2CAP packet that cannot be acknowledged at the physical layer is dropped
0x0000	Request to use the existing flush timeout value if one exists, otherwise the default value (0xFFFF) will be used
0x0001	Perform no retransmissions at the Baseband layer
0xFFFF	Perform retransmission at the Baseband layer until the link timeout terminates the channel

LinkTO

Size 2 octets

Value	Description
0xXXXX	Number of milliseconds to wait before terminating an unresponsive link

Output Parameters:

Result

Size 2 octets

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Failure – invalid CID
0x0002	Failure – unacceptable parameters
0x0003	Failure – signalling MTU exceeded
0x0004	Failure – unknown options
0xEEEE	Configuration timeout occurred. This is a result of a timer expiration indication being included in the configuration confirm

InMTU

Size 2 octets

Value	Description
0xXXXX	Maximum transmission unit that the remote unit will send across this channel (maybe less or equal to the InMTU input parameter).

OutFlow

Size 2 octets

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of the agreed-upon outgoing data flow if Result is successful. Otherwise this represents the requested Quality of Service

OutFlushTO

Size 2 octets

Value	Description
0xXXXX	Number of milliseconds before an L2CAP packet that cannot be acknowledged at the physical layer is dropped. This value is informative of the actual value that will be used for outgoing packets. It may be less or equal to the OutFlushTO parameter given as input.

7.5 CONFIGURATION RESPONSE

Service	Input Parameters	Output Parameters
L2CA_ConfigRsp	CID, OutMTU, InFlow	Result

Description:

This primitive represents the L2CAP_ConfigRsp.

The use of this primitive issues a response to a configuration request event indication. Input parameters include the local CID of the endpoint being configured, outgoing transmit MTU (which may be equal or less to the OutMTU parameter in the L2CA_ConfigInd event) and the accepted flowspec for incoming traffic. The output parameter is the Result value.

Input Parameters:

LCID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Local channel identifier

OutMTU *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Maximum transmission unit this channel will send

InFlow *Type: Flow* *Size: x octets*

Value	Description
FlowSpec	Quality of service parameters dealing with the traffic characteristics of the incoming data flow

Output Parameters:

Result

Size 2 octets

Value	Description
0x0000	Configuration is successful. Parameters contain agreed upon values
0x0001	Configuration failed – unacceptable parameters
0x0002	Configuration failed – rejected
0x0003	Configuration failed – invalid CID
0x0004	Configuration failed – unknown options
0XXXXX	Reserved

7.6 DISCONNECT

Service	Input Parameters	Output Parameters
L2CA_DisconnectReq	CID	Result

Description:

This primitive represents the L2CAP_DisconnectReq and the returned output parameters represent the corresponding L2CAP_DisconnectRsp or the RTX timer expiration.

The use of this primitive requests the disconnection of the channel. Input parameter is the *CID* representing the local channel endpoint. Output parameter is *Result*. *Result* is zero if a L2CAP_DisconnectRsp is received, otherwise a non-zero value is returned. Once disconnection has been requested, no process will be able to successfully read or write from the CID. Writes in progress should continue to be processed.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Disconnection successful. This is a result of the receipt of a disconnection response message
0xEEEE	Disconnection timeout occurred.

7.7 WRITE

Service	Input Parameters	Output Parameters
L2CA_DataWrite	CID, Length, OutBuffer	Size, Result

Description:

The use of this primitive requests the transfer of data across the channel. If the length of the data exceeds the OutMTU then only the first OutMTU bytes are sent This command may be used for both connection-oriented and connection-less traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

Length *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Size, in bytes, of the buffer where data to be transmitted are stored

OutBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the input buffer used to store the message

Output Parameters:

Size *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	The number of bytes transferred

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful write
0x0001	Error – Flush timeout expired
0x0002	Error – Link termination (perhaps this should be left to the indication)

7.8 READ

Service	Input Parameters	Output Parameters
L2CA_DataRead	CID, Length, InBuffer	Result, N

Description:

The use of this primitive requests for the reception of data. This request returns when data is available or the link is terminated. The data returned represents a single L2CAP payload. If not enough data is available, the command will block until the data arrives or the link is terminated. If the payload is bigger than the buffer, only the portion of the payload that fits into the buffer will be returned, and the remainder of the payload will be discarded. This command may be used for both connection-oriented and connectionless traffic.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	CID

Length *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Size, in bytes, of the buffer where received data are to be stored

InBuffer *Type: pointer* *Size: N/A*

Value	Description
N/A	Address of the buffer used to store the message

Output parameters:

Result

Value	Description
0x0000	Success

N *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Number of bytes transferred to InBuffer

7.9 GROUP CREATE

Service	Input Parameters	Output Parameters
L2CA_GroupCreate	PSM	CID

Description:

The use of this primitive requests the creation of a CID to represent a logical connection to multiple devices. Input parameter is the *PSM* value that the outgoing connectionless traffic is labelled with, and the filter used for incoming traffic. Output parameter is the *CID* representing the local endpoint. On creation, the group is empty but incoming traffic destined for the PSM value is readable.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Protocol/service multiplexer value

Output Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

7.10 GROUP CLOSE

Service	Input Parameters	Output Parameters
L2CA_GroupClose	CID	Result

Description:

The use of this primitive closes down a Group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful closure of the channel
0x0001	Invalid CID

7.11 GROUP ADD MEMBER

Service	Input Parameters	Output Parameters
L2CA_GroupAddMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the addition of a member to a group. The input parameter includes the CID representing the group and the BD_ADDR of the group member to be added. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Remote device address

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure to establish connection to remote device
Other	Reserved

7.12 GROUP REMOVE MEMBER

Service	Input Parameters	Output Parameters
L2CA_GroupRemoveMember	CID, BD_ADDR	Result

Description:

The use of this primitive requests the removal of a member from a group. The input parameters include the CID representing the group and BD_ADDR of the group member to be removed. The output parameter Result confirms the success or failure of the request.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Channel ID representing local end-point of the communication channel

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Unique Bluetooth address device to be removed

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – device not a member of the group
Other	Reserved

7.13 GET GROUP MEMBERSHIP

Service	Input Parameters	Output Parameters
L2CA_GroupMembership	CID	Result, N, BD_ADDR_List

Description:

The use of this primitive requests a report of the members of a group. The input parameter CID represents the group being queried. The output parameter Result confirms the success or failure of the operation. If the Result is successful, BD_ADDR_List is a list of the Bluetooth addresses of the N members of the group.

Input Parameters:

CID *Type: uint* *Size: 2 octets*

Value	Description
0xXXXX	Channel ID representing local end-point of the communication channel

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Success
0x0001	Failure – group does not exist
Other	Reserved

N *Type: uint* *Size: 2 octets*

Value	Description
0x0000-0xFFFF	The number of devices in the group identified by the channel end-point CID. If Result indicates failure, N should be set to 0

| *BD_ADDR_List* *Type: pointer* *Size: N/A*

Value	Description
0XXXXXXXXXXXXX	List of N unique Bluetooth addresses of the devices in the group identified by the channel end-point CID. If Result indicates failure, the all-zero address is the only address that should be returned

7.14 PING

Service	Input Parameters	Output Parameters
L2CA_Ping	BD_ADDR, ECHO_DATA, Length	Result, ECHO_DATA, Size

Description:

This primitive represents the initiation of an L2CA_EchoReq command and the reception of the corresponding L2CA_EchoRsp command.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Unique Bluetooth address of target device.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents to be transmitted in the data payload of the Echo Request command.

Length *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Size, in bytes, of the data in the buffer.

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received.
0x0001	Timeout occurred.

ECHO_DATA *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Echo Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Size, in bytes, of the data in the buffer.

7.15 GETINFO

Service	Input Parameters	Output Parameters
L2CA_GetInfo	BD_ADDR, InfoType	Result, InfoData, Size

Description:

This primitive represents the initiation of an L2CA_InfoReq command and the reception of the corresponding L2CA_InfoRsp command.

Input Parameters:

BD_ADDR *Type: uint* *Size: 6 octets*

Value	Description
0XXXXXXXXXXXXX	Unique Bluetooth address of target device

InfoType *Type: uint* *Size: 2 octets*

Value	Description
0x0001	Maximum connectionless MTU size

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Response received
0x0001	Not supported
0x0002	Informational PDU rejected, not supported by remote device
0x0003	Timeout occurred

InfoData *Type: pointer* *Size: N/A*

Value	Description
N/A	The buffer containing the contents received in the data payload of the Information Response command.

Size *Type: uint* *Size: 2 octets*

Value	Description
0XXXXX	Size, in bytes, of the data in the InfoData buffer.

7.16 DISABLE CONNECTIONLESS TRAFFIC

Service	Input Parameters	Output Parameters
L2CA_DisableCLT	PSM	Result

Description:

General request to disable the reception of connectionless packets. The input parameter is the *PSM* value indicating service that should be blocked. This command may be used to incrementally disable a set of PSM values. The use of the 'invalid' PSM 0x0000 blocks all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general blocking rather than PSM-specific blocks and would fail to block a single non-zero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Block all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to be blocked

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

7.17 ENABLE CONNECTIONLESS TRAFFIC

Service	Input Parameters	Output Parameters
L2CA_EnableCLT	PSM	Result

Description:

General request to enable the reception of connectionless packets. The input parameter is the *PSM* value indicating the service that should be unblocked. This command may be used to incrementally enable a set of PSM values. The use of the 'invalid' PSM 0x0000 enables all connectionless traffic. The output parameter *Result* indicates the success or failure of the command. A limited device might support only general enabling rather than PSM-specific filters, and would fail to enable a single non-zero PSM value.

Input Parameters:

PSM *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Enable all connectionless traffic
0xXXXX	Protocol/Service Multiplexer field to enable

Output Parameters:

Result *Type: uint* *Size: 2 octets*

Value	Description
0x0000	Successful
0x0001	Failure – not supported

8 SUMMARY

The Logical Link Control and Adaptation Protocol (L2CAP) is one of two link level protocols running over the Baseband. L2CAP is responsible for higher level protocol multiplexing, MTU abstraction, group management, and conveying quality of service information to the link level.

Protocol multiplexing is supported by defining channels. Each channel is bound to a single protocol in a many-to-one fashion. Multiple channels can be bound to the same protocol, but a channel cannot be bound to multiple protocols. Each L2CAP packet received on a channel is directed to the appropriate higher level protocol.

L2CAP abstracts the variable-sized packets used by the Baseband Protocol (page 33). It supports large packet sizes up to 64 kilobytes using a low-overhead segmentation-and-reassembly mechanism.

Group management provides the abstraction of a group of units allowing more efficient mapping between groups and members of the Bluetooth piconet. Group communication is connectionless and unreliable. When composed of only a pair of units, groups provide connectionless channel alternative to L2CAP's connection-oriented channel.

L2CAP conveys QoS information across channels and provides some admission control to prevent additional channels from violating existing QoS contracts.

9 REFERENCES

- [1] Internet Engineering Task Force, "A Proposed Flow Specification", RFC 1363, September 1992.

10 LIST OF FIGURES

Figure 1.1:	L2CAP within protocol layers	249
Figure 1.2:	ACL Payload Header for single-slot packets.....	250
Figure 1.3:	ACL Payload Header for multi-slot packets	250
Figure 1.4:	L2CAP in Bluetooth Protocol Architecture	251
Figure 2.1:	Channels between devices	254
Figure 2.2:	L2CAP Architecture.....	255
Figure 2.3:	L2CAP SAR Variables.....	255
Figure 2.4:	L2CAP segmentation	256
Figure 2.5:	Segmentation and Reassembly Services in a unit with an HCI	257
Figure 3.1:	L2CAP Layer Interactions	258
Figure 3.2:	MSC of Layer Interactions.....	259
Figure 3.3:	State Machine Example	270
Figure 3.4:	Message Sequence Chart of Basic Operation.....	271
Figure 4.1:	L2CAP Packet (field sizes in bits).....	272
Figure 4.2:	Connectionless Packet.....	273
Figure 5.1:	Signalling Command Packet Format.....	275
Figure 5.2:	Command format	275
Figure 5.3:	Command Reject Packet	277
Figure 5.4:	Connection Request Packet.....	278
Figure 5.5:	Connection Response Packet.....	279
Figure 5.6:	Configuration Request Packet	281
Figure 5.7:	Configuration Request Flags field format.....	281
Figure 5.8:	Configuration Response Packet.....	283
Figure 5.9:	Configuration Response Flags field format.....	283
Figure 5.10:	Disconnection Request Packet	285
Figure 5.11:	Disconnection Response Packet	286
Figure 5.12:	Echo Request Packet	286
Figure 5.13:	Echo Response Packet.....	287
Figure 5.14:	Information Request Packet.....	287
Figure 5.15:	Information Response Packet.....	288
Figure 6.1:	Configuration option format.....	289
Figure 6.2:	MTU Option Format	290
Figure 6.3:	Flush Timeout	290
Figure 6.4:	Quality of Service Flow Specification	291
Figure 6.5:	Configuration State Machine.....	294
Figure I	Basic MTU exchange	318
Figure II	Dealing with Unknown Options	319
Figure III	Unsuccessful Configuration Request	320

11 LIST OF TABLES

Table 1.1:	Logical channel L_CH field contents	250
Table 2.1:	CID Definitions	253
Table 2.2:	Types of Channel Identifiers	254
Table 3.1:	L2CAP Channel State Machine	267
Table 5.1:	Signalling Command Codes	276
Table 5.2:	Reason Code Descriptions	277
Table 5.3:	Reason Data values	278
Table 5.4:	Defined PSM Values	278
Table 5.5:	Result values	280
Table 5.6:	Status values	280
Table 5.7:	Configuration Response Result codes	284
Table 5.8:	InfoType definitions	287
Table 5.9:	Information Response Result values	288
Table 5.10:	Information Response Data fields	288
Table 6.1:	Service type definitions	292
Table 6.2:	Parameters allowed in Request	293
Table 6.3:	Parameters allowed in Response	294
Table I	Result of Second Link Timeout Request	322
Table II	Result of Second Flush Timeout Request	322

TERMS AND ABBREVIATIONS

Baseband	Baseband Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IrDA	Infra-red Data Association
L_CH	Logical Channel
LC	Link Controller
LM	Link Manager
LMP	Link Manager Protocol
MTU	Maximum Transmission Unit
PPP	Point-to-Point Protocol
Reliable	Characteristic of an L2CAP channel that has an infinite flush timeout
RFC	Request For Comments
SAR	Segmentation and Reassembly

APPENDIX A: CONFIGURATION MSCs

The examples in this appendix describe a sample of the multiple possible configuration scenarios that might occur. Currently, these are provided as suggestions and may change in the next update of the Specification.

Figure I illustrates the basic configuration process. In this example, the devices exchange MTU information. All other values are assumed to be default.

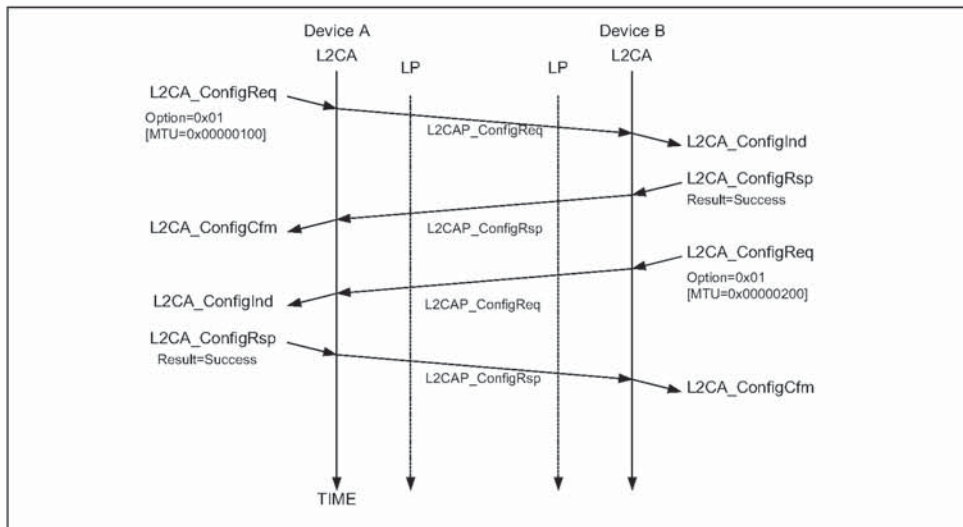


Figure I: Basic MTU exchange

Figure II on page 319 illustrates how two devices interoperate even though one device supports more options than the other does. Device A is an upgraded version. It uses a hypothetically defined option type 0x20 for link-level security. Device B rejects the command using the Configuration Response packet with result 'unknown parameter' informing Device A that option 0x20 is not understood. Device A then resends the request omitting option 0x20. Device B notices that it does not need to such a large MTU and accepts the request but includes in the response the MTU option informing Device A that Device B will not send an L2CAP packet with a payload larger than 0x80 octets over this channel. On receipt of the response, Device A could reduce the buffer allocated to hold incoming traffic.

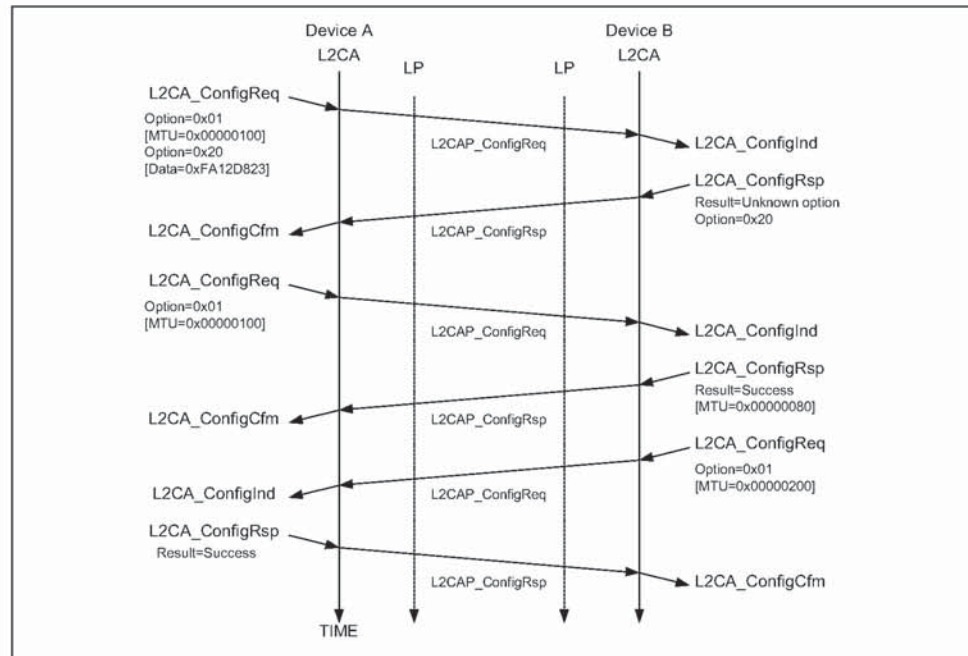


Figure II: Dealing with Unknown Options

Figure III on page 320 illustrates an unsuccessful configuration request. There are two problems described by this example. The first problem is that the configuration request is placed in an L2CAP packet that cannot be accepted by the remote device, due to its size. The remote device informs the sender of this problem using the Command Reject message. Device A then resends the configuration options using two smaller L2CAP_ConfigReq messages.

The second problem is an attempt to configure a channel with an invalid CID. For example device B may not have an open connection on that CID (0x01234567 in this example case).

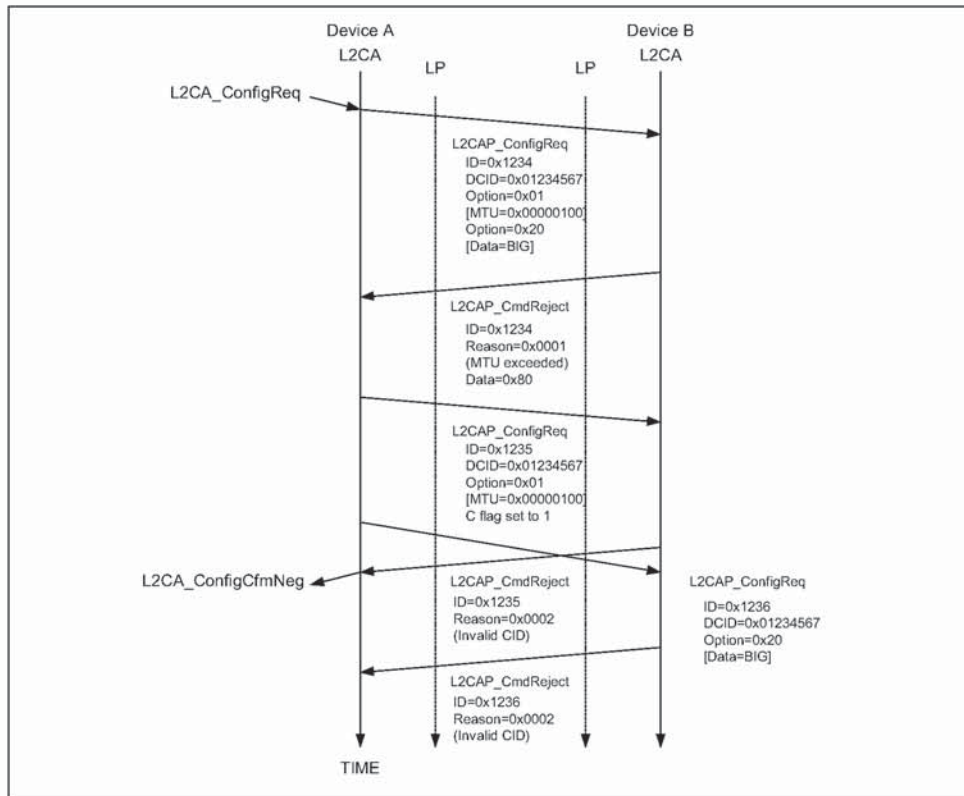


Figure III: Unsuccessful Configuration Request

APPENDIX B: IMPLEMENTATION GUIDELINES

This section contains some guidelines for implementations. These guidelines are not part of the compliance tests. At the moment they are simply suggestions on how to solve some difficult problems.

RTX TIMER

Implementations should not start this timer on an L2CAP Connection Request packet unless the physical link has been established. Otherwise the Baseband paging mechanism might increase the cost of the request beyond that of the minimal timeout value. If an implementation performs some form of security check it is recommended that the connection pending response be sent back prior to any consultation with a security manager that might perform Baseband authentication commands. If any security check requires user interaction, the link might timeout waiting for the user to enter a PIN.

QOS MAPPING TO LM AND L2CAP IMPLEMENTATIONS

Token Rate

The Link Manager (LM) should ensure data is removed from the transmission buffer at this rate. The LM should ensure the polling interval is fast enough to support this data rate. The polling interval should be adjusted if the packet type changes. If the buffer overflows, and the service type is Guaranteed, a QoS violation should be reported. If the service type is Best Effort, and a Token Rate was non-zero, a QoS violation should also be reported.

Given a Token Rate of 0xFFFFFFFF, and Service Type of Guaranteed, the LM should refuse any additional connections from remote devices and disable all periodic scans.

Token Bucket Size

L2CAP implementations should ensure that a buffer meeting the size request is allocated for the channel. If no buffer is available, and the service type is Guaranteed, the request should be rejected. If no appropriately sized buffer is available, and the service type is Best Effort, the largest available buffer should be allocated.

Peak Bandwidth

If the token bucket buffer overflows, a QoS violation should be raised.

Latency

The LM should ensure the polling interval is at least this value. If the polling interval necessary to support the token rate is less than this value, the smaller interval should be used. If this interval cannot be supported, a QoS violation should be raised.

Delay Variation

The LM may ignore this value because there is no clear mapping between L2CAP packet delays and the necessary polling interval without requiring the LM to comprehend the length field in L2CAP packets.

COLLISION TABLES

Current Value	Requested Value	Result
X	X	X
X	Y	If $(X < Y)$ then X, else Y

Table I: Result of Second Link Timeout Request

Current Value	Requested Value	Result
N	0	N
N	N	N
N	$M \neq N$	Reject

Table II: Result of Second Flush Timeout Request

Part E

**SERVICE DISCOVERY
PROTOCOL (SDP)**

This specification defines a protocol for locating services provided by or available through a Bluetooth device.



CONTENTS

1 Introduction327

1.1 General Description327

1.2 Motivation.....327

1.3 Requirements.....327

1.4 Non-requirements and Deferred Requirements328

1.5 Conventions329

1.5.1 Bit And Byte Ordering Conventions.....329

2 Overview330

2.1 SDP Client-Server Interaction330

2.2 Service Record.....332

2.3 Service Attribute.....334

2.4 Attribute ID335

2.5 Attribute Value.....335

2.6 Service Class336

2.6.1 A Printer Service Class Example336

2.7 Searching for Services337

2.7.1 UUID.....337

2.7.2 Service Search Patterns.....338

2.8 Browsing for Services338

2.8.1 Example Service Browsing Hierarchy339

3 Data Representation341

3.1 Data Element341

3.2 Data Element Type Descriptor341

3.3 Data Element Size Descriptor342

3.4 Data Element Examples.....343

4 Protocol Description344

4.1 Transfer Byte Order344

4.2 Protocol Data Unit Format.....344

4.3 Partial Responses and Continuation State346

4.4 Error Handling346

4.4.1 SDP_ErrorResponse PDU347

4.5 ServiceSearch Transaction348

4.5.1 SDP_ServiceSearchRequest PDU.....348

4.5.2 SDP_ServiceSearchResponse PDU.....349

4.6 ServiceAttribute Transaction351

4.6.1 SDP_ServiceAttributeRequest PDU.....351

4.6.2 SDP_ServiceAttributeResponse PDU.....352

4.7	ServiceSearchAttribute Transaction	354
4.7.1	SDP_ServiceSearchAttributeRequest PDU	354
4.7.2	SDP_ServiceSearchAttributeResponse PDU	356
5	Service Attribute Definitions.....	358
5.1	Universal Attribute Definitions.....	358
5.1.1	ServiceRecordHandle Attribute.....	358
5.1.2	ServiceClassIDList Attribute.....	359
5.1.3	ServiceRecordState Attribute.....	359
5.1.4	ServiceID Attribute	359
5.1.5	ProtocolDescriptorList Attribute.....	360
5.1.6	BrowseGroupList Attribute	361
5.1.7	LanguageBaseAttributeIDList Attribute	361
5.1.8	ServiceInfoTimeToLive Attribute	362
5.1.9	ServiceAvailability Attribute.....	363
5.1.10	BluetoothProfileDescriptorList Attribute	363
5.1.11	DocumentationURL Attribute	364
5.1.12	ClientExecutableURL Attribute.....	364
5.1.13	IconURL Attribute.....	365
5.1.14	ServiceName Attribute	365
5.1.15	ServiceDescription Attribute.....	366
5.1.16	ProviderName Attribute.....	366
5.1.17	Reserved Universal Attribute IDs.....	366
5.2	ServiceDiscoveryServer Service Class Attribute Definitions ...	367
5.2.1	ServiceRecordHandle Attribute.....	367
5.2.2	ServiceClassIDList Attribute.....	367
5.2.3	VersionNumberList Attribute	367
5.2.4	ServiceDatabaseState Attribute.....	368
5.2.5	Reserved Attribute IDs.....	368
5.3	BrowseGroupDescriptor Service Class Attribute Definitions ...	369
5.3.1	ServiceClassIDList Attribute.....	369
5.3.2	GroupID Attribute	369
5.3.3	Reserved Attribute IDs.....	369
Appendix A – Background Information		370
Appendix B – Example SDP Transactions		371

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See "Appendix A – Background Information," on page 370, for further information on this topic.

1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

1.5 CONVENTIONS

1.5.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right.

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See Section 4.1 Transfer Byte Order on page 344.

2 OVERVIEW

2.1 SDP CLIENT-SERVER INTERACTION

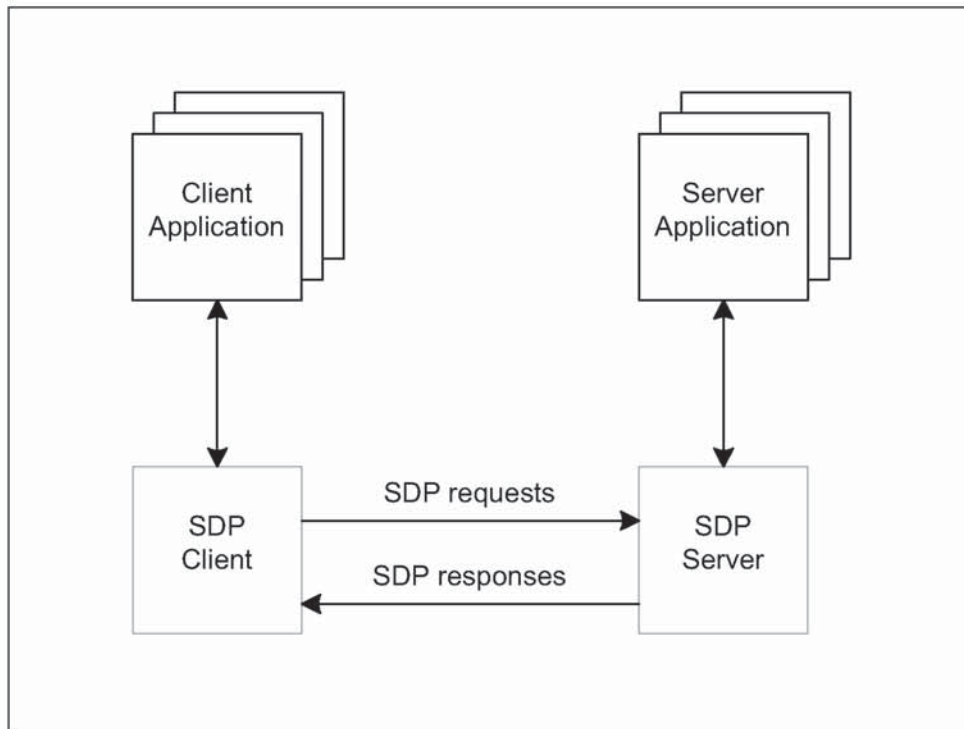


Figure 2.1:

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 1 may be simplified to that shown in Figure 2.

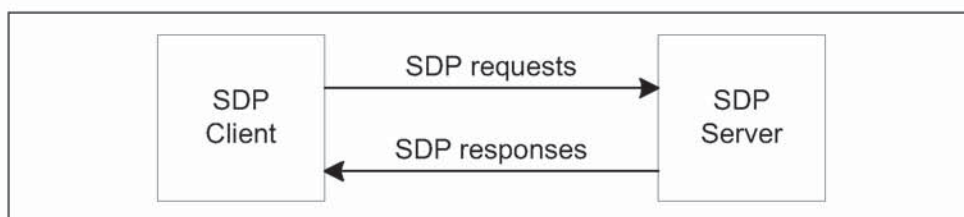


Figure 2.2: