# Mesh Profile

## *Bluetooth*® Change Request: Directed Forwarding

- **Revision:** 7
- **Revision Date:** 2018-Aug-14
- **Group Prepared By:** Mesh Working Group
- **Feedback Email:** mesh-main@bluetooth.org

This Change Request proposes changes to the following specification:

- Mesh_v1.0.docx ("Bluetooth Mesh Profile Specification, Version 1.0")

**Abstract:**
This change proposes the introduction of a Directed Forwarding feature in Bluetooth mesh.

### *CR Revision History*

| Revision Number | Date | Comments |
|---|---|---|
| r00 | yyyy-mmm-dd | Initial draft. |
| r01 | 2018-may-18 | In the Forwarding Table state, renamed the Updated field as ProactivePathUpdated and the Validated field as BackwardPathValidated. Added a table for the values of the ProactivePathUpdated field. Modified the formats of PERR and of PTREP. Modified the description of the Directed Forwarding control messages in the summary table of the Transport Control messages. Modified terminology in sections of Directed Forwarding control messages and operations. Restructured the sections of Forwarding Table and Discovery Table based on the type of received message. Removed note on flushing a Forwarding Table if Path Metric is modified. Specified the classes of address for a Path Destination. Specified the default value of the Directed Forwarding state. Added a new table for the DF Forwarding Table List message and renamed the Index field as FirstIndex associated with only the first path entry. Specified that all the accesses to a path entry performed for Directed Forwarding operations shall be executed only if it is a non-fixed entry. Fixed clauses to trigger a path error notification. Specified Directed Forwarding behavior for multiple subnets. Merged DF Forwarding Table Add and Update messages. Added a table for the Status Codes associated with the Forwarding Table state. |
| r02 | 2018-jun-04 | Introduced definition of the Bearer Index and added BTO and BTD fields in the configuration messages. Added check to NT for NINFO and PTREQ. Incorporated text for link-specific retransmissions. Reviewed text in Directed Forwarding Tracing Operation and changed “path indicated as invalid” to “path is removed”. Reviewed behavior of LPN neighbors. Added clarification in Forwarding Table section addressing duplicated FT entries with bi-directional path discovery. |
| r03 | 2018-jun-13 | Specified more details about combined support of Low Power feature and of Directed Forwarding feature. Added definition of unassigned bearer index. Specified that filtering after updating the Neighbor Table is applied to all Directed Forwarding control messages. Specified that when the NetKey of a subnet is deleted, the states associated with the Directed Forwarding feature in the subnet are reset to their default values. Replaced LastIndex with IndexRange in DF Forwarding Table Get message. Aligned Path Tracing flow chart with edited text. Specified how the RET value in Neighbor Table is used in case of forwarding toward all types of address. Removed PMT and PLT fields from PREQ, |

| Revision Number | Date | Comments |
|---|---|---|
| | | PTREQ, NINFO, and Discovery Table. Added recommendation to configure Directed Forwarding nodes of the same subnet in order to use the same path metric and path lifetime. Added conditions to find a path entry for message forwarding in presence of delegate nodes. |
| r04 | 2018-jun-22 | Incorporated Directed Forwarding Assistance operation |
| | 2018-jun-23 | Added Path Request Transmit state and associated messages in the Directed Forwarding Server and Client models. Redrawn topology figure. |
| r05 | 2018-jul-02 | Reviewed the DF Assistance operation. Modified the format of PCONF, PTREQ, PTREP, PAREQ, and PAREP. Modified the Forwarding Table entry search conditions when receiving any of PREP, PCONF, PTREQ, PAREQ, and PAREP. Modified the Discovery Table entry search conditions when receiving any of PTREQ or PTREP. Reviewed the meaning of the EA flag in PREQ and in PREP and modified the values of OAR and DAR in the messages that are stored. Added a section with examples of forward and backward path entries. |
| | 2018-jul-04 | Clarified that messages sent by a LPN to be directed by the associated Friend node toward a Path Destination are encrypted with friendship security credentials. Added more references to the Directed Forwarding feature bit in the Heartbeat message. Reviewed the conditions for searching a Forwarding Table entry corresponding to a PERR without PO and to a PREP with OFN lower than FN in the entry (in case of reply from a member of a group). Fixed the value of the unassigned bearer index. Specify that the Relay feature is to be enabled before forwarding PREP, PTREQ, PTREP, and NINFO reporting sender's neighbor information. |
| | 2018-jul-05 | LC reviewed the Path Metric section. Modified Path Metric state as a composite state. Changed in messages and in tables the meaning of OAR/DAR/NAR fields to report secondary element address ranges. Modified the NINFO format to take into account secondary element address ranges (aligned to PREQ/PREP). Modified the values of OAR and DAR reported in the DF Forwarding Table Status message with Status equal to Success. Clarified that PAREQ message may be segmented and the PAREP timer shall be started right after sending the PAREQ message. Specified that Heartbeat published if DF State changes in the same subnet. Removed the condition checking that PTREQ.PD value is different from any local address before forwarding the PTREQ message. Specified how to set the PO value in PAREP. |
| | 2018-jul-11 | Specified that a DF entry with TM set to 1 is deleted when the corresponding FT entry is deleted as well. Specified that after |

| Revision Number | Date | Comments |
|---|---|---|
| | | Path Request/Reply Delay timer is expired, the node should check again if the conditions to propagate/reply to the PREQ message are still met. Removed additional conditions to be met, beside checking whether the node is a Relay node, before propagating PREP/PTREP messages. Clarified ranges of OAR/DAR in PAREQ. |
| | 2018-jul-12 | Specified that if a PCONF or a PTREQ is received, the BackwardPathValidated flag of the associated Forwarding Table entry is set to 1 (for Path Destination and intermediate nodes). Added a BackwardPathValidated (BPV) flag to the option flag list in PAREQ. Specified that if a path entry that corresponds to a PREQ originated or received exists, the entry is either deleted or invalidated in the backward direction depending on the value of the BackwardPathValidated flag. |
| | 2018-jul-17 | Incorporated all revised sample data and flow-charts for Directed Forwarding. Added notes about setting of BackwardPathValidated flag in path entry. Closed open comments. |
| r06 | 2018-jul-24 | Resolved TE comments. Revised normative text in Section 3.6.8 |
| r07 | 2018-aug-13 | Resolved additional TE feedback. |

***Contributors***

| Name | Company |
|---|---|
| Piergiuseppe Di Marco | Ericsson |
| Wei Shen | Ericsson |
| Danilo Blasi | STMicroelectronics |
| L.C. Ko | MediaTek Inc. |
| Robert Hughes | Intel Corporation |
| Yiting Liao | Intel Corporation |
| Szymon Slupik | Silvair |
| Bogdan Alexandru | NXP Semiconductors |
| Jingcheng Zhang | Ericsson |
| Per Elmdahl | Ericsson |

| Name | Company |
|---|---|
| Arvind Kandhalu | Texas Instruments |
| Robin Heydon | Qualcomm Technologies International |
| Yao Wang | Barrot |

# Contents

**1 Introduction ........ 16**
1.1 Conformance ........ 16
1.2 Bluetooth specification release compatibility ........ 16
1.3 Language ........ 16
1.3.1 Language conventions ........ 16
1.3.2 Acronyms and abbreviations ........ 17
1.3.3 Terminology ........ 19
**2 Mesh system architecture ........ 21**
2.1 Layered architecture ........ 21
2.1.1 Model layer ........ 21
2.1.2 Foundation Model layer ........ 22
2.1.3 Access layer ........ 22
2.1.4 Upper transport layer ........ 22
2.1.5 Lower transport layer ........ 22
2.1.6 Network layer ........ 22
2.1.7 Bearer layer ........ 22
2.2 Overview of mesh operation ........ 22
2.2.1 Network and subnets ........ 23
2.2.2 Devices and nodes ........ 24
2.2.3 Adding devices to a mesh network ........ 24
2.2.4 Communications support ........ 25
2.2.5 Low power support ........ 25
2.3 Architectural concepts ........ 25
2.3.1 States ........ 25
2.3.2 Bound states ........ 26
2.3.3 Messages ........ 26
2.3.4 Elements ........ 27
2.3.5 Addresses ........ 28
2.3.6 Models ........ 28
2.3.7 Example device ........ 34
2.3.8 Publish-subscribe and message exchange ........ 35
2.3.9 Security ........ 36
2.3.10 Directed Forwarding ........ 38
2.3.11 Friendship ........ 39
2.3.12 Features ........ 39

2.3.13 Topology ... 41
2.4 Mesh gateway ... 42
2.5 Concurrency limitations and restrictions ... 42
2.6 Topology limitations and restrictions ... 42
**3 Mesh networking ... 43**
3.1 Conventions ... 43
3.1.1 Endianness and field ordering ... 43
3.2 Features ... 45
3.3 Bearers ... 45
3.3.1 Advertising bearer ... 45
3.3.2 GATT bearer ... 46
3.4 Network layer ... 47
3.4.1 Endianness ... 47
3.4.2 Addresses ... 47
3.4.3 Address validity ... 50
3.4.4 Network PDU ... 51
3.4.5 Network interfaces ... 53
3.4.6 Network layer behavior ... 55
3.5 Lower transport layer ... 59
3.5.1 Endianness ... 59
3.5.2 Lower Transport PDU ... 59
3.5.3 Segmentation and reassembly ... 64
3.5.4 Lower transport layer behavior ... 71
3.5.5 Friend Queue ... 71
3.6 Upper transport layer ... 72
3.6.1 Endianness ... 72
3.6.2 Upper Transport Access PDU ... 72
3.6.3 Upper Transport Control PDU ... 73
3.6.4 Upper transport layer behavior ... 74
3.6.5 Transport Control messages ... 75
3.6.6 Friendship ... 106
3.6.7 Heartbeat ... 124
3.6.8 Directed Forwarding ... 127
*3.6.8.1.2 Network interfaces and bearers ... 129*
*3.6.8.1.3 Group destinations ... 129*
3.7 Access layer ... 173

3.7.1 Endianness ... 173
3.7.2 Model identifier ... 173
3.7.3 Access payload ... 174
3.7.4 Access layer behavior ... 175
3.7.5 Unacknowledged and acknowledged messages ... 177
3.7.6 Publish and subscribe ... 178
3.7.7 Example message sequence charts ... 180
3.8 Mesh security ... 181
3.8.1 Endianness ... 182
3.8.2 Security toolbox ... 182
3.8.3 Sequence number ... 185
3.8.4 IV Index ... 186
3.8.5 Nonce ... 186
3.8.6 Keys ... 191
3.8.7 Message security ... 195
3.8.8 Message replay protection ... 200
3.9 Mesh beacons ... 201
3.9.1 Endianness ... 202
3.9.2 Unprovisioned Device beacon ... 202
3.9.3 Secure Network beacon ... 204
3.10 Mesh network management ... 206
3.10.1 Mesh Network Creation procedure ... 206
3.10.2 Temporary guest access ... 207
3.10.3 Device UUID ... 207
3.10.4 Key Refresh procedure ... 207
3.10.5 IV Update procedure ... 212
3.10.6 IV Index Recovery procedure ... 215
3.10.7 Node Removal procedure ... 216
3.11 Message processing flow ... 217
**4 Foundation models ... 222**
4.1 Conventions ... 222
4.1.1 Endianness ... 222
4.1.2 Log field transformation ... 222
4.2 State definitions ... 223
4.2.1 Composition Data ... 223
4.2.2 Model Publication ... 225

4.2.3 Subscription List ... 228
4.2.4 NetKey List ... 228
4.2.5 AppKey List ... 228
4.2.6 Model to AppKey List ... 228
4.2.7 Default TTL ... 228
4.2.8 Relay ... 229
4.2.9 Attention Timer ... 229
4.2.10 Secure Network Beacon ... 230
4.2.11 GATT Proxy ... 230
4.2.12 Node Identity ... 231
4.2.13 Friend ... 231
4.2.14 Key Refresh Phase ... 231
4.2.15 Health Fault ... 232
4.2.16 Health Fast Period Divisor ... 236
4.2.17 Heartbeat Publication ... 236
4.2.18 Heartbeat Subscription ... 238
4.2.19 Network Transmit ... 240
4.2.20 Relay Retransmit ... 240
4.2.21 PollTimeout List ... 241
4.2.22 Directed Forwarding ... 241
4.2.23 Path Metric ... 242
4.2.24 Path Request Transmit ... 244
4.2.25 Neighbor Information ... 244
4.2.26 Forwarding Table ... 245
4.3 Message definitions ... 248
4.3.1 Supplemental parameter requirements ... 248
4.3.2 Configuration messages ... 249
4.3.3 Health messages ... 281
4.3.4 Directed Forwarding messages ... 287
4.3.5 Messages summary ... 297
4.3.6 Summary of status codes ... 304
4.4 Model definitions ... 305
4.4.1 Configuration Server model ... 305
4.4.2 Configuration Client model ... 322
4.4.3 Health Server model ... 333
4.4.4 Health Client model ... 336

4.4.5 Directed Forwarding Server model ........ 339
4.4.6 Directed Forwarding Client model ........ 342
4.4.7 Summary of SIG Model IDs........ 345
**5 Provisioning........ 346**
5.1 Endianness........ 347
5.2 Provisioning bearer layer ........ 347
5.2.1 PB-ADV........ 347
5.2.2 PB-GATT........ 349
5.3 Generic Provisioning layer ........ 349
5.3.1 Generic Provisioning PDU types ........ 350
5.3.2 Link Establishment procedure........ 354
5.3.3 Generic Provisioning behavior ........ 356
5.4 Provisioning protocol ........ 357
5.4.1 Provisioning PDUs........ 357
5.4.2 Provisioning behavior........ 365
5.4.3 Provisioning security........ 378
5.4.4 Provisioning errors........ 381
**6 Proxy protocol........ 382**
6.1 Endianness........ 382
6.2 Proxy roles ........ 382
6.3 Proxy PDU........ 382
6.3.1 PDU format........ 382
6.3.2 Segmentation ........ 384
6.4 Proxy filtering........ 385
6.4.1 Filter types........ 385
6.5 Proxy configuration messages ........ 385
6.5.1 Set Filter Type ........ 386
6.5.2 Add Addresses to Filter........ 387
6.5.3 Remove Addresses from Filter........ 387
6.5.4 Filter Status ........ 387
6.6 Proxy Server behavior ........ 388
6.7 Proxy Client behavior........ 389
6.8 MSC examples ........ 390
6.8.1 White list filtering........ 390
6.8.2 Black list filtering........ 391
**7 Mesh GATT services ........ 392**

7.1 Mesh Provisioning Service ... 392
7.1.1 Introduction ... 392
7.1.2 Service requirements ... 393
7.1.3 Mesh Provisioning Service characteristics ... 395
7.2 Mesh Proxy Service ... 396
7.2.1 Introduction ... 396
7.2.2 Service requirements ... 397
7.2.3 Mesh Proxy Service characteristics ... 401
**8 Sample data ... 403**
8.1 Security sample data ... 403
8.1.1 s1 SALT generation function ... 403
8.1.2 k1 function ... 403
8.1.3 k2 function (master) ... 403
8.1.4 k2 function (friendship) ... 404
8.1.5 k3 function ... 404
8.1.6 k4 function ... 404
8.2 Mesh key derivation sample data ... 405
8.2.1 Application key AID ... 405
8.2.2 Encryption and privacy keys (Master) ... 405
8.2.3 Encryption and privacy keys (Friendship) ... 405
8.2.4 Network ID ... 406
8.2.5 IdentityKey ... 406
8.2.6 BeaconKey ... 406
8.3 Mesh message sample data ... 406
8.3.1 Message #1 ... 407
8.3.2 Message #2 ... 408
8.3.3 Message #3 ... 410
8.3.4 Message #4 ... 411
8.3.5 Message #5 ... 413
8.3.6 Message #6 ... 414
8.3.7 Message #7 ... 417
8.3.8 Message #8 ... 418
8.3.9 Message #9 ... 419
8.3.10 Message #10 ... 420
8.3.11 Message #11 ... 422
8.3.12 Message #12 ... 422

8.3.13 Message #13 ........ 424
8.3.14 Message #14 ........ 424
8.3.15 Message #15 ........ 426
8.3.16 Message #16 ........ 427
8.3.17 Message #17 ........ 428
8.3.18 Message #18 ........ 429
8.3.19 Message #19 ........ 430
8.3.20 Message #20 ........ 432
8.3.21 Message #21 ........ 434
8.3.22 Message #22 ........ 435
8.3.23 Message #23 ........ 437
8.3.24 Message #24 ........ 438
8.4 Beacon sample data ........ 441
8.4.1 Unprovisioned device beacon (without URI) ........ 441
8.4.2 Unprovisioned device beacon (with URI) ........ 441
8.4.3 Secure Network beacon ........ 442
8.4.4 Secure Network beacon (IV update in progress) ........ 442
8.4.5 Secure Network beacon (IV update complete) ........ 442
8.5 Provisioning Service sample data ........ 443
8.5.1 Mesh Provisioning Service advertising service data ........ 443
8.6 Mesh Proxy Service sample data ........ 443
8.6.1 Service data using Network ID ........ 443
8.6.2 Service data using Node Identity ........ 444
8.7 PB-ADV provisioning sample data ........ 444
8.7.1 PB-ADV Link Open ........ 444
8.7.2 PB-ADV Link Ack ........ 445
8.7.3 PB-ADV Provisioning Invite ........ 445
8.7.4 PB-ADV Provisioning Capabilities ........ 446
8.7.5 PB-ADV Provisioning Start ........ 446
8.7.6 PB-ADV Provisioning Public Key (Provisioner) ........ 447
8.7.7 PB-ADV Provisioning Public Key (Device) ........ 448
8.7.8 PB-ADV Provisioning Confirmation (Provisioner) ........ 449
8.7.9 PB-ADV Provisioning Confirmation (Device) ........ 450
8.7.10 PB-ADV Provisioning Random (Provisioner) ........ 451
8.7.11 PB-ADV Provisioning Random (Device) ........ 452
8.7.12 PB-ADV Provisioning Data ........ 452

8.7.13 PB-ADV Provisioning Complete ..... 453
8.7.14 PB-ADV Link Close ..... 454
8.8 PB-GATT SAR sample data ..... 454
8.8.1 1st segment ..... 454
8.8.2 2nd segment ..... 455
8.8.3 3rd segment ..... 455
8.8.4 4th segment ..... 455
8.9 Proxy Configuration Message sample data ..... 455
8.10 Composition Data sample data ..... 456
8.11 Directed Forwarding sample data ..... 456
8.11.1 PREQ with all unset options ..... 456
8.11.2 PREQ with all set options ..... 458
8.11.3 PREP with all unset options ..... 459
8.11.4 PREP with all set options ..... 461
8.11.5 PCONF ..... 463
8.11.6 PTREQ with all unset options ..... 464
8.11.7 PTREQ with all set options ..... 466
8.11.8 PTREP ..... 467
8.11.9 PAREQ with all unset options ..... 469
8.11.10 PAREQ with all set options ..... 470
8.11.11 PAREP ..... 474
8.11.12 PERR with all unset options ..... 476
8.11.13 PERR with all set options ..... 477
8.11.14 NINFO with all unset options and no neighbor information ..... 479
8.11.15 NINFO with all set options and with neighbor information ..... 480
8.11.16 LREQ ..... 483
8.11.17 LREP ..... 485
8.11.18 DF Directed Forwarding Get ..... 486
8.11.19 DF Directed Forwarding Set ..... 488
8.11.20 DF Directed Forwarding Status ..... 489
8.11.21 DF Path Metric Get ..... 491
8.11.22 DF Path Metric Set ..... 493
8.11.23 DF Path Metric Status ..... 494
8.11.24 DF Neighbor Information Get ..... 496
8.11.25 DF Neighbor Information Set ..... 497
8.11.26 DF Neighbor Information Status ..... 499

8.11.27 DF Forwarding Table Add ........ 501
8.11.28 DF Forwarding Table Delete ........ 502
8.11.29 DF Forwarding Table Status ........ 504
8.11.30 DF Forwarding Table Get ........ 507
8.11.31 DF Forwarding Table List ........ 509
8.11.32 DF Path Request Transmit Get ........ 512
8.11.33 DF Path Request Transmit Set ........ 513
8.11.34 DF Path Request Transmit Status ........ 515
**9 References ........ 517**

# 1 Introduction

The Bluetooth Mesh Profile specification defines fundamental requirements to enable an interoperable mesh networking solution for Bluetooth low energy wireless technology.

## 1.1 Conformance

If conformance to this specification is claimed, all capabilities indicated as mandatory for this specification shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated.

## 1.2 Bluetooth specification release compatibility

This specification shall be used with:

- Core Specification Addendum 6 [3] combined with an allowed Bluetooth Core Specification (see [3] Volume 1, Part D, Section 1.2, Table 1.3), OR
- Any version of the Bluetooth Core Specification later than v5.0.

The Generic Attribute Profile (GATT) is required if the GATT provisioning bearer defined in Section 5.2.2 is supported or if the GATT bearer defined in Section 3.3.2 is supported.

## 1.3 Language

### 1.3.1 Language conventions

The Bluetooth SIG has established the following conventions for use of the words ***shall***, ***must***, ***will***, ***should***, ***may***, ***can***, ***is***, and ***note*** in the development of specifications:

| | |
|---|---|
| Shall | is required to – used to define requirements |
| Must | is a natural consequence of – used only to describe unavoidable situations |
| Will | it is true that – only used in statements of fact |
| Should | is recommended that – used to indicate that among several possibilities one is recommended as particularly suitable, but not required |
| May | is permitted to – used to allow options |
| Can | is able to – used to relate statements in a causal manner |
| is | is defined as – used to further explain elements that are previously required or allowed |
| note | Used to indicate text that is included for informational purposes only and is not required in order to implement the specification. Informative text in a note continues to the end of the paragraph. |

For clarity of the definition of those terms, see Core Specification Volume 1, Part E, Section 1.

#### 1.3.1.1 Reserved for Future Use

Where a field in a packet, Protocol Data Unit (PDU), or other data structure is described as "Reserved for Future Use" (irrespective of whether in uppercase or lowercase), the device creating the structure shall set its value to zero unless otherwise specified. Any device receiving or interpreting the structure shall ignore that field; in particular, it shall not reject the structure because of the value of the field.

Where a field, parameter, or other variable object can take a range of values and some values are described as "Reserved for Future Use," a device sending the object shall not set the object to those values. A device receiving an object with such a value should reject it, and any data structure containing it, as being erroneous; however, this does not apply in a context where the object is described as being ignored or it is specified to ignore unrecognized values.

When a field value is a bit field, unassigned bits can be marked as Reserved for Future Use and shall be set to 0. Implementations that receive a message that contains a Reserved for Future Use bit that is set to 1 shall process the message as if that bit was set to 0, except where specified otherwise.

The acronym RFU is equivalent to "Reserved for Future Use."

#### 1.3.1.2 Prohibited

When a field value is an enumeration, unassigned values can be marked as “Prohibited.” These values shall never be used by an implementation, and any message received that includes a Prohibited value shall be ignored and shall not be processed and shall not be responded to.

Where a field, parameter, or other variable object can take a range of values, and some values are described as “Prohibited,” devices shall not set the object to any of those Prohibited values. A device receiving an object with such a value should reject it, and any data structure containing it, as being erroneous.

“Prohibited” is never abbreviated.

### 1.3.2 Acronyms and abbreviations

| Abbreviation or Acronym | Meaning |
|---|---|
| ACK | Acknowledgment |
| AD | Advertising Data |
| AES | Advanced Encryption Standard |
| AID | Application Key Identifier |
| AKF | Application Key Flag |
| ASCII | American Standard Code for Information Interchange |
| ATT | Attribute Protocol |

| Abbreviation or Acronym | Meaning |
|---|---|
| ATT_MTU | Attribute Protocol Maximum Transmission Unit |
| BR/EDR | Basic Rate / Enhanced Data Rate |
| CCM | Counter with CBC-MAC |
| CID | Company Identifier |
| CMAC | Cipher-based Message Authentication Code |
| CTL | Network control message indication |
| DST | Destination |
| ECB | Electronic CodeBook |
| ECDH | Elliptic Curve Diffie-Hellman |
| FCS | Frame Check Sequence |
| FIPS | Federal Information Processing Standards |
| FSN | Friend Sequence Number |
| GAP | Generic Access Profile |
| GATT | Generic Attribute Profile |
| ID | Identifier |
| IEEE | Institute of Electrical and Electronics Engineers |
| IKM | Input Key Material |
| IVI | Initialization Vector Index |
| JSON | JavaScript Object Notation |
| LED | Light Emitting Diode |
| LREP | Link Quality Measurement Reply message |
| LREQ | Link Quality Measurement Request message |
| LSO | Least Significant Octet |
| MAC | Message Authentication Code |
| MD | More Data |
| MIC | Message Integrity Check |
| MSO | Most Significant Octet |
| MTU | Maximum Transmission Unit |
| NID | Network Identifier |
| NINFO | Neightbor Information message |
| OBO | On Behalf Of (another element) |
| OKM | Output Key Material |

| Abbreviation or Acronym | Meaning |
|---|---|
| OOB | Out of Band |
| PDU | Protocol Data Unit |
| PID | Product Identifier |
| PAREQ | Path Assistant Request message |
| PAREP | Path Assistant Reply message |
| PCONF | Path Confirmation message |
| PERR | Path Error message |
| PREP | Path Reply message |
| PREQ | Path Request message |
| PTREQ | Path Tracing Request message |
| PTREP | Path Tracking Reply message |
| RFU | Reserved for Future Use |
| RPL | Replay Protection List |
| RSSI | Received Signal Strength Indicator |
| SAR | Segmentation And Reassembly |
| SEG | Segmentation indication bit |
| SEQ | Sequence Number |
| SIG | Special Interest Group |
| SRC | Source |
| SZMIC | Size of Message Integrity Check |
| TTL | Time To Live |
| URI | Uniform Resource Indicator |
| UUID | Universally Unique Identifier |
| VID | Version Identifier |
| WG | Working Group |

*Table 1.1: Abbreviations and acronyms*

### 1.3.3 Terminology

| Term | Definition |
|---|---|
| Address | The identity of one or more elements in one or more nodes. |
| Configuration Client | A node that implements the Configuration Client model. |
| Destination | The address to which a message is sent. |

| | |
|---|---|
| Device | An entity that is capable of being provisioned onto a mesh network. |
| Element | An addressable entity within a device. A device is required to have at least one element. |
| Message | A sequence of octets that is sent from a source to a destination. |
| Neighbors | Nodes in direct radio range (single hop). |
| Network | A group of nodes sharing a common address space. |
| Node | A provisioned device. |
| Provision | The process of authenticating and providing basic information (including unicast addresses and a network key) to a device. A device must be provisioned to become a node. Once provisioned, a node can transmit or receive messages in a mesh network. |
| Provisioner | A node that is capable of adding a device to a mesh network. |
| Relay | A node that receives and then retransmits messages. |
| Source | The address from which a message is sent. |
| State | A value representing a condition of an element that is exposed by an element of a node. |
| Subnet | A group of nodes that can communicate with each other. |

*Table 1.2: Terminology*

# 2 Mesh system architecture

This section provides an overview of the mesh network operation and layered system architecture.

## 2.1 Layered architecture

The Mesh Profile specification is defined as a layered architecture as shown in Figure 2.1.

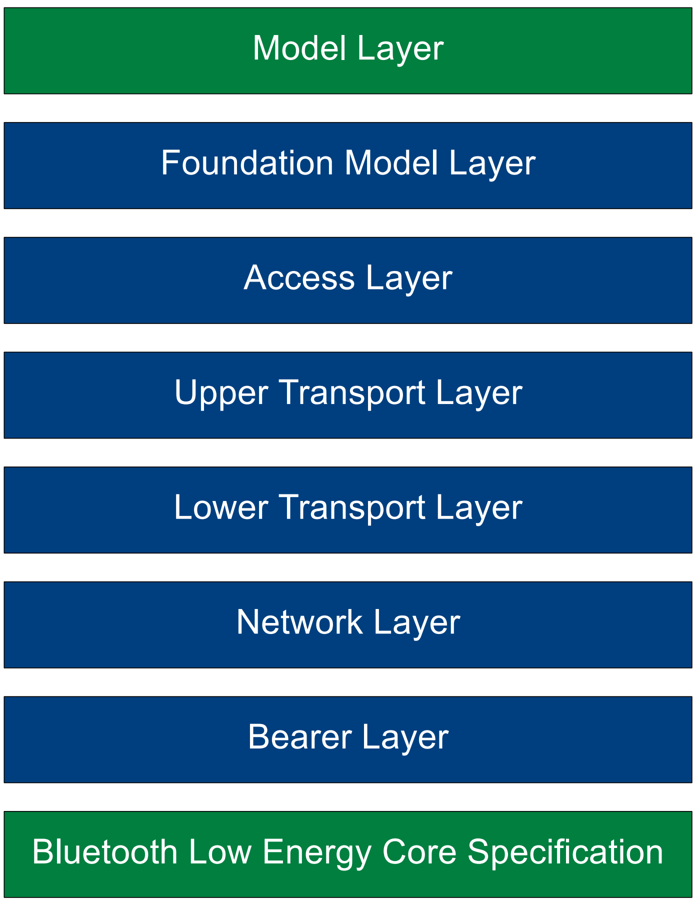


*Figure 2.1: Mesh system architecture*

### 2.1.1 Model layer

The Model layer defines models that are used to standardize the operation of typical user scenarios and are defined in the Bluetooth Mesh Model specification [11] or other higher layer specifications. Examples of higher layer model specifications include models for lighting and sensors.

### 2.1.2 Foundation Model layer

The Foundation Model layer defines the states, messages, and models required to configure and manage a mesh network.

### 2.1.3 Access layer

The access layer defines how higher layer applications can use the upper transport layer. It defines the format of the application data; it defines and controls the application data encryption and decryption performed in the upper transport layer; and it checks whether the incoming application data has been received in the context of the right network and application keys before forwarding it to the higher layer.

### 2.1.4 Upper transport layer

The upper transport layer encrypts, decrypts, and authenticates application data and is designed to provide confidentiality of access messages. It also defines how transport control messages are used to manage the upper transport layer between nodes, including when used by the Friend feature and by the Directed Forwarding feature.

### 2.1.5 Lower transport layer

The lower transport layer defines how upper transport layer messages are segmented and reassembled into multiple Lower Transport PDUs to deliver large upper transport layer messages to other nodes. It also defines a single control message to manage segmentation and reassembly.

### 2.1.6 Network layer

The network layer defines how transport messages are addressed towards one or more elements. It defines the network message format that allows Transport PDUs to be transported by the bearer layer. The network layer decides whether to relay/forward messages, accept them for further processing, or reject them. It also defines how a network message is encrypted and authenticated.

### 2.1.7 Bearer layer

The bearer layer defines how network messages are transported between nodes. There are two bearers defined, the advertising bearer and the GATT bearer. Additional bearers may be defined in the future.

## 2.2 Overview of mesh operation

The mesh network operation defined by this specification is designed to:

- enable messages to be sent from one element to one or more elements;
- allow messages to be relayed via other nodes to extend the range of communication;

- secure messages against known security attacks, including eavesdropping attacks, man-in-the-middle attacks, replay attacks, trash-can attacks, brute-force key attacks, and possible additional security attacks not documented here;
- work on existing devices in the market today;
- deliver messages in a timely manner;
- continue to work when one or more devices are moved or stop operating; and
- have built-in forward compatibility to support future versions of the Mesh Profile specification.

This specification defines a managed-flood-based mesh network, which uses broadcast channels to transmit messages so that other nodes can receive messages and relay these messages; thus extending the range of the original message. Any device in a managed-flood mesh network can send a message at any time as long as there is a sufficient density of devices that are listening and relaying messages.

There are a number of methods used by this specification to restrict the unlimited relaying of messages in a managed-flood mesh network. The two main methods used are the network message cache method and the time to live method.

The network message cache is designed to prevent devices from relaying previously received messages by adding all messages to a cached list. When a message is received, it is checked against the list and ignored if already present. If not already received, then it is added to the cache so that it can be ignored in the future. To prevent this list from becoming too long, the number of messages that are cached is limited by implementation.

Each message includes a Time to Live (TTL) value that limits the number of times a message can be relayed. Each time a message is received and then relayed (up to a maximum of 126 times) by a device, the TTL value is decremented by 1.

In addition, nodes use algorithms for discovering and maintaining paths to one or more destinations so that message forwarding is enabled only along a subset of nodes in the network.

### 2.2.1 Network and subnets

A mesh network consists of nodes sharing four common resources:

- network addresses used to identify source and destination of messages (see Section 3.4.2);
- network keys used to secure and authenticate messages at the network layer (see Section 3.8.6.3);
- application keys used to secure and authenticate messages at the access layer (see Section 3.8.6.2); and
- an IV Index used to extend the lifetime of the network (see Section 3.8.4).

A network can have one or more subnets that facilitate ”area” isolation (e.g., isolated hotel room subnets within a hotel network). A subnet is a group of nodes that can communicate with each other at a network layer because they share a network key. A node may belong to one or more subnets by knowing one or more network keys. At the time of provisioning, a device is provisioned to one subnet and may be added to more subnets using the Configuration Model.

There is one special subnet called the primary subnet, which is based on the primary NetKey (see Section 3.8.6.4). Nodes on the primary subnet participate in the IV Update procedure (see Section 3.10.5), and propagate IV updates to other subnets, while nodes on other subnets only propagate the IV Index updates to those subnets.

The network resources are managed by a node that implements the Configuration Client model, known as the Configuration Client, (typically a smart phone or other mobile computing device) and are allocated to nodes at the time of configuration (see Section 5) using the Configuration Server model (see Section 4.4.1). In particular, a Provisioner manages allocation of addresses to make sure no duplicate unicast addresses are allocated, whereas a Configuration Client generates and distributes network and application keys and makes sure that devices that need to communicate with each other share proper keys for both network and access layers. The Configuration Client also knows device keys (see Section 3.8.6.1), which are used to secure communication with each individual node, including distributing updated network and application keys.

### 2.2.2 Devices and nodes

A device that is not a member of a mesh network is known as an unprovisioned device. A device that is a member of a mesh network is known as a node. A Provisioner is used to manage the transitions between an unprovisioned device and a node.

An unprovisioned device cannot send or receive mesh messages; however, it advertises its presence to Provisioners. A Provisioner will invite an unprovisioned device into a mesh network after it has been authenticated, converting the unprovisioned device into a node.

A node can send or receive mesh messages and is managed by a Configuration Client, that may also be the same device as the Provisioner, over the mesh network to configure how the node communicates with other nodes. A Configuration Client can remove a node from a mesh network, which reverts it back to an unprovisioned device.

A device may support multiple instances of a node by offering itself to be provisioned to another mesh network after already being provisioned to a mesh network. Each instance of a mesh network is determined by addresses and a device key obtained by the device during provisioning.

### 2.2.3 Adding devices to a mesh network

Devices are added to a mesh network by a Provisioner, at which point they become nodes. The provisioning of devices into a mesh network differs from the point-to-point bonding and pairing that is typically used in Bluetooth wireless technology. Provisioning of devices is enabled using either a simple advertising bearer or a point-to-point GATT-based bearer. A single provisioning protocol is used over both bearers. Provisioning over an advertising-based bearer is implemented by all devices. Provisioning over a GATT-based bearer allows devices such as legacy phones (i.e., devices that do not support provisioning over an advertising bearer natively) to be Provisioners.

To assist with provisioning of multiple devices, a device has an attention timer that can be set by a Provisioner. When set to a non-zero value, the device identifies itself using any means it can. For example, the device may flash a light, make a sound, or vibrate. When the attention timer expires, the device stops identifying itself. This allows a Provisioner to send a single message to a device to cause it to identify itself and the device automatically stops identifying itself after a given time.

The protocol to run over these two bearers is a derivative of the Security Manager protocol of v4.2 of the Bluetooth Core Specification to introduce the ability to authenticate devices that have a very limited user interface, such as a light or a switch. The Security Manager protocol requires a reliable bearer, something that cannot be guaranteed by the advertising provisioning bearer; therefore the protocol used in this specification is designed to enable reliable delivery of messages independent of the bearer. The similarity to the Security Manager protocol enables significant reuse of existing code on devices that have implemented such functionality.

### 2.2.4 Communications support

Many current devices are unable to advertise or comprehend mesh messages without being updated. To enable these devices to communicate with a node in a mesh network without the need for an operating system update or similar hardware/software update, the specification enables the use of GATT connectivity for all existing devices.

### 2.2.5 Low power support

The features within this specification enable many devices in the mesh network to be battery-powered or to use techniques such as energy harvesting. Such devices may be constrained in how they can function as a part of a mesh network (e.g., devices that only send data when interacted with). This specification does not require devices to coordinate transmissions, make connections, or restart security on every connection; thus facilitating low power operation. Devices needing low power support can associate themselves with an always-on device that stores and relays messages on their behalf, using the concept known as Friendship (see Section 3.6.6). However, devices that relay messages will receive messages as well as forward messages a majority of the time and are likely to use significantly more power than could be provided by typical small batteries or capacitors.

## 2.3 Architectural concepts

The mesh networking architecture uses several different concepts: states, messages, bindings, elements, addressing, models, publish-subscribe, mesh keys, and associations.

### 2.3.1 States

A state is a value representing a condition of an element.

An element exposing a state is referred to as a server. For example, the simplest server is a Generic OnOff Server, representing that it is either on or off.

An element accessing a state is referred to as a client. For example, the simplest client is a Generic OnOff Client (a binary switch) that is able to control a Generic OnOff Server via messages defined by the Generic OnOff Model.

States that are composed of two of more values are known as composite states. For example, a color-changing lamp can control color hue separately from color saturation and brightness.

### 2.3.2 Bound states

When a state is bound to another state, a change in one results in a change in the other. Bound states may be from different models in one or more elements. For example, a common type of binding is between a Level state and an OnOff state: changing the Level to 0 changes the bound OnOff state to Off and changing the Level to a non-zero value changes the bound OnOff state to On.

### 2.3.3 Messages

All communication within a mesh network is accomplished by sending messages. Messages operate on states. For each state, there is a defined set of messages that a server supports and a client may use to request a value of a state or to change a state. A server may also transmit unsolicited messages carrying information about states and/or changing states.

A message is defined as having an opcode, associated parameters, and behavior. An opcode may be a single octet (for special messages that require maximum possible payload for parameters), 2 octets (for standard messages), or 3 octets (for vendor-specific messages).

A total message size, including an opcode, is determined by the underlying transport layer, which may use a Segmentation and Reassembly (SAR) mechanism. To maximize performance and avoid the overhead of SAR, a design goal is to fit messages in a single segment. The transport layer provides up to 11 octets for a non-segmented message, leaving up to 10 octets that are available for parameters when using a 1-octet opcode, up to 9 octets available for parameters when using a 2-octet opcode, and up to 8 octets available for parameters when using a vendor-specific 3-octet opcode.

The transport layer provides a mechanism of SAR capable of transporting up to 32 segments. The maximum message size when using the SAR is 384 octets. This means (excluding an Application MIC) up to 379 octets are available for parameters when using a 1-octet opcode, up to 378 octets are available for parameters when using a 2-octet opcode, and up to 377 octets are available for parameters when using a vendor-specific 3-octet opcode.

SAR effectively does not impose any extra overhead on the access layer payload per segment: a 10-octet message is transported as an unsegmented message, and a 20-octet message is transported as a segmented message that uses two segments.

Message definitions contain tables of parameters. In a message payload, parameters follow an opcode, and parameter offsets are in octets unless otherwise specified.

Messages are defined as acknowledged or unacknowledged. An acknowledged message (e.g., a Get message) requires a response whereas an unacknowledged message (e.g. a status message) does not require a response.

Set, Clear, Recall, and Store messages are defined in two variants: acknowledged and unacknowledged. The message variants are semantically identical but use separate opcodes.

### 2.3.4 Elements

An element is an addressable entity within a node. Each node has at least one element, the primary element, and may have one or more additional secondary elements. The number and structure of elements is static and does not change throughout the lifetime of a node (that is, as long as the node is part of a network).

The primary element is addressed using the first unicast address assigned to the node during provisioning. Each additional secondary element is addressed using the subsequent addresses. These unicast element addresses allow nodes to identify which element within a node is transmitting or receiving a message.

If the number and structure of elements changes, for example due to a firmware update, the node must be reprovisioned. The Node Removal procedure (see Section 3.10.7) is used when a firmware update is performed that changes the number or structure of elements.

Messages are dispatched within models based on opcodes and element addresses.

An element is not allowed to contain multiple instances of models that use the same message (for example, an “On” message). When multiple models within the same element use the same message, the models are said to “overlap.” To implement multiple instances of overlapping models within a single node (for example, to control multiple light fixtures that can be turned on and off), the node is required to contain multiple elements.

For example, a light fixture may have two lamps, each implementing an instance of the Light Lightness Server model and an instance of the Generic Power OnOff Server model. This requires that the node contain two elements, one for each lamp. When it receives an "On" message, the node uses the unicast address of the element to identify which instance of the Generic Power OnOff Server model the message is addressed to.

In another example, a dual-socket power strip contains two independent energy measurement sensors that can measure power consumed by an appliance connected to a socket. This would require that the node have two Sensor Data states, each in a separate element. The first element, the primary element, would be identified using the unicast address for the node and would include a state for the first energy sensor as well as states representing the configuration of the node. The second element, a secondary element, would be identified using a unicast element address and would include the state for the second energy sensor.

Each element has a GATT Bluetooth Namespace Descriptor [5] value that helps identify which part of the node this element represents. These namespace descriptor values use the same definitions as GATT. For example, the elements of the temperature sensor would use the values “inside” and “outside.”

### 2.3.5 Addresses

An address may be a unicast address, a virtual address, or a group address. There is also a special value to represent an unassigned address that is not used in messages.

A unicast address is allocated to an element and always represents a single element of a node. There are 32767 unicast addresses per mesh network.

A virtual address is a multicast address and can represent multiple elements on one or more nodes. Each virtual address logically represents a Label UUID, which is a 128-bit value that does not have to be managed centrally. Each message sent to a Label UUID includes the full Label UUID in the message integrity check value that is used to authenticate the message. To reduce the overhead of checking every known Label UUID, a hash of the Label UUID is used. There are 16384 hash values, each of which codifies a set of virtual addresses. While there are only 16384 hash values used in a virtual address, each hash value can represent millions of possible Label UUIDs; therefore, the number of virtual addresses is considered very large.

A group address is a multicast address and can represent multiple elements on one or more nodes. There are 16384 group addresses per mesh network. There are a set of fixed group addresses that are used to address a subset of all primary elements of nodes based on the functionality of those nodes. All other group addresses are known as dynamically assigned group addresses. There are 256 fixed group addresses and 16128 dynamically assigned group addresses.

### 2.3.6 Models

A model defines the basic functionality of a node. A node may include multiple models. A model defines the required states (as described in Section 2.3.1), the messages that act upon those states (as described in Section 2.3.3), and any associated behavior.

A mesh application is specified using a client-server architecture communicating with a publish-subscribe paradigm. Due to the nature of mesh networks and the recognition that the configuration of behavior is performed by a Configuration Client, an application is not defined in a single end-to-end specification such as a profile. Instead, an application is defined in a client model, a server model, and a control model.

This specification defines three types of model: server models, client models, and control models:

- **Server model:** A server model is composed of one or more states spanning one or more elements. The server model defines a set of mandatory messages that it can transmit or receive, the behavior required of the element when it transmits and receives such messages, and any additional behavior that occurs after messages are transmitted or received.

- **Client model:** A client model defines a set of messages (both mandatory and optional) that a client uses to request, change, or consume corresponding server states, as defined by a server model. The client model does not have state.

- **Control model:** A control model may contain client model functionality to communicate with other server models and server model functionality to communicate with other client models. A control model may also contain *control logic*, which is a set of rules and behaviors that coordinate the interactions between other models that the control model connects to.

A single device may include server, client, and control models.

For example, Figure 2.2 shows the element-model structure for a device that implements a server model (Device C) with a state and supporting messages R, S, T, X, Y, Z; and two devices that implement a client model, with Device A supporting messages X, Y, and Z and Device B supporting messages R, S, T, and Z.

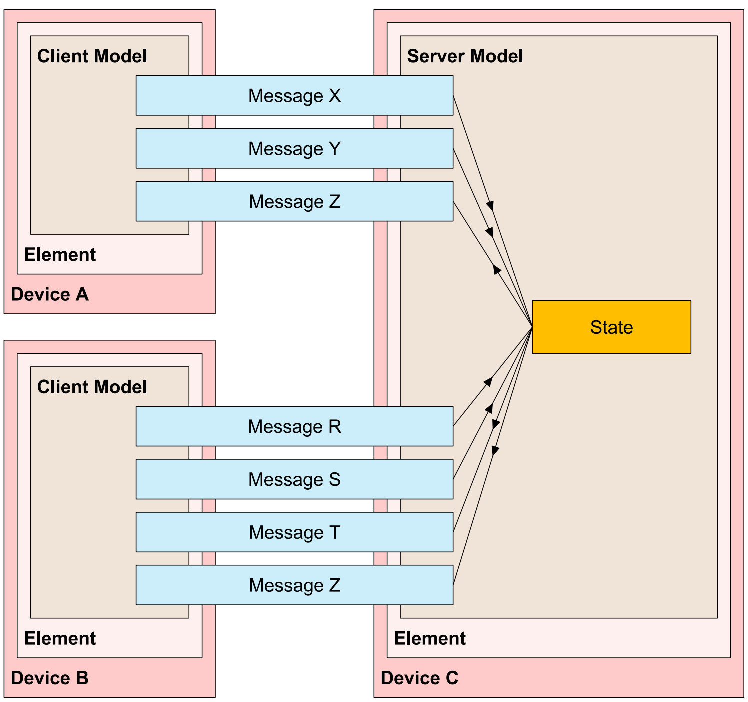


*Figure 2.2: Client-server model communication*

In another example, Figure 2.3 shows the element-model structure of a device that implements a control model. Device C can communicate with server models as a client (supporting messages X, Y, and Z and messages R, S, and T respectively) and can communicate with client models as a server (supporting messages A, B, and C).

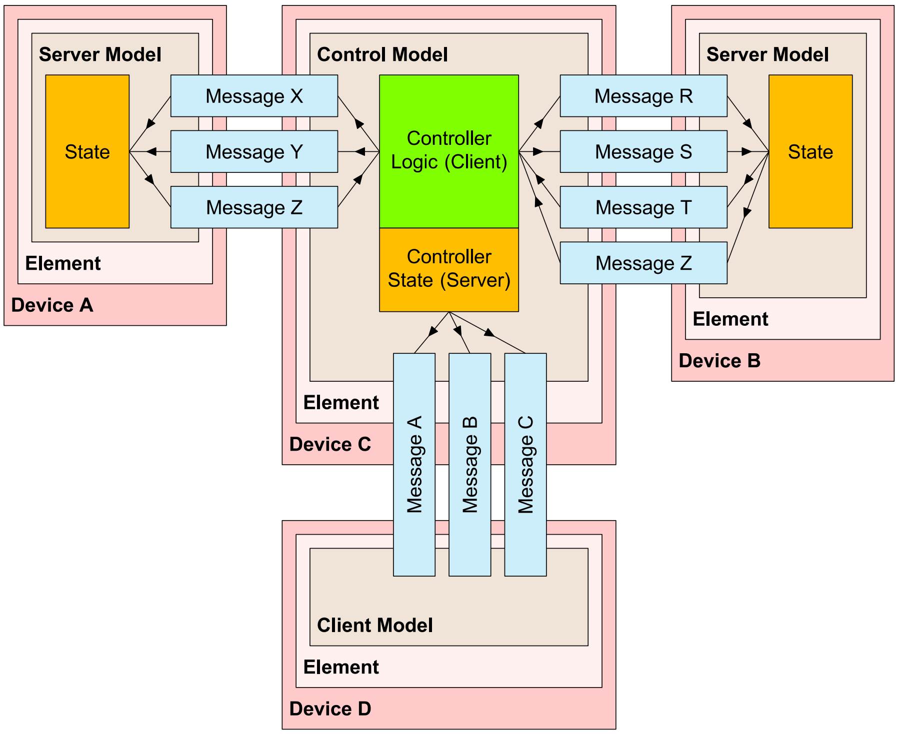


*Figure 2.3: Control model communication*

A lighting controller is an example of an implementation of a control model. The lighting controller needs to function as a client to sensors (to measure occupancy and/or ambient light) and to light sources (such as lamps or other luminaires). The lighting controller also would function as a server to a settings client (such as a smartphone application that configures its parameters). Such a lighting controller may be included within a sensor or light source or it may a separate device.

Models can define functions of a device as a network node, such as key management, address assignment, and relaying of messages. Models also define physical behaviors of a device built around a network node, such as power control, lighting control, and sensor data collection. There may be nodes implementing only network-related functions, such as Relay nodes or Proxy nodes, while the majority of nodes are able to interact with the physical world by means of controlling electrical power, controlling light emissions, or sensing environmental data.

A message can be used by multiple different models. Message behavior is the same in each model, enabling a common understanding among client, server, and control models because the behavior is consistent regardless of the models that send and process the message.

Model specifications are designed to be very small and self-contained. A model can, at the specification definition time, require other models that must also be instantiated within the same node. This is called *extending*, which means a model can extend other models.

Models that do not extend other models are referred to as *root models*.

Model specifications are immutable: it is not possible to remove or add behavior to a model, whether the desired behavior is optional behavior or mandatory. Models are not versioned and have no feature bits. If additional behavior is required in a model, then a new extended model is defined that exposes the required behavior and can be implemented alongside the original model.

Therefore, knowledge of the models supported by an element determines the exact behavior exposed by that element.

Models may be defined and adopted by Bluetooth SIG and may be defined by vendors. Models defined by Bluetooth SIG are known as SIG adopted models, and models defined by vendors are known as vendor models. Models are identified by unique identifiers, which can be either 16 bits, for SIG adopted models, or 32 bits, for vendor models.

For example, Figure 2.4 shows the element-model structure of a device that implements a root model with two bound states and a set of messages operating on each state. The root model is within the primary element and is extended by the extended model that adds another state on a secondary element. Messages are not capable of differentiating among multiple instances of the same state on the same element. Therefore, when more than one instance of a given state is present on a device, each instance is required to be in a separate element. In this example, the second instance of State X is required to be located on the second element because it is the same type of a state and thus has the same types of messages serving it.

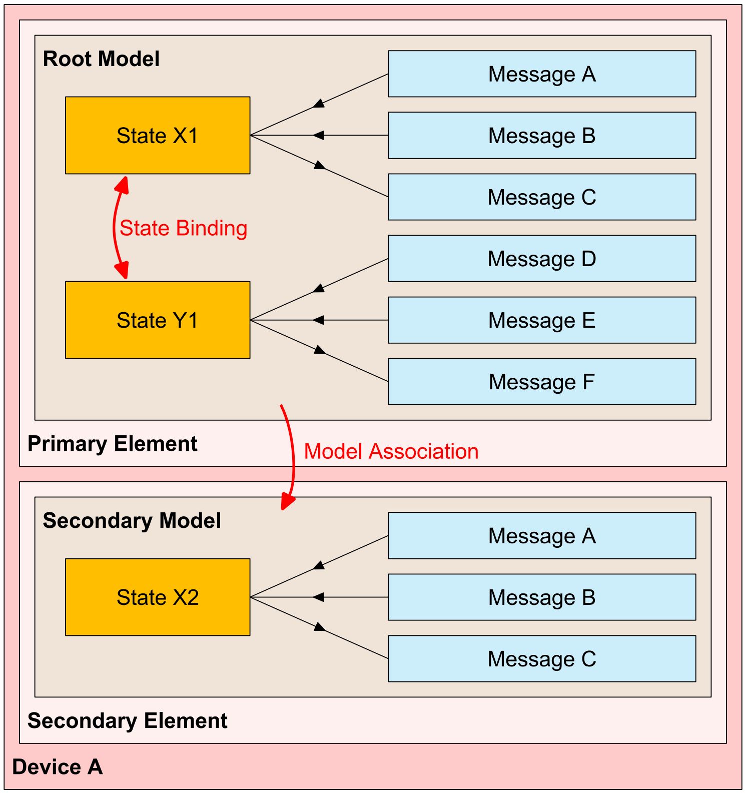


*Figure 2.4: Element-model structure of a device*

This example structure may be multiplied for a composite device. For example, a composite device may have multiple instances of the same root model (or extended models), each on a separate element (or set of elements). Also, if a model (root or extended) needs more than one instance of a particular state, the states must be distributed across several elements so that, at most, a single instance of any given state is on an element.

Figure 2.5 illustrates how the element-model structure of the device in Figure 2.4 might be implemented in a composite device. The element-model structure of the device is described by the Composition Data (see Section 4.2.1) that is read by a Configuration Client after provisioning (see Section 5), using the Configuration Server model and the Configuration Client model (see Sections 4.4.1 and 4.4.2).

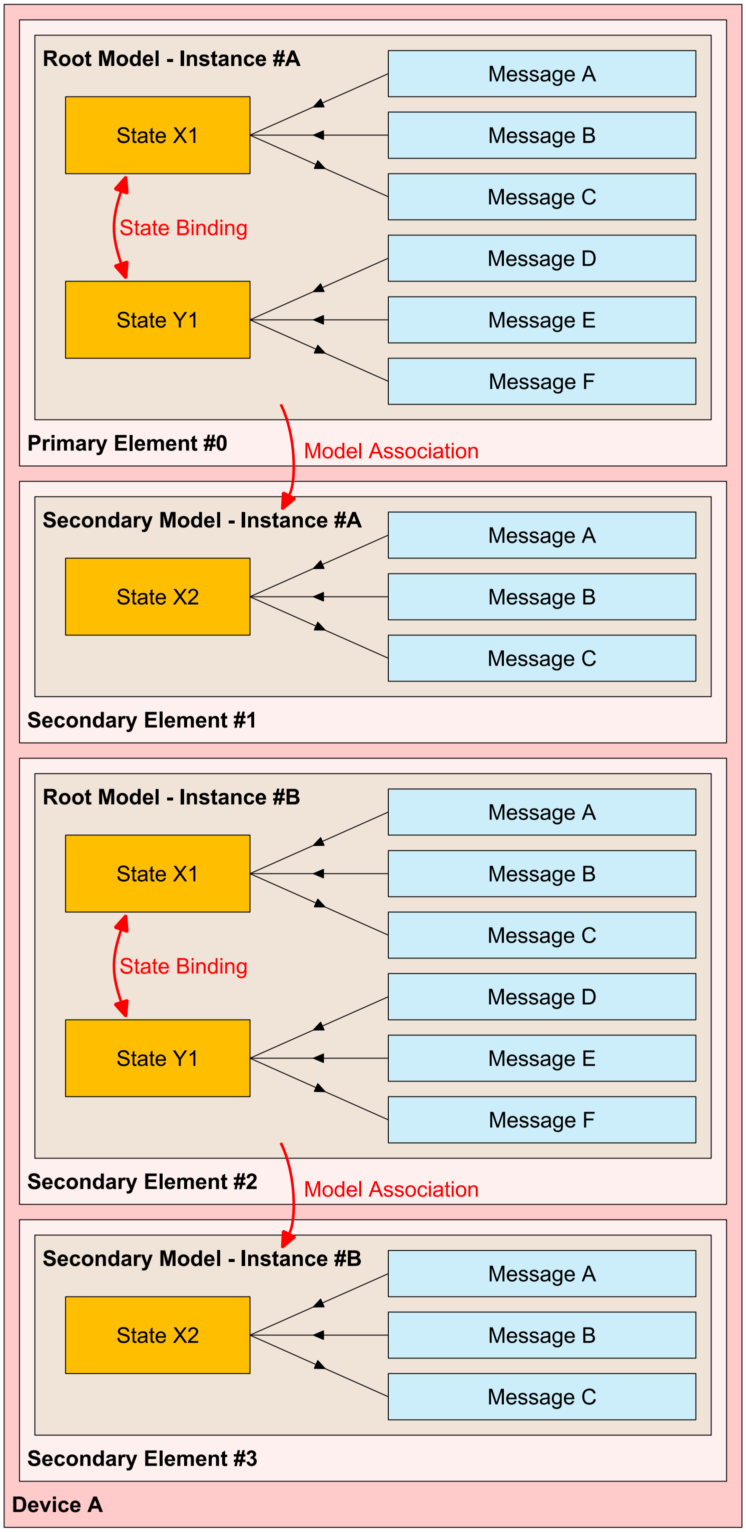
Root Model - Instance #A
State X1
State Binding
State Y1
Message A
Message B
Message C
Message D
Message E
Message F
Primary Element #0
Model Association
Secondary Model - Instance #A
State X2
Message A
Message B
Message C
Secondary Element #1
Root Model - Instance #B
State X1
State Binding
State Y1
Message A
Message B
Message C
Message D
Message E
Message F
Secondary Element #2
Model Association
Secondary Model - Instance #B
State X2
Message A
Message B
Message C
Secondary Element #3
Device A

*Figure 2.5: Element-model structure of a composite device*

### 2.3.7 Example device

To help explain how the arrangement of models within elements determines the state and behavior of a device, we will use a dual-socket smart power strip device (shown in Figure 2.6) as an example. This device has a single radio that has the low energy feature of Bluetooth and two independent power sockets, each capable of controlling the power output. This example includes states, messages, and models defined in the Mesh Model specification [11].

*Figure 2.6: Dual-socket smart power strip*

The device has two elements (see Section 2.3.4) that represent each of the two power sockets. Each element has a unicast address assigned to it.

The functionality of each element is defined by the Generic Power Level Server model. The model defines a set of states on a server, as well as a set of messages that operate on the states.

A Generic Power Level Set message may be sent to the device to control the output power. The message is addressed to an element and carries the element's address in the Destination Address (DST) field (see Section 2.3.8).

The sockets can also be controlled by generic devices (such as a dimmer) that implement the Generic Level Client model (and do not know anything about power control). This model simply sets a desired level to zero, a maximum value, or a value in between. Power to the sockets is controlled through state binding. In each power socket, the Generic Power Actual state is bound to the Generic Level state. A Generic Level Client sends Generic Level messages to the Generic Level Server. The Generic Level state is changed, which in turn (via the defined binding) changes the Generic Power Actual state that controls the power output.

Elements can report states. In our example, each socket may report power level as well as the energy consumption of a device plugged into the socket. Energy consumption is reported using messages defined by the Sensor Server model. Each message has the element's address, which identifies the socket, in its SRC field (see Section 3.4.4.6).

Figure 2.7 illustrates the element-model structure for the dual-socket device. Functionally, both elements of the device have identical features. The only difference is that the primary element handles the Configuration Server model, which is used for network management, in addition to the other models.

Each element may have other models defined such as the Health Server model (see Section 4.4.4) or models defined in the Mesh Model specification [11].

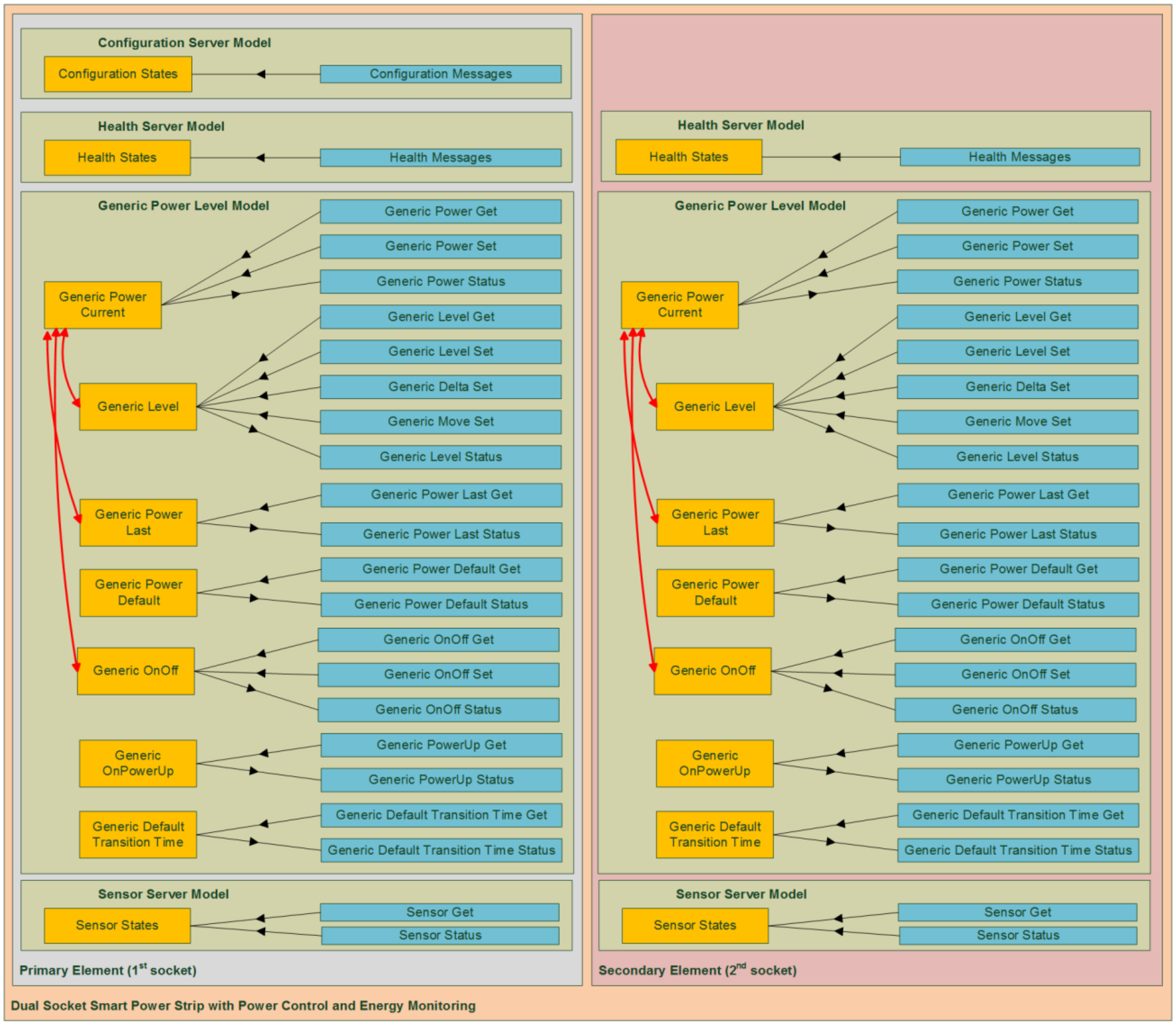


*Figure 2.7: Element-model structure for the example device*

## 2.3.8 Publish-subscribe and message exchange

Publication and subscription of data within the mesh network is described as using a publish-subscribe paradigm. Nodes that generate messages publish the messages to a unicast address, group address, or virtual address. Nodes that are interested in receiving the messages will subscribe to these addresses.

Generated messages are sent to destination mesh addresses that can be unicast, pre-configured group addresses, or virtual addresses. Messages can be sent as replies to other messages or can be unsolicited messages. When an instance of a model is sending a reply message, it uses the incoming message originator's source address as the destination address. When an instance of a model is sending unsolicited messages, it uses a model publish address as the destination address. Each instance of a model within a node has a single publish address.

On the receiving side, each instance of a model within a node can subscribe to one or more group addresses or virtual addresses. Whenever a message that is addressed to a group address or a virtual address on one of the model's subscription lists arrives, it is processed by the node. A message is also processed when its destination address is the unicast address of a receiving element or when its destination address is a fixed group address that this device is a member of. If a node has multiple elements, then the message is processed once on each of the addressed elements.

Publish addresses and subscription lists for models defined by higher layer specifications use the Model Publication and Subscription List states that are managed by the Configuration Server Model.

A node can have any number of subscriptions per instance of a model's element, although nodes may limit the number of subscriptions that are supported. Using multiple subscription addresses allows a node to respond to messages that are published to different groups. For example, a light may be subscribed to messages sent to the bedside light group, the bedroom group, the upstairs group, and the house group.

Each message is sent from a single unicast address (an element address) and sequenced using a unique sequence number to facilitate detection of and protection against replay attacks.

Messages can be acknowledged or unacknowledged (see Section 3.7.5). An acknowledged message (for example, a Get message) causes a response message; an unacknowledged message (for example, a Status message) does not cause a response.

### 2.3.9 Security

All messages are encrypted and authenticated using two types of keys. One key type is for the network layer communication, such that all communication within a mesh network would use the same network key. The other key type is for application data. Separating the keys for networking and applications allows sensitive access messages (e.g., for access control to a building) to be separated from non-sensitive access messages (e.g., for lighting). There are no unencrypted or unauthenticated messages within a mesh network.

#### 2.3.9.1 Application and network security

Encrypting and authenticating messages at the upper transport layer and network layer is designed to secure communications within the mesh network against eavesdroppers and malicious attacks. Each layer maintains distinct keys to allow separation between application and network entities.

Splitting application keys from network keys enables secure relay transmission of application messages: Relay nodes can authenticate messages at network level without accessing the application data. For example, a light bulb acting as a Relay node should not be able to unlock doors.

This means that nodes can relay access messages using keys derived from the network key without having to know the application key; therefore they would not have the ability to change or understand the application data. It is expected that network keys would be widely known by many nodes within a network, thereby increasing the density of Relay nodes while protecting the different application areas from each other. This requires separate keys for each application. For example, the sensitive door security application would be separated from the non-sensitive doorbell and lighting application.

The application key is used directly along with an associated application key identifier that is used in certain contexts to identify the application used. However, the network key is always used through a key derivation function to generate other keys that are used directly. Examples of such keys include encryption and privacy keys. This allows a single network key to be changed and all the associated values that are derived from that key to be quickly derived. As with the application key, the network key is also used to derive a network key identifier (see Section 3.8.6).

The security model defines three separate keys (the device key (DevKey), the application key (AppKey), and the network key (NetKey)) to secure the messages. When a node is given a key, it is authorized to use that key. A key that is shared between multiple nodes enables any node with that key to transmit and receive messages using that key.

The device key facilitates confidentiality and authentication of key material between a Configuration Client and a single node. The application key facilitates confidentiality and authentication of application data sent between intended nodes. The network key facilitates privacy, confidentiality, and authenticity of network messages. A node may have knowledge of a single device key, multiple application keys, and multiple network keys.

A device key is similar to an application key in that it is designed to secure information sent by an application in the upper transport layer. However, a device key is only known by a Configuration Client and the single node. The Configuration Client knows the device keys for all nodes, which allows the Configuration Client to securely distribute keys to a set of nodes by sending these keys secured with the device key for each individual node, allowing a key distribution to be targeted at only those nodes that need to know. Use of a device key is designed to protect against the “trash-can” attack (a technique to retrieve information from a disposed device that can be used to carry out an attack on a network) by allowing the distribution of new network and application keys to selected devices only.

An application key can only be used with a single network key. This implies that a network key has one or more application keys associated with it. This association is known as the key binding.

The granularity of access layer security is on a per-model basis. Each server model has a set of application keys bound to it, defining the possible keys that should be used to encrypt and authenticate a message to be processed by the model. This allows multiple entities to operate certain node functions. Up to 251 application keys can be bound to a model. For example, a Light Lightness Server Model has three keys bound to it because the admin, user, and guest can all switch on a light. However, only the admin can configure the lamp, so the Configuration Server Model has only the admin application key bound to it.

#### 2.3.9.2 Obfuscation

The network security model utilizes a privacy mechanism called obfuscation that utilizes AES to encrypt the source address, sequence numbers, and other header information using a privacy key. The intent for obfuscation is to make tracking nodes more difficult.

#### 2.3.9.3 Network and application key identifiers

A node may have multiple network or application keys.

By using a key identifier, it is possible to identify which subset of keys are used to secure the message. For example, instead of checking 20 keys, a node may only need to check two keys that have the same least significant bits of the key identifier. If a message is received with a key identifier that is not known, then the node can immediately discard it.

The key identifier is generated from the network or application key using a key derivation function.

This specification defines a separate identifier for the network key and application key. A network key identifier is transmitted in each Network PDU using a 7-bit value, while the application key identifier is transmitted in each Lower Transport PDU using a 6-bit value.

#### 2.3.9.4 Initialization vector index

A Network PDU contains a 24-bit sequence number that allows an element to transmit 16,777,216 Network PDUs. The sequence number is used in the security nonce to provide uniqueness; therefore the sequence number must not wrap. If an element is transmitting a new message at 2 Hz, then these sequence numbers would be exhausted after 97 days. To enable a mesh network to operate for longer periods of time than the sequence number space allows, an additional 4-octet value called the IV Index is defined that is included in the security nonce. For example, using the same 2 Hz message frequency would measure the lifetime of the network using the IV Index in billions of years.

To enable a gradual transition from one IV Index to the next, each Network PDU includes the least significant bit of the IV Index that was used to transmit the message. A node can also use an IV Update procedure to signal to peer nodes that it is updating the IV Index. This procedure takes a minimum of eight days to transition from the old IV Index to the new IV Index, thereby limiting the frequency that a node can transmit messages to 24 Hz. However, a node should not send more than 100 Network PDUs in any 10 second window, so this would typically take approximately 19 days to exhaust.

### 2.3.10 Directed Forwarding

Directed Forwarding is used to discover and maintain paths among nodes in a subnet, so that forwarding of packets may be directed and limited toforwarded by selected nodes in the paths. This avoids forwarding unnecessary packets in the network, which can reduces interference, and eventually increases the network throughput.

Directed Forwarding is based on a hybrid reactive and proactive protocol that is followed to perform discovery and maintenance of paths. Each path is identified by a combination of the addresses of the originator of the path and the originatoraddress of the destination of the path. When a path is established, it is used by the path originator to direct messages toward the path destination. If the path is validated by both sides, it can be also used in the opposite direction by the path destination to direct messages toward the path originator. Every Relay node uses a combination of the source address of the message and of the destination address of the message to determine whether or not it is allowed to forward the message. This method eventually restricts the message forwarding to Relay nodes that form the established path.

### 2.3.11 Friendship

Friendship is used by Low Power nodes to limit the amount of time that they need to listen. If a node cannot receive continuously, then it is possible that it will not receive mesh messages that it should be processing. This includes security updates required for maintaining the security of the network as well as the normal mesh messages.

If the Low Power node does not receive such messages, then it may not operate as desired and it may also fail to keep up-to-date with the latest security state of the network and eventually drop off the network if this security is changed without its knowledge.

Friendship is a special relationship between a Low Power node and one neighboring Friend node. These nodes must be within a single hop of each other and in the same subnet.

Friendship is first established and initiated by the Low Power node; once established, the Friend node performs a number of actions that help reduce the power consumption on the Low Power node. The Friend node maintains a Friend Queue for the Low Power node, which stores all incoming messages addressed to the Low Power node. The Friend node delivers those messages to the Low Power node when requested by the Low Power node. Also, the Friend node delivers security updates to the Low Power node.

When friendship is established between a Low Power node and one Friend node, the two nodes are considered to be “friends”.

A Friend node may be friends with multiple Low Power nodes. A Low Power node can only be friends with a single Friend node.

An example topology of a mesh network illustrating Friend nodes and Low Power nodes is shown and described in Section 2.3.13.

### 2.3.12 Features

The functionality of nodes is determined by the features that they support. All nodes have the ability to transmit and receive mesh messages. Nodes can also optionally support one or more additional features:

- Relay feature – the ability to receive and retransmit mesh messages over the advertising bearer to enable larger networks.
- Proxy feature – the ability to receive and retransmit mesh messages between GATT and advertising bearers.

- Low Power feature – the ability to operate within a mesh network at significantly reduced receiver duty cycles only in conjunction with a node supporting the Friend feature.
- Friend feature – the ability to help a node supporting the Low Power feature to operate by storing messages destined for those nodes.Directed Forwarding feature – the ability to discover and maintain paths among nodes in a subnet.

A node that supports a feature may have that feature enabled or disabled, and the feature, when enabled, may be or may not be in use.

A node supporting the Relay feature may have this feature disabled, but it would still support the Relay feature, it is just that it is not performing the functionality required by that feature. A node that supports the Relay feature and has the Relay feature enabled is known as a Relay node.

A node supporting the Proxy feature may have this feature disabled, but it would still support the Proxy feature, it is just that it is not performing the functionality required by that feature. A node that supports the Proxy feature and has the Proxy feature enabled is known as a Proxy node.

A node supporting the Low Power feature cannot have this feature disabled and must establish a friendship with another node supporting the Friend feature before it can use the Low Power feature to reduce receiver duty cycles. A node that supports the Low Power feature and has a friendship with a node that supports the Friend feature is known as a Low Power node.

A node supporting the Friend feature may have this feature disabled, but it would still support the Friend feature, it is just that it is not performing the functionality required by that feature. A node that supports the Friend feature, has the Friend feature enabled, and has a friendship with a node that supports the Low Power feature is known as a Friend node.

A node supporting the Directed Forwarding feature may have this feature disabled for one or more of the subnets to which it belongs. When the Directed Forwarding feature is disabled, the node still supports the Directed Forwarding feature, but it is not performing the functionality that the feature requires. A node that supports the Directed Forwarding feature and has the Directed Forwarding feature enabled in a subnet is called a Directed Forwarding node in the context of that subnet. A Directed Forwarding node in a subnet that has the Relay feature enabled is called a Directed Relay node in the context of that subnet. A Directed Forwarding node that does not support the Relay feature or that has the Relay feature disabled can only act as a Path Originator or a Path Destination.

A node that supports the Low Power feature may also support the Directed Forwarding feature. A node that supports both the Low Power feature and the Directed Forwarding feature may have the Directed Forwarding feature enabled in a subnet only if the node is not operating as a Low Power node in the context of that subnet.

Directed Forwarding can be performed only among nodes in the subnet that have the Directed Forwarding feature enabled. In some cases, a Directed Forwarding node performs discovery and establishment of paths as a delegate for other nodes. For example, a Low Power node that does not support Directed Forwarding in a subnet may establish friendship with a Friend node that has the

Directed Forwarding feature enabled in the context of that subnet. Then, the Friend node can perform path discovery and path establishment operations on behalf of the Low Power node.

### 2.3.13 Topology

Nodes that support the various features described above can be formed into a mesh network. An illustration of a mesh network is shown in Figure 2.8 below.

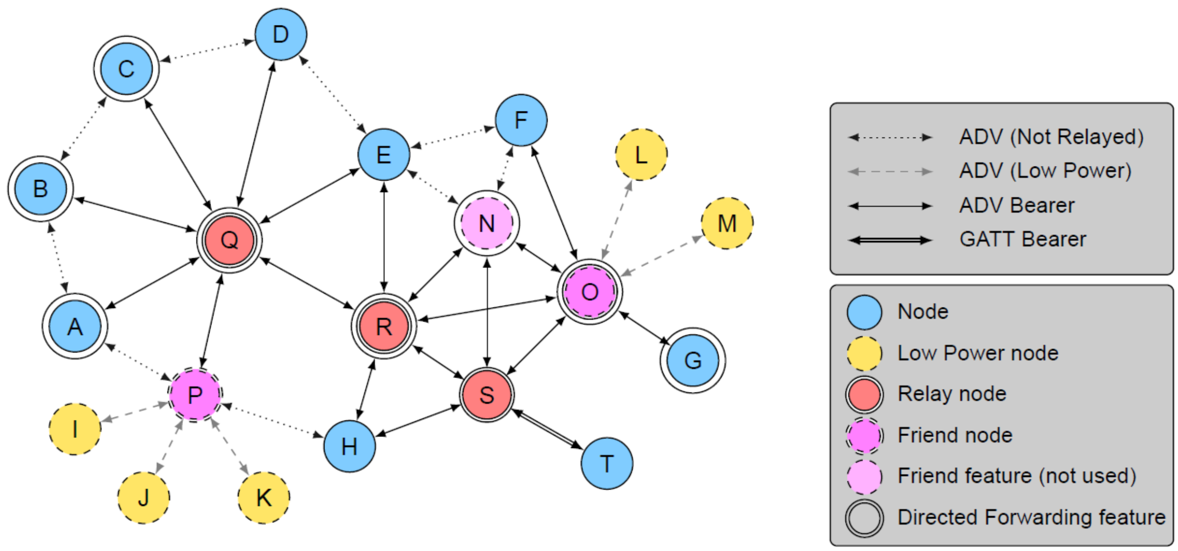


*Figure 2.8: Example topology of a mesh network*

Figure 2.8 shows three Relay nodes: Q, R, and S. Althoughhree nodes support the Friend feature N, O, and P), node N does not have any friendships, and therefore, only O and P are Friend nodes. There are five Low Power nodes: I, J, K, L, and M. Nodes I, J, and K have node P as their friend, while nodes L and M have node O as their friend. Nodes A, B, C, Q, R, N, O, and G all support the Directed Forwarding feature. Node T is only connected to the mesh network using a GATT bearer; therefore, node S must relay all messages to and from node T.

For example, if a message is to be sent from T to L, then T will send the message to node S using the GATT bearer. Node S will retransmit this message using the advertising bearer. Nodes H, R, N, and O are within radio range of node S; therefore they will receive this message. Node O, being the friend of node L will store the message, and if the message was a segmented message, node O will respond with an acknowledgment at the lower transport layer. Sometime later, L will poll node O to check for new messages, such that O will forward the message originally sent by T to L.

## 2.4 Mesh gateway

A mesh gateway is a node that is able to translate messages between the mesh network and a non-Bluetooth technology. A node may be able to send and receive mesh messages through a mesh gateway while not in the range of any of the Relay nodes. This translation is out of scope for this specification.

## 2.5 Concurrency limitations and restrictions

There are no concurrency limitations or restrictions for nodes imposed by this specification.

## 2.6 Topology limitations and restrictions

There are no topology limitations or restrictions imposed by this specification when used with the Bluetooth low energy transport.

# 3 Mesh networking

This section is structured as in the layered architecture that is described in Section 2.1. In addition, there are sections on mesh security and mesh network management.

## 3.1 Conventions

The following conventions apply to this specification.

### 3.1.1 Endianness and field ordering

For the network layer, lower transport layer, upper transport layer, mesh beacons, and Provisioning, all multiple-octet numeric values shall be sent in big endian, as described in Section 3.1.1.1.

For the access layer and Foundation Models, all multiple-octet numeric values shall be little endian as described in Section 3.1.1.2.

Where network data structures are made of multiple fields, the fields are listed in the tables from top to bottom and they appear in the corresponding figures from left to right (i.e., the top row of the table corresponds to the left of the figure). Table 3.1 shows an example data structure made up of multiple fields.

| Field | Size (bits) | Field Content Description |
|---|---|---|
| Field 0 | 4 | The start of this field is in Octet 0 (left most octet in corresponding figure) |
| Field 1 | 12 | The start of this field is in Octet 0 and ends in Octet 1 |
| Field 2 | 16 | The start of this field is in Octet 2 and ends in Octet 3 |

*Table 3.1: Field ordering example*

In order to convert the data structure defined in a table into a series of octets in the layer that uses big endian the following procedure is used. The binary number with N unassigned bits is created. The number of bits N in the number is equal to the sum of the number of bits of every field in the table. The most significant bits (MSb) of the number are set to the value of Field 0 (first row of the table), then the number's unassigned MSbs are set to the value of Field 1. This procedure is continued for consecutive fields of the table and ends when least significant bits (LSb) of the number are set to value of last field of the table. As a final step the number is transmitted in big endian form (i.e., most significant octet first). This is illustrated in Figure 3.1.

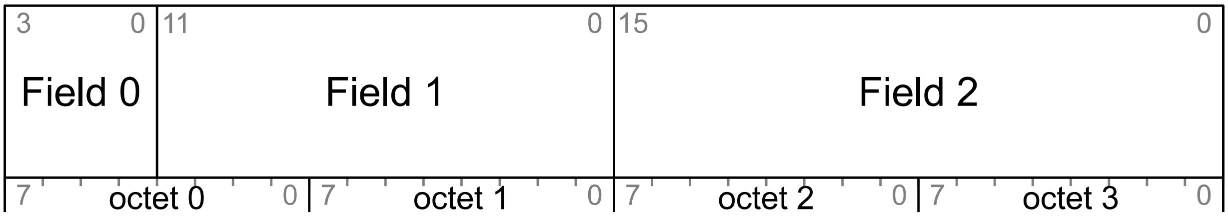


*Figure 3.1: Field ordering example: big endian*

For example, the field 0 is 4 bits wide and has value of 0x6, field 1 is 12 bits wide and has value 0x987, and field 2 is 16 bits wide and has value of 0x1234. The value of the binary number is 0x69871234 and shall be transmitted as 0x69, 0x87, 0x12, 0x34.

In order to convert the data structure defined in a table into a series of octets in the layer that uses little endian the following procedure is used. The binary number with N unassigned bits is created. The number of bits N in the number is equal to the sum of the number of bits of every field in the table. The LSbs of the number are set to the value of Field 0 (first row of the table), then the number's unassigned LSbs are set to the value of Field 1. This procedure is continued for consecutive fields of the table and ends when MSbs of the number are set to the value of last field of the table. As a final step the number is transmitted in little endian form (i.e., least significant octet first). This is illustrated in Figure 3.2.

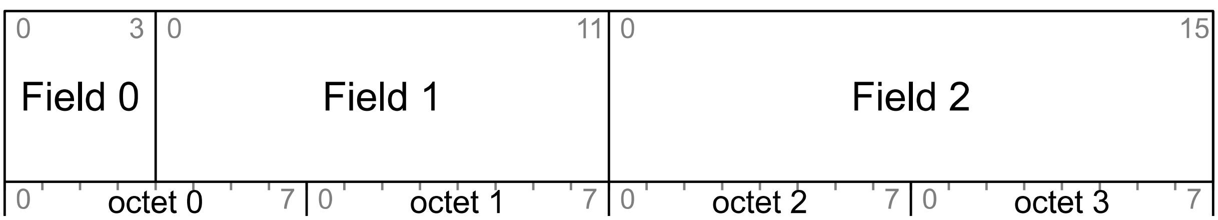


*Figure 3.2: Field ordering example: little endian*

For example, the field 0 is 4 bits wide and has a value of 0x6, field 1 is 12 bits wide and has a value of 0x987, and field 2 is 16 bits wide and has a value of 0x1234. The value of the binary number is 0x12349876 and shall be transmitted as 0x76, 0x98, 0x 34, 0x12.

#### 3.1.1.1 Big endian

When multiple-octet values are defined as sent in "big endian" (also known as "network byte order"), the conventions in this section apply. For example, the value 0x123456 shall be transmitted as 0x12, 0x34, and 0x56 (most significant octet first).

#### 3.1.1.2 Little endian

When multiple-octet values are defined as sent in "little endian", the conventions in this section apply. For example, the value 0x123456 shall be transmitted as 0x56, 0x34, and 0x12 (least significant octet first).

## 3.2 Features

This specification defines five optional features:

- Relay feature (see Section 3.4.6.1)
- Proxy feature (see Section 3.4.6.2)
- Friend feature (see Section 3.6.6.3)
- Low Power feature (see Section 3.6.6.4)
- Directed Forwarding feature (see Section 3.6.8.1)

## 3.3 Bearers

This specification defines two mesh bearers over which mesh messages may be transported:

- An advertising bearer (see Section 3.3.1)
- A GATT bearer (see Section 3.3.2)

A node shall support the advertising bearer or the GATT bearer or both.

Bearers are indexed by 1-octet indexes so that the advertising bearer is represented by the index value 0x01 and the GATT bearer is represented by the index value 0x02.

An unassigned bearer index is a bearer index that does not represent any bearer and is used for bearer index fields that are ignored. The unassigned bearer index shall have the value 0x00.

### 3.3.1 Advertising bearer

When using the advertising bearer, a mesh packet shall be sent in the Advertising Data of a Bluetooth Low Energy advertising PDU using the Mesh Message AD Type identified by «Mesh Message» as defined in [4]. The Mesh Message AD Type contains a Network PDU as defined in Table 3.2.

| Length | AD Type | Contents |
|---|---|---|
| 0xXX | «Mesh Message» | Network PDU |

*Table 3.2: Mesh Message AD Type*

Any advertisement using the Mesh Message AD Type shall be non-connectable and non-scannable undirected advertising events. If a node receives a Mesh Message AD Type in a connectable advertisement or scannable advertising event, the message shall be ignored.

Note: Non-connectable advertisements are used since there is no need to include the Flags AD Type in the advertising packets, thereby enabling two additional octets to be allocated to the Network PDU (see [7]). To lower the probability of packet collisions on all advertising channels, it is recommended to randomize the gap between consecutive packets within an Advertising Event (see [1]).

A device supporting only the advertising bearer should perform passive scanning with a duty cycle as close to 100 percent as possible in order to avoid missing any incoming mesh messages or Provisioning PDUs.

All devices shall support both the GAP Observer role and GAP Broadcaster role.

### 3.3.2 GATT bearer

The GATT bearer is provided to enable devices that are not capable of supporting the advertising bearer to participate in a mesh network. The GATT bearer uses the Proxy protocol (see Section 6) to transmit and receive Proxy PDUs between two devices over a GATT connection.

The GATT bearer uses a characteristic to write to and receive notifications of mesh messages using the attribute protocol.

The GATT bearer defines two roles: a GATT Bearer Client and a GATT Bearer Server.

The GATT Bearer Client shall be a GATT Client. The GATT Bearer Server shall be a GATT Server.

The GATT Bearer Server shall instantiate one and only one Mesh Proxy Service, as defined in Section 7.2.

The GATT Bearer Client shall support the Mesh Proxy Service.

The GATT Bearer Client shall perform primary service discovery using either the GATT *Discover All Primary Services* sub-procedure or the GATT *Discover Primary Services by Service UUID* sub-procedure to discover the Mesh Proxy Service.

As required by GATT, the GATT Bearer Client must be tolerant of additional optional characteristics in the service records of services used with this profile.

The GATT Bearer Client shall use either the GATT *Discover All Characteristics of a Service* sub-procedure or the GATT *Discover Characteristics by UUID* sub-procedure to discover the characteristics of the service.

The GATT Bearer Client shall use the GATT *Discover All Characteristic Descriptors* sub-procedure to discover the characteristic descriptors, which are described in the following sections.

The GATT Bearer Client shall discover the Mesh Proxy Data In characteristic, Mesh Proxy Data Out characteristic and its *Client Characteristic Configuration* descriptor. Once the *Client Characteristic Configuration* descriptor has been discovered, it shall enable notifications using this characteristic.

To send a Proxy PDU (see Section 6.3), the GATT Bearer Client shall use the *Write Without Response* sub-procedure to write the Proxy PDU to the GATT Bearer Server by writing to the Mesh Proxy Data In characteristic.

To receive a Proxy PDU, the GATT Bearer Client shall be able to receive multiple notifications of the Mesh Proxy Data Out characteristic. Each notification contains a single Proxy PDU.

## 3.4 Network layer

The network layer defines the Network PDU format that allows Lower Transport PDUs to be transported by the bearer layer. It decrypts and authenticates and forwards incoming messages received on input interfaces to upper layers and/or output interfaces and encrypts and authenticates and forwards outgoing messages delivering them to output network interfaces.

### 3.4.1 Endianness

All multiple-octet numeric values in this layer shall be sent in “big endian”, as described in Section 3.1.1.1.

### 3.4.2 Addresses

The network layer defines four basic types of addresses: unassigned, unicast, virtual, and group.

Addresses are 16 bits in length and are encoded as defined in Table 3.3 below.

| Values | Address Type |
|---|---|
| 0b0000000000000000 | Unassigned Address |
| 0b0xxxxxxxxxxxxxxx (excluding 0b0000000000000000) | Unicast Address |
| 0b10xxxxxxxxxxxxxx | Virtual Address |
| 0b11xxxxxxxxxxxxxx | Group Address |

*Table 3.3: 16-bit address allocations*

#### 3.4.2.1 Unassigned address

An unassigned address is an address in which the element of a node has not been configured yet or no address has been allocated. The unassigned address shall have the value 0x0000 as shown in Figure 3.3 below. This may be used, for example, to disable message publishing of a model by setting the publish address of a model to the unassigned address.

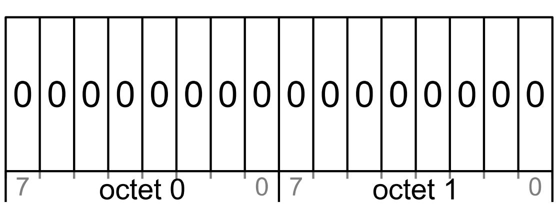

*Figure 3.3: Unassigned address format*

An unassigned address shall not be used in a source or destination address field of a message.

### 3.4.2.2 Unicast address

A unicast address is a unique address allocated to each element. A unicast address has bit 15 set to 0. The unicast address shall not have the value 0x0000, and therefore can have any value from 0x0001 to 0x7FFF inclusive as shown in Figure 3.4 below.

A unicast address is allocated to each element of a node for the lifetime of the node on the network by a Provisioner during provisioning as described in Section 5.4.2.5. The address may be unallocated by a Provisioner to allow the address to be reused using the procedure defined in Section 3.10.7.

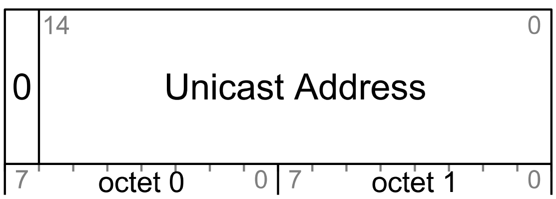


*Figure 3.4: Unicast address format*

A unicast address shall be used in the source address field of a message and may be used in the destination address field of a message. A message sent to a unicast address shall be processed by at most one element.

### 3.4.2.3 Virtual address

A virtual address represents a set of destination addresses. Each virtual address logically represents a Label UUID, which is a 128-bit value that does not have to be managed centrally. One or more elements may be programmed to publish or subscribe to a Label UUID. The Label UUID is not transmitted and shall be used as the Additional Data field of the message integrity check value in the upper transport layer (see Section 3.8.7.1).

The virtual address is a 16-bit value that has bit 15 set to 1, bit 14 set to 0, and bits 13 to 0 set to the value of a hash. This hash is a derivation of the Label UUID such that each hash represents many Label UUIDs.

SALT = s1 (“vtad”)

hash = $\text{AES-CMAC}_{\text{SALT}}$ (Label UUID) mod $2^{14}$

When an Access message is received to a virtual address that has a matching hash, each corresponding Label UUID is used by the upper transport layer as additional data as part of the authentication of the message until a match is found.

Control messages cannot use virtual addresses.

Label UUIDs may be generated randomly as defined in [8]. A Configuration Client may assign and track virtual addresses, however two devices can also create a virtual address using some out-of-band (OOB) mechanism. Unlike group addresses, these could be agreed upon by the devices involved and would not need to be registered in the centralized provisioning database, as they are unlikely to be duplicated.

A disadvantage of virtual addresses is that a multi-segment message is required to transfer a Label UUID to a publishing or subscribing node during configuration.

A virtual address can have any value from 0x8000 to 0xBFFF as shown in Figure 3.5 below.

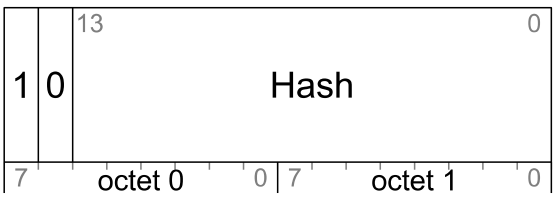


*Figure 3.5: Virtual address format*

Note: When factoring in a 32-bit MIC and the size of the hash, there is only a 1/246 = 1.42 × 10-14 likelihood that two matching virtual addresses using the same application key but different Label UUIDs will collide.

#### 3.4.2.4 Group address

A group address is an address that is programmed into zero or more elements. A group address has bit 15 set to 1 and bit 14 set to 1., as shown in Figure 3.6 below. Group addresses in the range 0xFF00 through 0xFFFF are reserved for Fixed Group addresses (see Table 3.4), and addresses in the range 0xC000 through 0xFEFF are generally available for other usage.

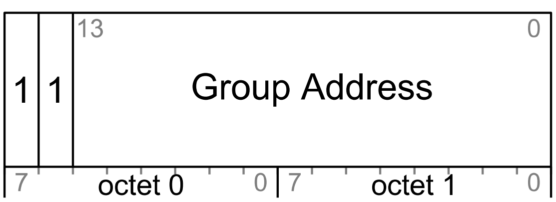


*Figure 3.6: Group address format*

A group address shall only be used in the destination address field of a message. A message sent to a group address shall be delivered to all the instances of models that subscribe to this group address.

There are two types of group address; those that can be assigned dynamically and those that are fixed. The fixed group addresses are defined in Table 3.4 below.

| Values | Fixed Group Address Name |
|---|---|
| 0xFF00–0xFFFA | RFU |

| Values | Fixed Group Address Name |
|---|---|
| 0xFFFB | all-directed-forwarding-nodes |
| 0xFFFC | all-proxies |
| 0xFFFD | all-friends |
| 0xFFFE | all-relays |
| 0xFFFF | all-nodes |

*Table 3.4: Fixed group addresses*

A message sent to the all-directed-forwarding-nodes address shall be processed by the primary element of all nodes that have the Directed Forwarding functionality enabled.

A message sent to the all-proxies address shall be processed by the primary element of all nodes that have the proxy functionality enabled.

A message sent to the all-friends address shall be processed by the primary element of all nodes that have the friend functionality enabled.

A message sent to the all-relays address shall be processed by the primary element of all nodes that have the relay functionality enabled.

A message sent to the all-nodes address shall be processed by the primary element of all nodes.

### 3.4.3 Address validity

Table 3.5 shows which address types are valid for use in the Source Address field and the Destination Address field.

| Address Type | Valid in Source Address Field | Valid in Destination Address Field | |
|---|---|---|---|
| | | **Segmented and Unsegmented Control Messages (see Section 3.5.2)** | **Segmented and Unsegmented Access Messages (see Section 3.5.2)** |
| Unassigned Address | No | No | No |
| Unicast Address | Yes | Yes | Yes |
| Virtual Address | No | No | Yes |
| Group Address | No | Yes | Yes |

*Table 3.5: Address type and message field validity*

Table 3.6 shows which address types are valid for use with device keys and application keys.

| Address Type | Valid with Device Key | Valid with Application Key |
|---|---|---|

| Address Type | Valid with Device Key | Valid with Application Key |
|---|---|---|
| Unassigned Address | No | No |
| Unicast Address | Yes | Yes |
| Virtual Address | No | Yes |
| Group Address | No | Yes |

*Table 3.6: Address type and access layer key type validity*

### 3.4.4 Network PDU

The mesh Network PDU format is defined in Table 3.7 and illustrated in Figure 3.7 below:

| Field Name | Bits | Notes |
|---|---|---|
| IVI | 1 | Least significant bit of IV Index |
| NID | 7 | Value derived from the NetKey used to identify the Encryption Key and Privacy Key used to secure this PDU |
| CTL | 1 | Network Control |
| TTL | 7 | Time To Live |
| SEQ | 24 | Sequence Number |
| SRC | 16 | Source Address |
| DST | 16 | Destination Address |
| TransportPDU | 8 to 128 | Transport Protocol Data Unit |
| NetMIC | 32 or 64 | Message Integrity Check for Network |

*Table 3.7: Network PDU field definitions*

Network PDUs are secured using keys derived from a single network key, as identified by the NID field.

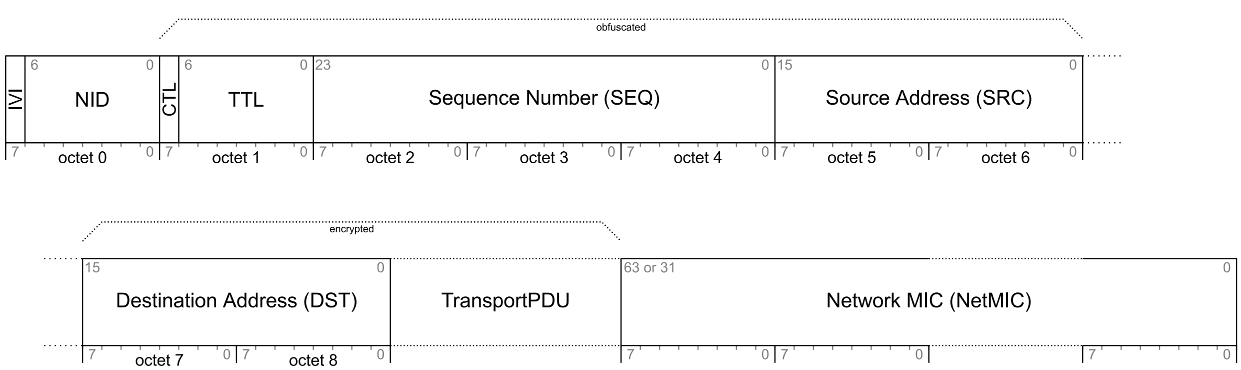


*Figure 3.7: Network PDU format*

#### 3.4.4.1 IVI

The IVI field contains the least significant bit of the IV Index used in the nonce to authenticate and encrypt this Network PDU (see Section 3.8.3).

#### 3.4.4.2 NID

The NID field contains a 7-bit network identifier that allows for an easier lookup of the Encryption Key and Privacy Key used to authenticate and encrypt this Network PDU (see Section 3.8.6.3.1).

The NID value is derived from the network key in conjunction with the Encryption Key and Privacy Key. It is derived differently for main network messages and for private network messages between a Friend and its Low Power node (see Section 3.8.6.3.1).

#### 3.4.4.3 CTL

The CTL field is a 1-bit value that is used to determine if the message is part of a Control Message or an Access Message, as illustrated in Table 3.8.

If the CTL field is set to 0, the NetMIC is a 32-bit value and the Lower Transport PDU contains an Access Message.

If the CTL field is set to 1, the NetMIC is a 64-bit value and the Lower Transport PDU contains a Control Message.

| CTL Field | Message Type | NetMIC Size (bits) |
|---|---|---|
| 0 | Access Message | 32 |
| 1 | Control Message | 64 |

*Table 3.8: CTL field message types and NetMIC sizes*

#### 3.4.4.4 TTL

The TTL field is a 7-bit field. The following values are defined:

- 0 = has not been relayed and will not be relayed
- 1 = may have been relayed, but will not be relayed
- 2 to 126 = may have been relayed and can be relayed
- 127 = has not been relayed and can be relayed

The initial value of this field is set by the transmitting layer (lower transport layer, upper transport layer, access, foundation model, model) or an application and used by the network layer when operating as a Relay node.

The use of the TTL value of zero allows a node to transmit a Network PDU that it knows will not be relayed, and therefore the receiving node can determine that the sending node is a single radio link away. The use of a TTL value of one or larger cannot be used for such a determination.

#### 3.4.4.5 SEQ

The SEQ field is a 24-bit integer that when combined with the IV Index, shall be a unique value for each new Network PDU originated by this node (see Section 3.8.3).

#### 3.4.4.6 SRC

The SRC field is a 16-bit value that identifies the element that originated this Network PDU. This address shall be a unicast address.

The SRC field is set by the originating element and untouched by nodes operating as a Relay node.

#### 3.4.4.7 DST

The DST field is a 16-bit value that identifies the element or elements that this Network PDU is directed towards. This address shall be a unicast address, a group address, or a virtual address.

The DST field is set by the originating node and is untouched by the network layer in nodes operating as a Relay node.

#### 3.4.4.8 TransportPDU

The TransportPDU field, from a network layer point of view, is a sequence of octets of data. When the CTL bit is 0, the TransportPDU field shall be a maximum of 128 bits. When the CTL bit is 1, the TransportPDU field shall be a maximum of 96 bits.

The TransportPDU field is set by the originating lower transport layer and shall not be changed by the network layer.

#### 3.4.4.9 NetMIC

The NetMIC field is a 32-bit or 64-bit field (depending on the value of the CTL bit) that authenticates that the DST and TransportPDU have not been changed.

When the CTL bit is 0, the NetMIC field shall be 32 bits. When the CTL bit is 1, the NetMIC field shall be 64 bits.

The NetMIC is set by the network layer at each node that transmits or relays this Network PDU.

### 3.4.5 Network interfaces

The network layer supports sending and receiving messages via multiple bearers. Multiple instances of a bearer may be present. Each instance of a bearer is connected to the network layer via a network interface. To allow sending messages between elements within the same node the local interface is used.

For example, a node may have three interfaces: one used to send and receive messages via an advertising bearer and two interfaces to a GATT bearer, one for each client connected via a GATT connection.

Interfaces provide input and output filters. Filters may be configured using bearer-specific PDUs or internally by services running on a node, such as the Mesh Proxy Service (see Section 7.2).

#### 3.4.5.1 Interface input filter

Interface input filter decides if an incoming Mesh message is delivered to the network layer for further processing or if it is dropped.

#### 3.4.5.2 Interface output filter

Interface output filter decides if an outgoing Mesh message is delivered to a bearer or if it is dropped.

The output filter of the interface connected to advertising or GATT bearers shall drop all messages with TTL value set to 1.

#### 3.4.5.3 Local Network Interface

A Local Network Interface allows sending messages between elements within the same node.

A node shall implement a Local Network Interface.

Upon receiving a message by a Local Network Interface, the message shall be delivered to all elements of the node.

#### 3.4.5.4 Advertising Bearer Network Interface

The Advertising Bearer Network Interface allows sending messages using the advertising bearer (see Section 3.3.1).

Upon receiving a Network PDU that is not tagged as relay from the network layer and is not using directed security credentials, the Advertising Bearer Network Interface shall retransmit the Network PDU over the advertising bearer using the value of the Network Transmit state (see Section 4.2.19).

Upon receiving a Network PDU that is tagged as relay from the network layer and is not using directed security credentials, the Advertising Bearer Network Interface shall retransmit the Network PDU over the advertising bearer using the value of the Relay Retransmit state (see Section 4.2.20).

Upon receiving a Network PDU that is using directed security credentials, the Advertising Bearer Network Interface shall retransmit the Network PDU over the advertising bearer using the Forwarding Table state (see Section 3.6.8.2) and the value of the RET field in the Neighbor Table (see Section 3.6.8.5). If the Network PDU is directed to a unicast address, it is used the RET value of the Neighbor Table entry with the Neighbor Address (NA) value that equals the address of the next hop toward the destination; if the destination is the Path Destination or the Delegate Path Destination of the path along which the Network PDU is directed, the NA value shall be equal to the NTD value of the Forwarding Table entry associated with the path; if the destination is the Path Originator or the Delegate Path Originator of the path, the NA value shall be equal to the NTO value of the Forwarding Table entry associated with the path. If the Network PDU is directed to a group address or to a virtual address, it is used the maximum value among all the RET values of the Neighbor Table entries with the Neighbor Address (NA) value that equals the address of the next hop toward any node which is subscribed to the group address or to the virtual address: each NA value shall be equal to the NTD value of any Forwarding Table entry that contains the group address or the virtual address in the Subscription List field.

### 3.4.6 Network layer behavior

#### 3.4.6.1 Relay feature

The Relay feature is used to relay/forward Network PDUs received by a node over the advertising bearer. This feature is optional and if supported can be enabled and disabled. If the Proxy feature is supported, then both GATT and advertising bearers shall be supported.

#### 3.4.6.2 Proxy feature

The Proxy feature is used to relay/forward Network PDUs received by a node between GATT and advertising bearers. This feature is optional and if supported can be enabled and disabled.

#### 3.4.6.3 Receiving a Network PDU

A message is delivered from a bearer to the network layer via a network interface. The interface shall apply filtering rules defined by its input filter (see Section 3.4.5.1). If the message passes the input filter, it is delivered to the network layer for further processing.

Each Network PDU that is received can be tagged with additional metadata that can be used later to change the processing of this message.

Upon receiving a message, the node shall check if the value of the NID field value matches one or more known NIDs. If the NID field value does not match a known NID, then the message shall be ignored. If the NID field value matches a known NID, the node shall authenticate the message against each known network key that matched. If the message does not authenticate against any known network key, then the message shall be ignored. If the message does authenticate against a network key, the SRC and DST fields are considered valid (see Section 3.4.3), and the message is not in the Network Message Cache (see Section 3.4.6.5), then the message shall be processed by the lower transport layer.

When a message is retransmitted, as defined below, the IV Index used when retransmitting the message shall be the same as the IV Index when it was received.

If the message delivered from the advertising bearer is processed by the lower transport layer, the Relay feature is supported and enabled, the Directed Forwarding feature is not supported or is supported but not enabled in the subnet from which the Network PDU is received or is supported and enabled in the subnet and the message is secured using the master security material (see Section 3.8.6.3.1), the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field value shall be decremented by 1, the Network PDU shall be tagged as relay, and the

Network PDU shall be retransmitted to all network interfaces connected to the advertising bearer using the master security credentials. It is recommended that a small random delay is introduced between receiving a Network PDU and relaying a Network PDU to avoid collisions between multiple relays that have received the Network PDU at the same time.

If the message delivered from the advertising bearer is processed by the lower transport layer, the Relay feature is supported and enabled, the Directed Forwarding feature is supported and enabled in the subnet from which the Network PDU is received, the message is secured with either friendship security material or directed security material (see Section 3.8.6.3.1), a path for the forwarding of the message exists as defined in Section 3.6.8.2, the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field value shall be decremented by 1, the Network PDU shall be tagged as relay, and the Network PDU shall be retransmitted to all network interfaces connected to the bearers indicated in the Forwarding Table state using the directed security credentials. It is recommended that a small random delay is introduced between receiving a Network PDU and relaying a Network PDU to avoid collisions between multiple relays that have received the Network PDU at the same time.

If the message delivered from the GATT bearer is processed by the lower transport layer, the Proxy feature is supported and enabled, the Directed Forwarding feature is not supported or is supported but not enabled in the subnet from which the Network PDU is received, the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field value shall be decremented by 1, and the Network PDU shall be retransmitted to all network interfaces using the master security credentials.

If the message delivered from the GATT bearer is processed by the lower transport layer, the Proxy feature is supported and enabled, the Directed Forwarding feature is supported and enabled in the subnet from which the Network PDU is received, a path for the forwarding of the message exists as defined in Section 3.6.8.2, the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field value shall be decremented by 1, and the Network PDU shall be retransmitted to all network interfaces connected to the bearers indicated in the Forwarding Table state using the directed security credentials.

If the message delivered from the advertising bearer is processed by the lower transport layer, the Proxy feature is supported and enabled, the Directed Forwarding feature is not supported or is supported but not enabled in the subnet from which the Network PDU is received, the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field shall be decremented by 1 and the Network PDU shall be retransmitted to all network interfaces connected to the GATT bearer using the master security credentials.If the message delivered from the advertising bearer is processed by the lower transport layer, the Proxy feature is supported and enabled, the Directed Forwarding feature is supported and enabled in the subnet from which the Network PDU is received, a path for the forwarding of the message exists as defined in Section 3.6.8.2, the TTL field has a value of 2 or greater, and the destination is not a unicast address of an element on this node, then the TTL field shall be decremented by 1 and the Network PDU shall be retransmitted to all network interfaces connected to the GATT bearer using the directed security credentials.

Figure 3.8 illustrates an example of processing steps for an incoming Network PDU.

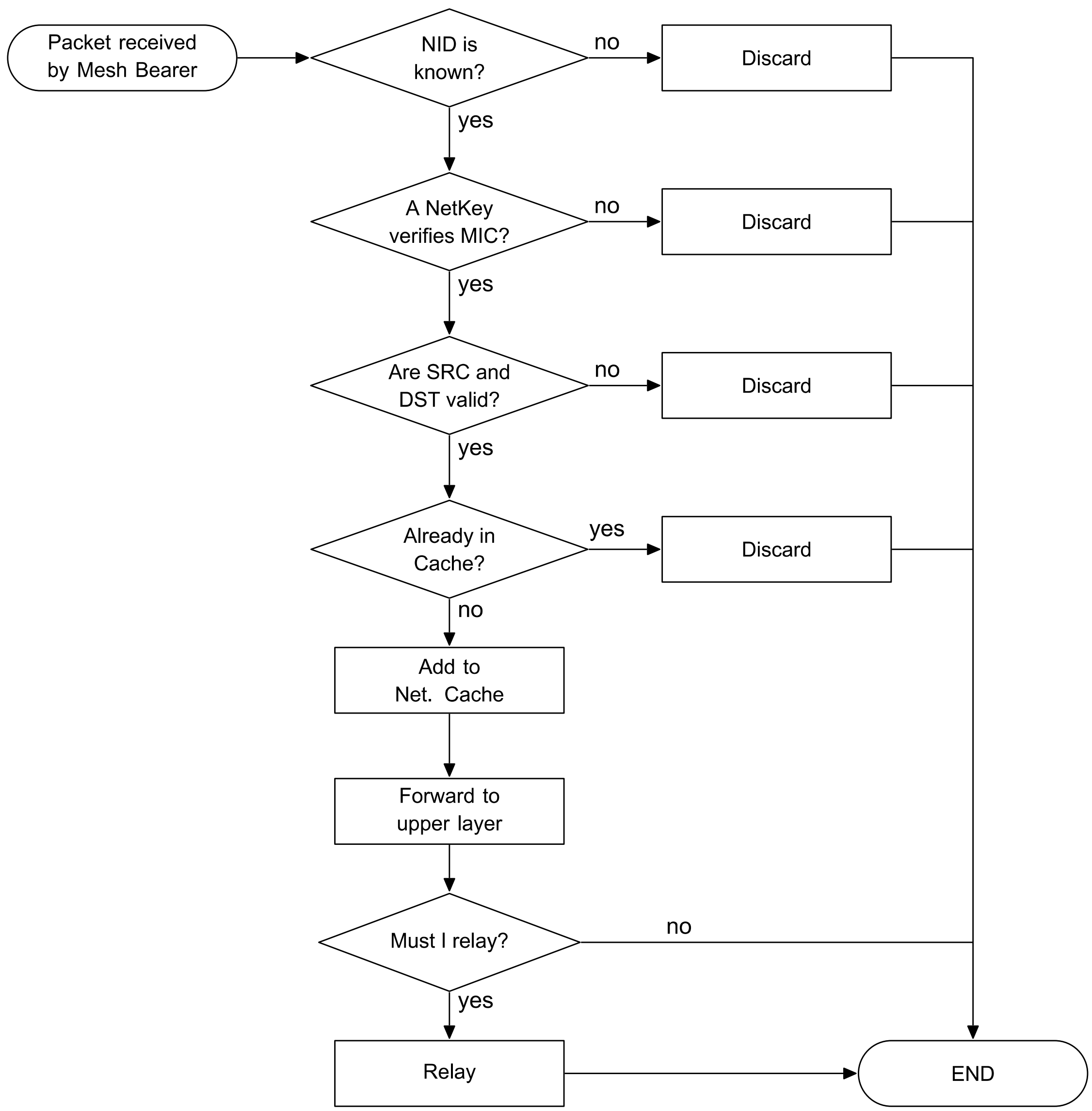


*Figure 3.8: Example of Network PDU processing steps*

### 3.4.6.4 Transmitting a Network PDU

Messages are transmitted by an element in the context of a mesh subnet, which is identified by a unique network key.

The IVI field shall be set to the least significant bit of the IV Index value being used to transmit for the mesh subnet.

The NID field shall be set to the NID value associated with the Encryption Key and Privacy Key used for encryption and obfuscation.

The CTL field shall be set by a higher layer.

The TTL field shall be set by a higher layer.

The SEQ field shall be set by the network layer to the sequence number of the element. The sequence number shall then be incremented by one for every new Network PDU.

The SRC field shall be set by the network layer to the unicast address of the element that is sending this Network PDU.

The DST field shall be set to a unicast address, a group address, or a virtual address to identify the destination element or elements and shall be set by the lower transport, upper transport, or access layer.

The TransportPDU field shall be set by a higher layer.

The NetMIC field shall be set as defined in Section 3.8.7.2.

If the Directed Forwarding feature is supported and enabled in the subnet over which the Network PDU is transmitted, and a path for the forwarding of the message exists as defined in Section 3.6.8.2, then the message shall be secured using the directed security material (see Section 3.8.6.3.1). Otherwise, the message shall be secured using the security material defined for the message (that is, either master security material or friendship security material).

The message shall be delivered to all network interfaces. Each interface shall apply filtering rules defined by its output filter (see Section 3.4.5.2). If the Network PDU passes the output filter, it shall be transmitted on a bearer.

#### 3.4.6.5 Network Message Cache

In order to reduce unnecessary security checks and excessive relaying, a node shall include a Network Message Cache of all recently seen Network PDUs. If a Network PDU is received that is already in the Network Message Cache, then the Network PDU shall not be processed (i.e., it shall be immediately discarded). If a Network PDU is received and that Network PDU is not in the Network Message Cache, then the Network PDU can be processed (e.g., checked against network security), and if it is a valid Network PDU, it shall be stored in the Network Message Cache.

The node is not required to cache the entire Network PDU and may cache only part of it for tracking, such as values for NetMIC, SRC/SEQ or others. However, this is left to the implementation as long as the condition of not processing the same Network PDU more than once is achieved within the limits of the device capabilities.

When the Network Message Cache is full and an incoming new Network PDU needs to be cached, an incoming new Network PDU shall replace the oldest Network PDU that is already in the Network Message Cache.

The Network Message Cache shall be able to store at least two Network PDUs, although it is highly recommended to have a Network Message Cache size appropriate to the anticipated network density. The details of the incoming message processing procedure are left to the implementation.

## 3.5 Lower transport layer

The lower transport layer takes an Upper Transport PDU from the upper transport layer and transmits those messages to a peer lower transport layer. These Upper Transport PDUs may fit into a single Lower Transport PDU, or may be segmented into multiple Lower Transport PDUs. Upon receiving messages, the lower transport layer processes Lower Transport PDUs, reassembling Upper Transport PDUs from possibly multiple PDUs and sending these up to the upper transport layer once reassembly is complete.

### 3.5.1 Endianness

All multiple-octet numeric values in this layer shall be sent in “big endian”, as described in Section 3.1.1.1.

### 3.5.2 Lower Transport PDU

The Lower Transport PDU is used to transmit Upper Transport PDUs to another node.

The most significant bit of the first octet of the Lower Transport PDU is the SEG field, which is used to determine if the Lower Transport PDU is formatted as a segmented or unsegmented message.

There are four formats used, depending on the value of the CTL field in the Network PDU and the SEG field in the Lower Transport PDU as defined in Table 3.9 below.

| CTL Field | SEG Field | Lower Transport PDU Format |
|---|---|---|
| 0 | 0 | Unsegmented Access Message |
| 0 | 1 | Segmented Access Message |
| 1 | 0 | Unsegmented Control Message |
| 1 | 1 | Segmented Control Message |

*Table 3.9: Lower Transport PDU format types*

#### 3.5.2.1 Unsegmented Access message

The Unsegmented Access message is used to transport an Upper Transport Access PDU that fits into a single Network PDU. Figure 3.9 shows an illustration of an Unsegmented Access message, and Table 3.10 shows the fields for this message.


*Figure 3.9: Unsegmented Access message*

| **Field** | **Size (bits)** | **Notes** |
|---|---|---|
| SEG | 1 | 0 = Unsegmented Message |
| AKF | 1 | Application Key Flag |
| AID | 6 | Application key identifier |
| Upper Transport Access PDU | 40 to 120 | The Upper Transport Access PDU |

*Table 3.10: Unsegmented Access message format*

The SEG field shall be set to 0.

The AKF and AID fields shall be set by the upper transport layer according to the application key or device key used to encrypt the access payload (see Section 3.6.4.1).

The Upper Transport Access PDU is supplied by the upper transport layer.

This message does not have a SZMIC field. The TransMIC in the upper transport layer shall be a 32-bit value, as if the SZMIC field has the value 0.

### 3.5.2.2 Segmented Access message

The Segmented Access message is used to transport a segment of an Upper Transport Access PDU. Figure 3.10 shows an illustration of a Segmented Access message, and Table 3.11 shows the fields for this message.

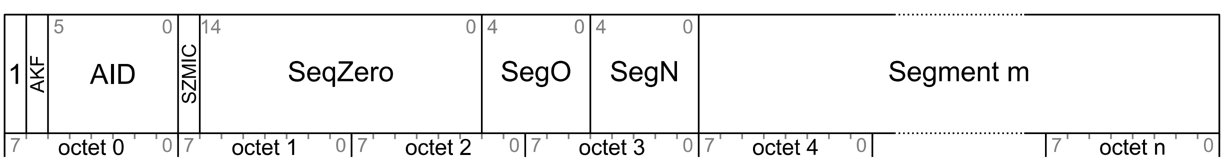


*Figure 3.10: Segmented Access message*

| **Field** | **Size (bits)** | **Notes** |
|---|---|---|
| SEG | 1 | 1 = Segmented Message |
| AKF | 1 | Application Key Flag |
| AID | 6 | Application key identifier |
| SZMIC | 1 | Size of TransMIC |
| SeqZero | 13 | Least significant bits of SeqAuth |

| Field | Size (bits) | Notes |
|---|---|---|
| SegO | 5 | Segment Offset number |
| SegN | 5 | Last Segment number |
| Segment m | 8 to 96 | Segment m of the Upper Transport Access PDU |

*Table 3.11: Segmented Access message format*

The SEG field shall be set to 1.

The SZMIC field indicates the size of the TransMIC in the Upper Transport Access PDU. If the SZMIC field is set to 0, the TransMIC is a 32-bit value. If the SZMIC field is set to 1, the TransMIC is a 64-bit value.

The AKF and AID fields shall be set by the upper transport layer according to the application key or device key used to encrypt the access payload (see Section 3.6.4.1).

The SeqZero field shall be set by the upper transport layer.

The SegO field shall be set to the segment number (zero-based) of the segment m of this Upper Transport PDU.

The SegN field shall be set to the last segment number (zero-based) of this Upper Transport PDU.

The Segment m field, with the segment number m, shall be set to the subset of octets from the Upper Transport Access PDU. For all segments except the last segment, Segment m shall be octets 12*m to 12*m+11. In the last segment, Segment m shall be 12*m through the end of the message.

Every Segmented Access message for the same Upper Transport Access PDU shall have the same values for AKF, AID, SZMIC, SeqZero, and SegN.

### 3.5.2.3 Unsegmented Control Message

The Unsegmented Control Message is used to transport either a Segment Acknowledgment message or a Transport Control message. Figure 3.11 shows an illustration of an Unsegmented Control message, and Table 3.12 shows the fields for this message.

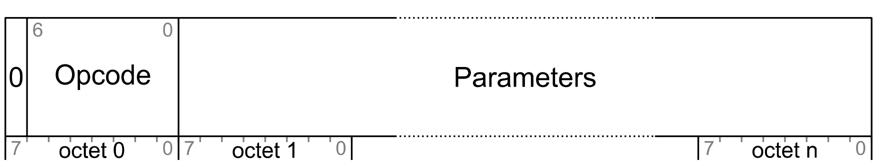


*Figure 3.11: Unsegmented Control message*

| Field | Size (bits) | Notes |
|---|---|---|
| SEG | 1 | 0 = Unsegmented Message |

| Field | Size (bits) | Notes |
|---|---|---|
| | | |
| Opcode | 7 | 0x00 = Segment Acknowledgment<br>0x01 to 0x7F = Opcode of the Transport Control message |
| Parameters | 0 to 88 | Parameters for the Transport Control message |

*Table 3.12: Unsegmented Control message format*

The SEG field shall be set to 0.

The Opcode field shall be set to either 0x00 (for a Segment Acknowledgment message) or the appropriate opcode (see Table 3.59).

The Parameters field is set according to the requirements of the opcode.

### 3.5.2.3.1 Segment Acknowledgment message

The Segment Acknowledgment message is used by the lower transport layer to acknowledge segments received by a peer lower transport layer. The Segment Acknowledgment message is illustrated in Figure 3.12 and defined in Table 3.13.

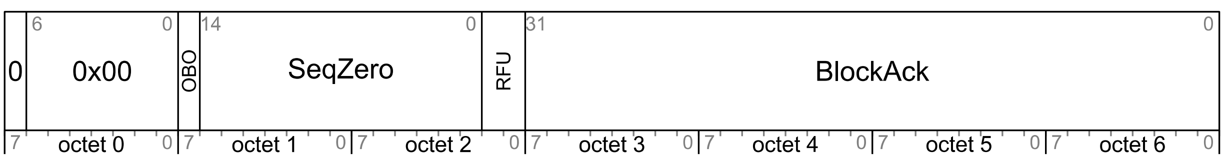


*Figure 3.12: Segment Acknowledgment message*

| Field | Size (bits) | Notes |
|---|---|---|
| SEG | 1 | 0 = Unsegmented Message |
| | | |
| Opcode | 7 | 0x00 = Segment Acknowledgment Message |
| OBO | 1 | Friend on behalf of a Low Power node |
| SeqZero | 13 | SeqZero of the Upper Transport PDU |
| RFU | 2 | Reserved for Future Use |
| BlockAck | 32 | Block acknowledgment for segments |

*Table 3.13: Segment Acknowledgment message format*

The SEG field shall be set to 0.

The Opcode field of the Transport Control message shall be set to 0x00.

The OBO field shall be set to 0 by a node that is directly addressed by the received message and shall be set to 1 by a Friend node that is acknowledging this message on behalf of a Low Power node.

The SeqZero field shall be set to the SeqZero of the upper transport layer message being acknowledged.

The BlockAck field shall be set to indicate the segments received. The least significant bit, bit 0, shall represent segment 0; and the most significant bit, bit 31, shall represent segment 31. If bit n is set to 1, then segment n is being acknowledged. If bit n is set to 0, then segment n is not being acknowledged. Any bits for segments larger than SegN shall be set to 0 and ignored upon receipt.

If the received segments were sent with TTL set to 0, it is recommended that the corresponding Segment Acknowledgment message is sent with TTL set to 0.

### 3.5.2.4 Segmented Control message

The Segmented Control message is used to transport part of a Transport Control message when the Transport Control message will not fit into a single Network PDU.

Figure 3.13 shows an illustration of a Segmented Control message, and Table 3.14 shows the fields for this message.

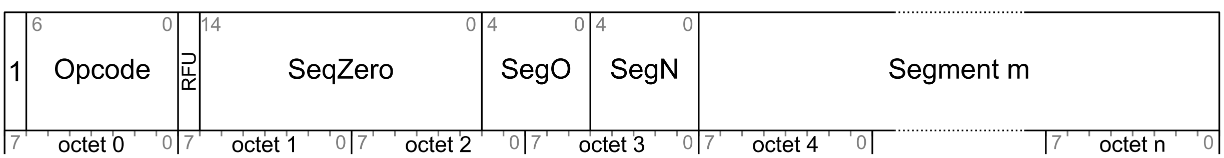


*Figure 3.13: Segmented Control message*

| Field | Size (bits) | Notes |
|---|---|---|
| SEG | 1 | 1 = Segmented Message |
| | | |
| Opcode | 7 | 0x00 = Reserved<br>0x01 to 0x7F = Opcode of the Transport Control message |
| RFU | 1 | Reserved for Future Use |
| SeqZero | 13 | Least significant bits of SeqAuth |
| SegO | 5 | Segment Offset number |
| SegN | 5 | Last Segment number |
| Segment m | 8 to 64 | Segment m of the Upper Transport Control PDU |

*Table 3.14: Segmented Control message format*

The SEG field shall be set to 1.

The Opcode field shall be set by the upper transport layer to indicate the format of the Parameters field. The value 0x00 is Reserved and shall not be transmitted and ignored upon receipt.

The SeqZero field shall be set by the upper transport layer.

The SegO field shall be set to the segment number (zero-based) of the Upper Transport PDU contained within this message.

The SegN field shall be set to the last segment number (zero-based) of this Upper Transport PDU.

The Segment m field shall be set to the subset of octets from the Upper Transport Control PDU. Segment m shall be octets 8*m to 8*m+7, except for the last segment where it is 8*m to the end of the message.

Every Segmented Control message for the same Upper Transport Control PDU shall have the same values for Opcode, SeqZero, and SegN.

### 3.5.3 Segmentation and reassembly

To transmit Upper Transport PDUs larger than 15 octets, the lower transport layer segments and reassembles Upper Transport PDUs. These segments are delivered to the peer lower transport layer using a block acknowledgment scheme to minimize the number of messages that need to be transmitted by the lower transport layer.

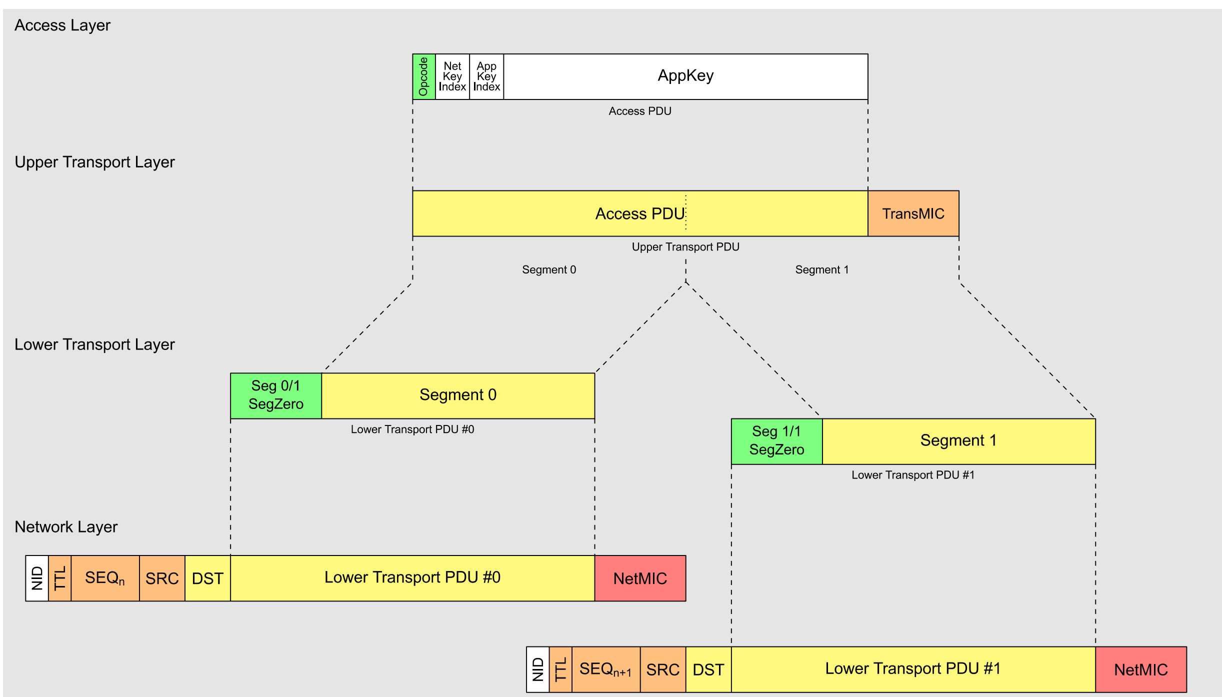


*Figure 3.14: Example of segmentation and reassembly for a two-segment PDU*

Figure 3.14 illustrates an Upper Transport Access PDU being sent that has a single octet opcode, 3 octets for the NetKeyIndexAndAppKeyIndex field, and 16 octets for the AppKey. This means that when encrypted and authenticated with an application key, the Upper Transport PDU is 24 octets. This is segmented by the lower transport layer into two segments, Segment 0 and Segment 1. Each segment has a header that identifies the segment number and is then passed to the network layer, where the complete Network PDU is computed. The network layer then encrypts the Network PDU using the sequence number for that Network PDU and then obfuscates those messages so that only the NID (and IV Index) octet is visible in clear text. Therefore, the single access message can be delivered securely using two Network PDUs.

The process of segmentation for Upper Transport Access PDUs and Upper Transport Control PDUs is identical, and the description below considers these two PDU types to be identical except where explicitly stated.

Note: The segment sizes are different for Upper Transport Access PDUs and Upper Transport Control PDUs.

#### 3.5.3.1 Segmentation

The lower transport layer segments an Upper Transport PDU into one or more Lower Transport PDUs. The lower transport layer shall only transmit Segmented Access messages or Segmented Control messages for a single Upper Transport PDU to the same destination at a time. The lower transport layer should only transmit segmented messages for another Upper Transport PDU once all segments of the last Upper Transport PDU that can be acknowledged have been acknowledged or the message has been canceled.

If the Upper Transport PDU can fit into a single Lower Transport PDU using an Unsegmented Message format, then the lower transport layer should use an unsegmented message to transmit this Upper Transport PDU. If the Upper Transport PDU can fit into a single Lower Transport PDU using a Segmented Message format, then the lower transport layer can use a single segmented message to transmit this Upper Transport PDU. Otherwise, two or more segmented messages shall be used.

Segmented messages are acknowledged at the lower transport layer, but unsegmented messages are not. A single-segment segmented message should be used when delivery of an Upper Transport PDU can be more efficiently transmitted using a segmented message than an unsegmented message.

For example, if a message is sent to acknowledge at the access layer that a multi-segment message has been received and acted upon, and this message is lost, then the complete multi-segment message must be sent again. In contrast, if the application acknowledgment message is sent in a single Segmented Access message, then the application acknowledgment message will be delivered at the lower transport layer, possibly using multiple transmissions of the segment, and thus removing the need to retransmit the multi-segment message again.

Each segment of the Upper Transport Access PDU shall be 12 octets long with the exception of the last segment, which may be shorter.

Each segment of the Upper Transport Control PDU shall be 8 octets long with the exception of the last segment, which may be shorter.

For example, when using a 32-bit TransMIC, if the Upper Transport Access PDU is 42 octets long, then the first 12 octets, octets 0 to 11, are in segment 0; the second set of 12 octets, octets 12 to 23, are in segment 1; the third set of 12 octets, octets 24 to 35, are in segment 2; and the remaining 6 octets, octets 36 to 41, are in segment 3.

For example, if the Upper Transport Control PDU is 42 octets long, then the first 8 octets, octets 0 to 7, are in segment 0; the second set of 8 octets, octets 8 to 15, are in segment 1; the third set of 8 octets, octets 16 to 23, are in segment 2; the fourth set of 8 octets, octets 24 to 31, are in segment 3; the fifth set of 8 octets, octets 32 to 39, are in segment 4; and the remaining 2 octets, octets 40 to 41, are in segment 5.

Each segment of an Upper Transport Access PDU is identified using the SegO field. The segments are linked together using the SeqAuth value used to encrypt and authenticate the Upper Transport Access PDU. Each Lower Transport PDU for an Upper Transport Access PDU shall have the same IV Index as the SeqAuth value used to encrypt and authenticate the Upper Transport Access PDU.

Each segment of an Upper Transport Control PDU is identified using the SegO field. The segments are linked together using the SeqAuth value used to transmit the first segment of the Upper Transport Control PDU. Each Lower Transport PDU for an Upper Transport Control PDU shall have the same IV Index as the SeqAuth value used to identify the Upper Transport Control PDU.

The SeqAuth is composed of the IV Index and the sequence number (SEQ) of the first segment and is therefore a 56-bit value, where the IV Index is the most significant octets and the sequence number is the least significant octets. Only the least significant 13 bits of the value (known as SeqZero) is included in the Segmented message and Segment Acknowledgment message. Upon reassembling a complete Segmented Access message, the SeqAuth value can be derived from the IV Index, SeqZero, and SEQ in any of the segments, by determining the largest SeqAuth value for which SeqZero is between SEQ - 8191 and SEQ inclusive and using the same IV Index. For example, if the received SEQ of a message was 0x647262, the IV Index was 0x58437AF2, and the received SeqZero value was 0x1849, then the SeqAuth value is 0x58437AF2645849. If the received SEQ of a message was 0x647262 and the received SeqZero value was 0x1263, then the SeqAuth value is 0x58437AF2645263.

Because of the limited size of SeqZero, it is not possible to send a segmented message once the SEQ is 8192 higher than SeqAuth. If a segmented message has not been acknowledged by the time that SEQ is 8192 higher than SeqAuth, then the delivery of the Upper Transport PDU shall be canceled.

Each segment of the message includes both its segment offset number and the last segment number.

For example, in the above 42-octet Upper Transport Control PDU, the segments are numbered 0, 1, 2, 3, 4, and 5, and the last segment number is 5. Both the segment number (SegO) and last segment number (SegN) are included in messages to allow a receiver to always determine the size of the Upper Transport PDU (to the nearest 8 octets) after receiving any segment of the message.

### 3.5.3.2 Reassembly

Reassembly is performed in the receiving device. When the Low Power node feature is in use, acknowledgment of messages is performed by a Friend node and the Low Power node will not send Segment Acknowledgment messages.

Upon receiving a Segmented message, the SeqAuth shall be checked to determine if the Upper Transport PDU is being received or has previously been received. If the Segmented message has not been received yet, then the receiving device shall allocate sufficient memory, as determined by the last segment number (SegN), to store the segments of the Upper Transport PDU as they are received and keep track of the segments it has received, and it will then consider that this message is being received.

If the Low Power node feature is not in use, and if the message is destined to a unicast address, and if the node cannot receive this Upper Transport PDU at this time, for example because the node is busy or out of resources sufficient to reassemble this message, then the node shall signal to the source node that it cannot receive this Upper Transport PDU by setting the BlockAck value to 0x00000000.

If the Segmented message is in the process of being received, then the segment number (SegO) shall be used to determine where the Upper Transport PDU octets in this Segmented message shall be placed in the previously allocated memory for the message. The receiver shall update the BlockAck value to record the successful delivery of the segment.

Once all the segments of the Upper Transport PDU for a given SeqZero have been received, the upper transport layer shall check the Upper Transport PDU (see Section 3.6.4.2).

### 3.5.3.3 Segmentation behavior

Once an Upper Transport PDU has been segmented, the lower transport layer will transmit the initial Lower Transport PDUs of each segment of that message. If the message is destined to a unicast address, then the lower transport layer will expect a Segment Acknowledgment message from that node, or from a Friend node on behalf of that node. If the message is addressed to a virtual or group address, then Segment Acknowledgment messages will not be sent by those devices.

If the Lower Transport PDUs of the segmented Upper Transport PDU are being sent to a group address or a virtual address, then the lower transport layer shall send all Lower Transport PDUs of the segmented Upper Transport PDU. It is recommended to send all Lower Transport PDUs multiple times, introducing small random delays between repetitions.

> Note: The Upper Transport PDU sent to a group or virtual address is not acknowledged by recipients, thus the message delivery status is unknown and should be considered to be unacknowledged. The behavior recommended above is designed to significantly increase the probability of the successful delivery of the segmented Upper Transport PDU.

The following requirements apply when Lower Transport PDUs are being sent to a unicast address.

When Lower Transport PDUs are sent, a segment transmission timer shall be started within which time a Segment Acknowledgment message is expected to be received. This timer shall be set to a minimum of 200 + 50 * TTL milliseconds.

If a Segment Acknowledgment message OBO field is set to 0, and the DST field is set to the unicast address of this element, and the SeqZero field is set to SeqZero of the segmented message, and the SRC field is set to the destination of the segmented message, the Segment Acknowledgment message is a valid acknowledgment for the segmented message.

If a Segment Acknowledgment message OBO field is set to 1, and the DST field is set to the unicast address of this element, and the SeqZero field is set to SeqZero of the segmented message, the Segment Acknowledgment message is a valid acknowledgment for the segmented message. For a given SeqAuth, only the Segment Acknowledgment messages from the first SRC address received should be considered valid.

Note: The reception of a Segment Acknowledgment message with the OBO field set to 1 does not mean that the segmented message has been delivered to the final destination, only that it has been delivered to the friend of that Low Power node. The message is stored in the Friend Queue, but the message can be discarded if other messages are received for that Low Power node or the Friendship is terminated.

If a Segment Acknowledgment message is received that is a valid acknowledgment for the segmented message, then the lower transport layer shall reset the segment transmission timer and retransmit all unacknowledged Lower Transport PDUs. If a Segment Acknowledgment message is received that acknowledges all Lower Transport PDUs for the Upper Transport PDU, then the Upper Transport PDU is complete. If a Segment Acknowledgment message with the BlockAck field set to 0x00000000 is received, then the Upper Transport PDU shall be immediately cancelled and the higher layers shall be notified that the Upper Transport PDU has been cancelled.

If the segment transmission timer expires and no valid acknowledgment for the segmented message is received, then the lower transport layer shall retransmit all unacknowledged Lower Transport PDUs.

Note: When retransmitting each Lower Transport PDU, the segment transmission timer is reset and started again.

Each Lower Transport PDU for an Upper Transport PDU shall be transmitted at least two times unless acknowledged earlier. If the lower transport layer stops retransmitting Lower Transport PDUs before all Lower Transport PDUs have been acknowledged, then the Upper Transport PDU is cancelled.

#### 3.5.3.4 Reassembly behavior

This section only applies when the Low Power feature is not in use.

A lower transport layer has a sequence authentication value and a block acknowledgment value for that SeqAuth for each source device.

If the lower transport layer receives a segment for a message with a SeqAuth value less than the sequence authentication value, then it shall ignore that segment. If the lower transport layer receives a segment for a new message, then it shall save the SeqAuth value from that segment as the new sequence authentication value.

Note: The sequence authentication value logically includes the IV Index, so if a Lower Transport PDU is received using a previous IV Index, then this would be a SeqAuth value that is less than the sequence authentication value.

If a lower transport layer receives a segment of a multi-segment message but cannot accept this multi-segment message at this time because it is currently busy or out of resources to accept this message, and if the message is destined to a unicast address, the lower transport layer shall respond with a Segment Acknowledgment message with the BlockAck field set to 0x00000000.

A lower transport layer that receives a segment of a multi-segment message for a SeqAuth value greater than the sequence authentication value shall start an incomplete timer, which defines the maximum amount of time the lower transport layer waits between unique segments of the same transaction. The incomplete timer shall be set to a minimum of 10 seconds.

A lower transport layer that receives a segment of a multi-segment message for a SeqAuth value greater than the sequence authentication value where the destination is a unicast address shall start an acknowledgment timer, which defines the amount of time after which the lower transport layer sends a Segment Acknowledgment message. The acknowledgment timer shall be set to a minimum of 150 + 50 * TTL milliseconds.

If the lower transport layer receives another segment for the sequence authentication value while the acknowledgment timer is inactive, it shall restart the acknowledgment timer.

Note: If the lower transport layer receives any segment for the sequence authentication value while the acknowledgment timer is active, then the acknowledgment timer is not restarted.

If the lower transport layer receives any segment for the sequence authentication while the incomplete timer is active, the incomplete timer shall be restarted.

The lower transport layer shall mark each segment received into a block acknowledgment value that can be later transmitted back to the source node.

When all segments of a Segmented message have been received, the lower transport layer shall send a Segment Acknowledgment message with the BlockAck field set to the block acknowledgment value for the sequence authentication value. It shall cancel the incomplete timer and the acknowledgment timer, and it shall send the reassembled message to the upper transport layer. If the segments were Segmented Access messages, then the reassembled message shall be processed as defined in Section 3.6.4.2. If the segments were Segmented Control messages, then the reassembled message shall be processed as defined in Section 3.6.5.

When the acknowledgment timer expires, the lower transport layer shall send a Segment Acknowledgment message with the BlockAck field set to the block acknowledgment value for the sequence authentication value.

When the incomplete timer expires, the lower transport layer shall consider that the message being received has failed and cancel the acknowledgment timer. Any segment of a canceled message shall be ignored.

If the lower transport layer receives another segment for the sequence authentication value, and the message has already been fully received, then it shall send a Segment Acknowledgment message immediately with the BlockAck field set to the block authentication value for that SeqAuth. If the device is acting as a Friend node for a Low Power node, then it shall reassemble Segmented messages destined for the Low Power node and act as described, except that it shall set the OBO field to 1 in the Segment Acknowledgment message. Otherwise, the OBO field shall be set to 0.

Note: A lower transport layer that receives a Segmented message that is addressed to a group or virtual address does not start the acknowledgment timer and does not send Segment Acknowledgment messages.

#### 3.5.3.5 Low Power feature reassembly behavior

This section only applies when the Low Power feature is in use.

A lower transport layer has a sequence authentication value and a block acknowledgment value for that SeqAuth for each source device.

If the lower transport layer receives a segment for a message with a SeqAuth value less than the sequence authentication value, then it shall ignore that segment. If the lower transport layer receives a segment for a new message, then it shall save the SeqAuth value from that segment as the new sequence authentication value.

If the friendship is terminated (see Section 3.6.6.4.2), then any previously partially received multi-segment message shall be cancelled.

Note: The sequence authentication value logically includes the IV Index, so if a Lower Transport PDU is received using a previous IV Index, then this would be a SeqAuth value that is less than the sequence authentication value.

A lower transport layer that receives a segment of a multi-segment message for a SeqAuth value greater than the sequence authentication value shall start reassembling the new message, and cancel any previously partially received message.

The lower transport layer shall mark each segment received into a block acknowledgment value.

When all segments of a Segmented message have been received, the lower transport layer shall send the reassembled message to the upper transport layer. If the segments were Segmented Access messages, then the reassembled message shall be processed as defined in Section 3.6.4.2. If the segments were Segmented Control messages, then the reassembled message shall be processed as defined in Section 3.6.5.

### 3.5.4 Lower transport layer behavior

#### 3.5.4.1 Transmitting a Lower Transport PDU

If the Directed Forwarding feature is not supported, or if it is supported and not enabled in the subnet over which the Lower Transport PDU is transmitted, or if it is supported and enabled in the subnet and the Lower Transport PDU is not a Segment Acknowledgment message, tn the Lower Transport PDU shall be delivered to the network layer.

If the Directed Forwarding feature is supported and enabled in the subnet and the Lower Transport PDU is a Segment Acknowledgement message, the node shall check whether a path for the forwarding of the message exists as defined in Section 3.6.8.2. If a path entry that meets these requirements exists, the Lower Transport PDU shall be delivered to the network layer. If no existing path entry meets these requirements, the node shall execute a Directed Forwarding Initialization operation (see Section **錯誤! 找不到參照來源。**), and the Lower Transport PDU may be delivered to the network layer before or after establishing the path entry.

#### 3.5.4.2 Receiving a Lower Transport PDU

If the Lower Transport PDU is a Segmented message or a Segment Acknowledgment message, then it shall be processed as defined in Section 3.5.3.4.

If the Lower Transport PDU is an Unsegmented message type, then it shall be processed as defined in Section 3.6.4.2.

If the Lower Transport PDU is a Transport Control PDU, then it shall be processed according to the value of the Opcode field as defined in 3.6.5.

### 3.5.5 Friend Queue

The Friend node shall have a Friend Queue for each friend Low Power node. The Friend Queue stores Lower Transport PDUs for a Low Power node. No field of the Lower Transport PDU shall be changed due to the message being in the Friend Queue. The CTL, TTL, SEQ, SRC, and DST fields shall be stored with the associated Lower Transport PDU.

When a Friend node receives a message that is destined for a friend Low Power node (i.e., the destination of the message is a unicast address of an element of the Low Power node or in the Friend Subscription List) and the TTL field has a value of 2 or greater, then the TTL field value shall be decremented by 1, and the message shall be stored into the Friend Queue.

If the message is a Segmented Access message or a Segmented Control message, then the message shall only be stored into the Friend Queue after the complete Upper Transport PDU has been successfully reassembled and the Friend node has acknowledged the reception of all segments.

If the Friend Queue is full and a new message needs to be stored that is not a Friend Update message, the oldest entries other than a Friend Update message shall be discarded to make room for the new message.

Note: An implementation may have to discard multiple messages to fit the new message into the Friend Queue.

If the message that is being stored is a Segment Acknowledgment message and the Friend Queue contains another Segment Acknowledgment message that has the same source and destination addresses, and the same SeqAuth value, but a lower IV Index or sequence number, then the older Segment Acknowledgment message shall be discarded.

When a Friend node becomes aware of a security update, for example by receiving a valid Secure Network beacon or by having the Key Refresh Phase state changed, it shall add a Friend Update message into the Friend Queue.

When the Low Power node requests a message from the Friend Queue, the oldest entry shall be sent. Once that message has been acknowledged by the Low Power node, that entry shall be discarded.

If the Friend node is polled for a message from a Low Power node using a Friend Poll, and the Friend Queue for that node is empty, then the Friend node shall generate a new Friend Update message and add that message to the Friend Queue before sending the response, so that this Friend Update message can be sent in response to the Friend Poll message.

## 3.6 Upper transport layer

The upper transport layer takes an access payload from the access layer or an internally generated upper transport layer Control message and transmits those messages to a peer upper transport layer.

For messages from the access layer, encryption and authentication of the message is performed using an application key. This allows the receiving upper transport layer to authenticate received messages.

Transport Control messages that are internally generated by the upper transport layer are only encrypted and authenticated at the network layer.

### 3.6.1 Endianness

All multiple-octet numeric values in this layer shall be sent in “big endian”, as described in Section 3.1.1.1.

### 3.6.2 Upper Transport Access PDU

When the CTL field in the Network PDU is 0, the Upper Transport Access PDU contains an access payload and is known as the Upper Transport Access PDU.

The access payload is encrypted using an application key or device key and the encrypted access payload and associated message integrity check value are combined into an Upper Transport Access PDU. The Upper Transport Access PDU fields are shown in Table 3.15 and illustrated in Figure 3.15.

| Field Name | Octets | Notes |
|---|---|---|
| Encrypted Access Payload | 1 to 380 | The encrypted access payload |
| TransMIC | 4 or 8 | The message integrity check value for the access payload |

*Table 3.15: Upper Transport Access PDU fields*

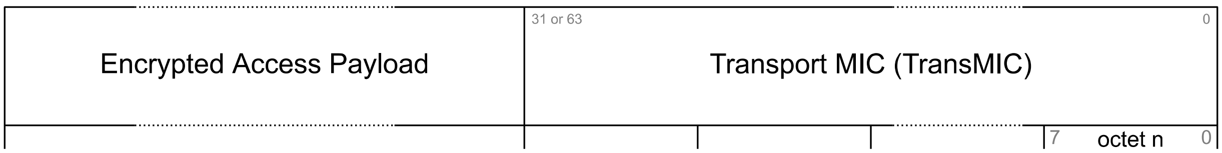


*Figure 3.15: Upper Transport Access PDU format*

### 3.6.2.1 Encrypted access payload

The access payload is supplied by the access layer. If the TransMIC is 32 bits, the access payload can be from a single octet to 380 octets in length. If the TransMIC is 64 bits, the access payload can be from a single octet to 376 octets in length. At the upper transport layer, this field is opaque and no information within this field may be used.

### 3.6.2.2 TransMIC

The Message Integrity Check for Transport (TransMIC) is a 32-bit or 64-bit field that authenticates that the access payload has not been changed. For a segmented message, where SEG is set to 1, the size of the TransMIC is determined by the value of the SZMIC field in the Lower Transport PDU. For unsegmented messages, the size of the TransMIC is 32 bits for data messages.

Note: Control messages do not have a TransMIC.

## 3.6.3 Upper Transport Control PDU

When the CTL bit is 1, the Upper Transport PDU contains a transport control message. A transport control message has a 7-bit opcode that determines the format of the parameters. This Opcode field is not included in the parameters field, but is included in the Lower Transport PDU Unsegmented Control message or in each segment of a Segmented Control Message.

The Upper Transport Control PDU is not authenticated at the upper transport layer and instead relies upon the authentication performed by the network layer. All Upper Transport Control PDUs use a 64-bit NetMIC.

The lower transport layer may segment messages into smaller PDUs for delivery over the network layer. It is therefore recommended to keep Transport Control PDU payload size as reflected in Table 3.16, where the values represent the maximum useful parameter field sizes depending on the number of packets.

| Number of Packets | Transport Control PDU Payload Size |
|---|---|

| | |
|---|---|
| 1 | 11 (Unsegmented) |
| 1 | 8 (Segmented) |
| 2 | 16 |
| 3 | 24 |
| n | n*8 |
| 32 | 256 |

*Table 3.16: Maximum Useful Transport Control PDU payload sizes*

The maximum size of an Upper Transport Control PDU is 256 octets.

## 3.6.4 Upper transport layer behavior

### 3.6.4.1 Transmitting an access payload

All access messages are sent in the context of an application key or a device key. The access payload shall be encrypted using this application key or device key, and the TransMIC shall be set to the message integrity check value, as defined in Section 3.8.6. The Upper Transport PDU shall then be processed as defined in Section 3.5.4.1.

A sequence number (SEQ) shall be allocated to this message. In the context of a message segmented in the lower transport layer, this SEQ corresponds to the 24 lowest bits of SeqAuth, the sequence number used for authenticating and decrypting the access message by the receiver, as defined in chapter 3.5.3.1.

The AKF and AID fields of the Lower Transport PDU shall be set according to the application key or device key used to encrypt and authenticate the Upper Transport PDU. If an application key is used, then the AKF field shall be set to 1 and the AID field shall be set to the application key identifier (AID). If the device key is used, then the AKF field shall be set to 0 and the AID field shall be set to 0b000000.

The upper transport layer shall not transmit a new segmented Upper Transport PDU to a given destination until the previous Upper Transport PDU to that destination has been either completed or cancelled.

### 3.6.4.2 Receiving an Upper Transport PDU

Upon receiving an Upper Transport Access PDU, the access payload shall be decrypted and the TransMIC shall be authenticated against all known application keys or the device key for which the AKF and AID fields match. If the Upper Transport Access PDU authenticates and it has been checked for replay attacks (see Section 3.8.8) then it is delivered to the access layer with the contextual information of this message such as the source address, destination addresses, and the keys used for decryption and authentication.

Upon receiving an Upper Transport Control PDU, the destination address of the PDU shall be checked against the unicast address of the elements of this node and if it matches then the message shall be processed (see Section 3.6.6).

If this node supports the Friend feature and this feature is enabled, the node has a friendship with a Low Power node, and the destination address of this message is an address that is currently in the Friend Subscription List for this Low Power node, then the message shall be stored in the appropriate Friend Queue.

### 3.6.5 Transport Control messages

The Transport Control messages can be transmitted using either a single Unsegmented Control message or sequence of Segmented Control messages. Each of these messages has a 7-bit opcode field that determines the format of the parameters field. Each Transport Control Message shall be sent in the smallest number of Lower Transport PDUs possible.

Opcode 0x00 is terminated at the lower transport layer and used in the Segmentation and Reassembly of messages and shall not be sent by the upper transport layer. All other control messages are terminated at the upper transport layer.

See Table 3.59 for a summary of Transport Control Message opcodes.

#### 3.6.5.1 Friend Poll

The Friend Poll message is sent by a Low Power node to ask the Friend node to send a message that it has stored for the Low Power node.

The Friend Poll message parameters are defined in Table 3.17.

| Field | Size (bits) | Notes |
|---|---|---|
| Padding | 7 | 0b0000000. All other values are Prohibited. |
| FSN | 1 | Friend Sequence Number, used to acknowledge receipt of previous messages from the Friend node to the Low Power node |

*Table 3.17: Friend Poll message parameters*

The Opcode field of the Transport Control message shall be set to 0x01.

The FSN field shall be set to 0 or 1, as defined in Section 3.6.6.4.2.

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the friendship security credentials.

#### 3.6.5.2 Friend Update

The Friend Update message is sent by a Friend node to a Low Power node to inform the Low Power node that the security parameters (see Section 3.6.6.1) for the network have changed or are changing, or that the Friend Queue is empty.

The Friend Update message parameters are defined in Table 3.18.

| Field | Size (octets) | Notes |
|---|---|---|
| Flags | 1 | Contains the IV Update Flag and the Key Refresh Flag |
| IV Index | 4 | The current IV Index value known by the Friend node |
| MD | 1 | Indicates if the Friend Queue is empty or not. |

*Table 3.18: Friend Update message parameters*

The Opcode field of the Transport Control message shall be set to 0x02.

The Flags field is an 8-bit field as defined in Table 3.19:

| Bit | Definition |
|---|---|
| 0 | Key Refresh Flag<br>0: Not-In-Phase2<br>1: In-Phase2 |
| 1 | IV Update Flag<br>0: Normal operation<br>1: IV Update active |
| 2–7 | Reserved for Future Use |

*Table 3.19: Flags field format*

The Key Refresh Flag indicates whether the Key Refresh procedure is in progress (see Section 3.10.4).

The IV Update Flag indicates whether the IV Update procedure is in progress (see Section 3.10.5).

The IV Index field contains the current IV Index.

The MD field is set to indicate if the Friend Queue is empty or not, as defined in Table 3.20.

| Value | Description |
|---|---|
| 0 | Friend Queue is empty |
| 1 | Friend Queue is not empty |
| 2–255 | Prohibited |

*Table 3.20: MD field format*

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the friendship security credentials.

#### 3.6.5.3 Friend Request

The Friend Request message is sent to the all-friends group by a Low Power node to start to find a friend.

The Friend Request message parameters are defined in Table 3.21.

| Field | Size (octets) | Notes |
|---|---|---|
| Criteria | 1 | The criteria that a Friend node should support in order to participate in friendship negotiation |
| ReceiveDelay | 1 | Receive delay requested by the Low Power node |
| PollTimeout | 3 | The initial value of the PollTimeout timer set by the Low Power node |
| PreviousAddress | 2 | Unicast address of the primary element of the previous friend |
| NumElements | 1 | Number of elements in the Low Power node |
| LPNCounter | 2 | Number of Friend Request messages that the Low Power node has sent |

*Table 3.21: Friend Request message parameters*

The Opcode field of the Transport Control message shall be set to 0x03.

The Criteria field format is defined in Table 3.22.

| **Field** | **Size (bits)** | **Notes** |
|---|---|---|
| RFU | 1 | Reserved for Future Use |
| RSSIFactor | 2 | The contribution of the RSSI measured by the Friend node used in Friend Offer Delay calculations |
| ReceiveWindowFactor | 2 | The contribution of the supported Receive Window used in Friend Offer Delay calculations |
| MinQueueSizeLog | 3 | The minimum number of messages that the Friend node can store in its Friend Queue |

*Table 3.22: Criteria field format*

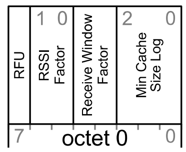


*Figure 3.16: Criteria field format*

The RSSIFactor field is used in the Friend Offer Delay calculation (see Section 3.6.6.3.1) and the values are defined in Table 3.23.

| Value | RSSI Factor |
|---|---|
| 0b00 | 1 |
| 0b01 | 1.5 |
| 0b10 | 2 |
| 0b11 | 2.5 |

*Table 3.23: RSSIFactor field values*

The ReceiveWindowFactor field is used in the Friend Offer Delay calculation (see Section 3.6.6.3.1) and the values are defined in Table 3.24.

| Value | Receive Window Factor |
|---|---|
| 0b00 | 1 |
| 0b01 | 1.5 |
| 0b10 | 2 |
| 0b11 | 2.5 |

*Table 3.24: ReceiveWindowFactor field values*

The MinQueueSizeLog field is defined as $\log_2(N)$, where N is the minimum number of maximum size Lower Transport PDUs that the Friend node can store in its Friend Queue and has one of the values defined in Table 3.25.

| Value | Description |
|---|---|
| 0b000 | Prohibited |
| 0b001 | N = 2 |
| 0b010 | N = 4 |
| 0b011 | N = 8 |
| 0b100 | N = 16 |
| 0b101 | N = 32 |
| 0b110 | N = 64 |

| Value | Description |
|---|---|
| 0b111 | N = 128 |

*Table 3.25: MinQueueSizeLog field values*

The ReceiveDelay field shall be set within the valid range as defined in Table 3.26.

| Value | Description |
|---|---|
| 0x00–0x09 | Prohibited |
| 0x0A–0xFF | Receive Delay in units of 1 millisecond |

*Table 3.26: ReceiveDelay field values*

The PollTimeout field shall be set within the valid range as defined in Table 3.27.

| Value | Description |
|---|---|
| 0x000000–0x000009 | Prohibited |
| 0x00000A–0x34BBFF | PollTimeout in units of 100 milliseconds |
| 0x34BC00–0xFFFFFF | Prohibited |

*Table 3.27: PollTimeout field values*

The PreviousAddress field shall be set to the previous Friend's unicast address or the unassigned address if no previous friendship was established.

The NumElements field shall be set to the number of elements on the Low Power node. The value is used by the Friend node to calculate the range of unicast addresses assigned to the Low Power node using the SRC address of this message.

The values of NumElements are defined in Table 3.28.

| Value | Description |
|---|---|
| 0x00 | Prohibited |
| 0x01–0xFF | Number of elements |

*Table 3.28: NumElements value definitions*

The LPNCounter field value is set to the number of Friend Request messages that the Low Power node has sent to date.

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the master security credentials.

### 3.6.5.4 Friend Offer

The Friend Offer message is sent by a Friend node to offer a friendship.

The Friend Offer message parameters are defined in Table 3.29.

| Field | Size (octets) | Notes |
|---|---|---|
| ReceiveWindow | 1 | Receive Window value supported by the Friend node |
| QueueSize | 1 | Queue Size available on the Friend node |
| SubscriptionListSize | 1 | Size of the Subscription List that can be supported by a Friend node for a Low Power node |
| RSSI | 1 | RSSI measured by the Friend node |
| FriendCounter | 2 | Number of Friend Offer messages that the Friend node has sent |

*Table 3.29: Friend Offer parameters*

The Opcode field of the Transport Control message shall be set to 0x04.

The values of ReceiveWindow are defined in Table 3.30.

| Value | Description |
|---|---|
| 0x00 | Prohibited |
| 0x01–0xFF | Receive Window in units of 1 millisecond |

*Table 3.30: ReceiveWindow value definitions*

The QueueSize field contains the number of messages that the Friend node can store for the Low Power node.

The SubscriptionListSize field contains the number of entries in the subscription list that the Friend node can support for the Low Power node.

The RSSI field contains a signed 8-bit value, and is interpreted as an indication of received signal strength measured in dBm. This is measured by the Friend node on the Friend Request. The value shall be 0x7F (127 dBm) if the signal strength indication is not available.

The FriendCounter field value is set to the number of Friend Offer messages that the Friend node has sent.

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the master security credentials.

#### 3.6.5.5 Friend Clear

The Friend Clear message is sent to a Friend node to inform it about the removal of a friendship.

The Friend Clear message parameters are defined in Table 3.31.

| Field | Size (octets) | Notes |
|---|---|---|
| LPNAddress | 2 | The unicast address of the Low Power node being removed |
| LPNCounter | 2 | Value of the LPNCounter of new relationship |

*Table 3.31: Friend Clear parameters*

The Opcode field of the Transport Control message shall be set to 0x05.

The LPNAddress field shall be set to the unicast address of the Low Power node that is being removed.

The LPNCounter field shall be set to the LPNCounter value from the latest Friend Request used to establish the relationship.

This is a confirmed message and the sending node expects to receive a Friend Clear Confirm message in response. If the Friend node does not receive a response, the new Friend node should resend the message at doubling intervals (2, 4, 8, 16 seconds etc.) until it either receives a response or it reaches the Poll Timeout of the Low Power node.

This message shall be sent using the master security credentials.

#### 3.6.5.6 Friend Clear Confirm

The Friend Clear Confirm message is sent by the old Friend node in response to the Friend Clear message to confirm that the friendship has been terminated. If the Friend Clear message was received with a TTL of 0, the confirmation should use TTL of 0 as well.

The Friend Clear Confirm message parameter is defined in Table 3.32.

| Field | Size (octets) | Notes |
|---|---|---|
| LPNAddress | 2 | The unicast address of the Low Power node being removed |
| LPNCounter | 2 | The value of the LPNCounter of corresponding Friend Clear message |

*Table 3.32: Friend Clear Confirm parameters*

The Opcode field of the Transport Control message shall be set to 0x06.

The LPNAddress field shall be set to the unicast address of the Low Power node that was removed.

The LPNCounter field shall be set to the LPNCounter value from the corresponding Friend Clear message.

The message should only be sent in response to a valid Friend Clear message received within the Poll Timeout of the previous friend relationship, but it should send a Friend Clear Confirm message for each valid Friend Clear message that it receives in that period. A Friend Clear message is considered valid if the result of the subtraction of the value of the LPNCounter field of the Friend Request message (the one that initiated the friendship) from the value of the LPNCounter field of the Friend Clear message, modulo 65536, is in the range 0 to 255 inclusive.

This message shall be sent using the master security credentials.

#### 3.6.5.7 Friend Subscription List Add

The Friend Subscription List Add message is sent by a Low Power node to a Friend node to indicate the list of group addresses and virtual addresses for which messages are to be stored.

The Friend Subscription List Add message parameter is defined in Table 3.33.

| Field | Size (octets) | Notes |
|---|---|---|
| TransactionNumber | 1 | The number for identifying a transaction |
| AddressList | 2 * N | List of group addresses and virtual addresses where N is the number of group addresses and virtual addresses in this message |

*Table 3.33: Friend Subscription List Add parameters*

The Opcode field of the Transport Control message shall be set to 0x07.

The TransactionNumber field is used to distinguish each individual transaction (see Section 3.6.6.4.3).

The AddressList field shall contain a list of group addresses and virtual addresses to add to the Friend Subscription List.

Note: When this message is sent as an Unsegmented Control message, the AddressList field cannot contain more than 5 addresses.

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the friendship security credentials.

#### 3.6.5.8 Friend Subscription List Remove

The Friend Subscription List Remove message is sent by a Low Power node to a Friend node to indicate the group addresses and virtual addresses to remove from the Friend Subscription List.

The Friend Subscription List Remove message parameters are defined in Table 3.34.

| Field | Size (octets) | Notes |
|---|---|---|
| TransactionNumber | 1 | The number for identifying a transaction |
| AddressList | 2 * N | List of group addresses and virtual addresses where N is the number of group addresses and virtual addresses in this message |

*Table 3.34: Friend Subscription List Remove parameters*

The Opcode field of the Transport Control message shall be set to 0x08.

The TransactionNumber field is used to distinguish each individual transaction (see Section 3.6.6.4.3).

The AddressList field shall contain a list of group addresses and virtual addresses to remove from the Friend Subscription List.

Note: When this message is sent as an Unsegmented Control message, the AddressList field cannot contain more than 5 addresses.

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the friendship security credentials.

#### 3.6.5.9 Friend Subscription List Confirm

The Friend Subscription List Confirm message is sent by a Friend node to a Low Power node to respond to a Friend Subscription List Add message or Friend Subscription List Remove message.

The Opcode field of the Transport Control Message shall be set to 0x09.

The Friend Subscription List Confirm message parameters are defined in Table 3.35.

| Field | Size (octets) | Notes |
|---|---|---|
| TransactionNumber | 1 | The number for identifying a transaction |

*Table 3.35: Friend Subscription List Confirm parameters*

The TransactionNumber field is used to distinguish each individual transaction (see Section 3.6.6.4.3).

This message shall set the TTL field of the Network PDU to 0.

This message shall be sent using the friendship security credentials.

#### 3.6.5.10 Heartbeat

The Heartbeat message is sent by a node to let other nodes determine topology of a subnet.

The Heartbeat message parameters are defined in Table 3.36.

| Field | Size (bits) | Notes |
|---|---|---|
| RFU | 1 | Reserved for Future Use |
| InitTTL | 7 | Initial TTL used when sending the message |
| Features | 16 | Bit field of currently active features of the node |

*Table 3.36: Heartbeat parameters*

The Opcode field of the Transport Control message shall be set to 0x0A.

The values of InitTTL are defined in Table 3.37.

| Value | Description |
|---|---|
| 0x00–0x7F | Initial TTL when sending a message |

*Table 3.37: InitTTL value definitions*

The InitTTL field contains the initial TTL used when sending the message.

The Features field contains a bit field indicating the features the node currently uses, as defined in Section 3.1. The format of the Features field is defined in Table 3.38.

| Bit | Feature | Notes |
|---|---|---|
| 0 | Relay | Relay feature in use: 0 = False, 1 = True |
| 1 | Proxy | Proxy feature in use: 0 = False, 1 = True |
| 2 | Friend | Friend feature in use: 0 = False, 1 = True |
| 3 | Low Power | Low Power feature in use: 0 = False, 1 = True |
| 4 | Directed Forwarding | Directed Forwarding feature in use: 0 = False, 1 = True |
| 5–15 | RFU | Reserved for Future Use |

*Table 3.38: Features field format*

### 3.6.5.11 PREQ

The Path Request (PREQ) message is a Transport Control message used to discover a path from a Path Originator to a Path Destination in a subnet. A PREQ message may be segmented.

The PREQ message parameters are defined in 錯誤! 找不到參照來源。. The parameters shall be set by the Path Originator or by the Delegate Path Originator, and all but the OPM field and the PTX field shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination |
| PO | 2 | Path Originator |
| OFN | 1 | Originator Forwarding Number generated by the Path Originator or by the Delegate Path Originator |
| OPM | variable | Originator Path Metric associated with the path to the Path Originator or to the Delegate Path Originator |
| PTX (optional) | 1 | Transmitted Power Level |
| DPO (optional) | 2 | Delegate Path Originator |
| OAR (optional) | 1 | Originator Address Range |

*Table 3.39: PREQ message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOO | 1 | On Behalf Of Originator flag |
| EA | 1 | Extended Address flag |
| WP | 1 | With Transmit Power flag |
| RFU | 5 | Reserved for Future Use |

*Table 3.40: PREQ.OPT field format*

The OBOO flag shall be set to 1 if PREQ is originated by a Delegate Path Originator; otherwise, it shall be set to 0.

The EA flag shall be set to 1 if the Path Originator supports secondary element addresses. If the EA flag is set to 0, the Path Originator supports only the primary unicast address.

The WP field shall be set to 1 if the primary element of the node includes its transmit power in the message. Whether or not the element includes its transmit power in the message is an implementation detail.

The PD field shall be set to the unicast address, group address, or virtual address of the Path Destination; if PD is a unicast address, it is the address of one of the elements of the Path Destination.

The PO field shall be set to the primary unicast address of the Path Originator or of the Low Power node associated with the Delegate Path Originator that is searching for a path on behalf of the Low Power node.

OFN represents the Originator Forwarding Number. If OBOO is set to 0, OFN shall be set to the last Forwarding Number generated by the Path Originator. If OBOO is set to 1, OFN shall be set to the last Forwarding Number generated by the Delegate Path Originator.

OPM represents the Originator Path Metric. If OBOO is set to 0, OPM shall be set to the value of the Path Metric calculated starting from the Path Originator, which is used to rank and select a path to the Path Originator itself, as defined in Section 錯誤! 找不到參照來源。. If OBOO is set to 1, OPM shall be set to the value of the Path Metric calculated starting from the Delegate Path Originator. The size of the OPM field and the calculation of its value performed by each Directed Forwarding node along the path depend on the Path Metric Type state of the Path Metric composite state. The OPM field is set to 0 by the Path Originator or by the Delegate Path Originator.If present, the PTX field shall be set to the value, in dBm, of the transmitted power of the actual PREQ message sender (that is, the Path Originator, the Delegate Path Originator, or an intermediate Directed Relay node). PTX is represented as a signed integer from -127 to 127. This field is present if WP is set to 1.

If present, the DPO field shall be set to the primary unicast address of the Delegate Path Originator. This field is present if OBOO is set to 1.

If present, the OAR field shall be set to the number of secondary element addresses of the Path Originator or of the Low Power node associated with the Delegate Path Originator. This field is present if EA is set to 1.

Note: A Delegate Path Originator has knowledge of the number of element addresses assigned to its associated Low Power node, since it collects this information after processing the Friend Request message from the Low Power node.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the all-directed-forwarding-nodes fixed group address.
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

#### 3.6.5.12 PREP

The Path Reply (PREP) message is a Transport Control message used to reply to a Path Request (PREQ) and establish a one-way path from a Path Originator to a Path Destination in a subnet. A PREP message may be segmented.

The PREP message parameters are defined in 錯誤! 找不到參照來源。. The parameters shall be set by the Path Destination or the Delegate Path Destination, and all shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination |
| PO | 2 | Path Originator or Delegate Path Originator |
| OFN | 1 | Originator Forwarding Number generated by the Path Originator or the Delegate Path Originator |
| OPM | variable | Originator Path Metric associated with the path to the Path Originator or the path to the Delegate Path Originator |
| DPD (optional) | 2 | Delegate Path Destination |
| DAR (optional) | 1 | Destination Address Range |

*Table 3.41: PREP message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOD | 1 | On Behalf Of Destination flag |
| OBOO | 1 | On Behalf Of Originator flag |
| EA | 1 | Extended Address flag |
| CRQ | 1 | Confirmation Requested flag |
| RFU | 4 | Reserved for Future Use |

*Table 3.42: PREP.OPT field format*

The OBOD flag shall be set to 1 if PREP is originated by a Delegate Path Destination; otherwise, it shall be set to 0.

The OBOO flag shall be set to 1 if PREP is sent in response to a PREQ originated by a Delegate Path Originator; otherwise, it shall be set to 0.

The EA flag shall be set to 1 if the Path Destination supports secondary element addresses. If the EA flag is set to 0, the Path Destination supports only the primary unicast address.

The CRQ flag shall be set to 1 if the Path Destination or the Delegate Path Destination requests that the Path Originator or the Delegate Path Originator acknowledge receipt of the PREP message in order to validate the path in both directions; otherwise, it is set to 0.

The PD field shall be set to the primary unicast address of the Path Destination or of the Low Power node associated with the Delegate Path Destination replying to a PREQ on behalf of the Low Power node.

If OBOO is set to 0, PO shall be set to the primary unicast address of the Path Originator; otherwise, PO shall be set to the primary unicast address of the Delegate Path Originator.

OFN represents the Originator Forwarding Number and shall be set to the OFN value reported in the PREQ message.

OPM represents the Originator Path Metric and shall be set to the OPM value reported in the Discovery Table entry that corresponds to the PREQ message (see Section 3.6.8.4.2).

If present, DPD shall be set to the primary unicast address of the Delegate Path Destination. This field is present if OBOD is set to 1.

If present, DAR shall be set to the number of secondary element addresses of the Path Destination or of the Low Power node associated with the Delegate Path Destination. This field is present if EA is set to 1.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the value of the SRC field in the Discovery Table entry in which the OFN field matches the OFN field of the message and either the PO field matches the PO field of the message (if OBOO is set to 0) or the DPO field matches the PO field of the message (if OBOO is set to 1) (see Section 3.6.8.4.2).
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

### 3.6.5.13 **PCONF**

The Path Confirmation (PCONF) message is an unsegmented Transport Control message used to reply to a Path Relay (PREP) and confirm that a two-way path has been established from a Path Originator to a Path Destination in a subnet.

The PCONF message parameters are defined in **錯誤! 找不到參照來源。**. The parameters shall be set by the Path Originator or by the Delegate Path Originator, and all shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination or Delegate Path Destination |
| PO | 2 | Path Originator or Delegate Path Originator |
| OFN | 1 | Originator Forwarding Number generated by the Path Originator or by the Delegate Path Originator |

*Table 3.43: PCONF message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOO | 1 | On Behalf Of Originator flag |
| RFU | 7 | Reserved for Future Use |

*Table 3.44: PCONF.OPT field format*

The OBOO flag shall be set to 1 if the PCONF is originated by a Delegate Path Originator; otherwise, it shall be set to 0.

PD shall be set to the primary unicast address of the Path Destination.

If OBOO is set to 0, PO represents the Path Originator and shall be set to the primary unicast address of the Path Originator. If OBOO is set to 1, PO represents the Delegate Path Originator and shall be set to the primary unicast address of the Delegate Path Originator.

OFN represents the Originator Forwarding Number and shall be set to the OFN value reported in the PREP message.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the value of the NTD field in the Forwarding Table entry in which the OFN field matches the OFN field of the message and either the PO field matches the PO field of the message (if OBOO is set to 0) or the DPO field matches the PO field of the message (if OBOO is set to 1) (see Section 3.6.8.2).

- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

#### 3.6.5.14 PTREQ

The Path Tracing Request (PTREQ) message is a Transport Control message used to trace and validate a path from a Path Originator to a Path Destination in a subnet. A PTREQ message may be segmented.

The PTREQ message parameters are defined in 錯誤! 找不到參照來源。. The parameters shall be set by the Path Originator or by the Delegate Path Originator, and all fields except the OPM field shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination |
| PO | 2 | Path Originator |
| OPM | variable | Originator Path Metric associated with the path to the Path Originator or to the Delegate Path Originator |
| DPO (optional) | 2 | Delegate Path Originator |

*Table 3.45: PTREQ message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOO | 1 | On Behalf Of Originator flag |
| RFU | 7 | Reserved for Future Use |

*Table 3.46: PTREQ.OPT field format*

The OBOO flag shall be set to 1 if PTREQ is originated by a Delegate Path Originator; otherwise, it shall be set to 0.

The PD field shall be set to the primary unicast address of the Path Destination.

The PO field shall be set to the primary unicast address of the Path Originator or of the Low Power node associated with the Delegate Path Originator tracing the path.

OPM represents the Originator Path Metric. If OBOO is set to 0, OPM shall be set to the value of the Path Metric calculated starting from the Path Originator, which is used to rank and select a path to the Path Originator itself, as defined in Section **錯誤! 找不到參照來源。**. If OBOO is set to 1, OPM shall be set to the value of the Path Metric calculated starting from the Delegate Path Originator. The OPM field is set to 0 by the Path Originator or by the Delegate Path Originator. The size of the OPM field and the calculation of its value performed by each Directed Forwarding node along the path depend on the Path Metric Type state of the Path Metric composite state.

If present, DPO shall be set to the primary unicast address of the Delegate Path Originator. This field is present if OBOO is set to 1.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the all-directed-forwarding-nodes fixed group address.
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

#### 3.6.5.15 PTREP

The Path Tracing Reply (PTREP) message is a Transport Control message used to reply to a Path Tracing Request (PTREQ) and confirm that a two-way path exists from a Path Originator to a Path Destination in a subnet. The PTREP message may be segmented.

The PTREP message parameters are defined in **錯誤! 找不到參照來源。**. The parameters shall be set by the Path Destination or by the Delegate Path Destination, and all shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination |
| PO | 2 | Path Originator or Delegate Path Originator |
| OPM | variable | Originator Path Metric associated with the path to the Path Originator or to the Delegate Path Originator |

*Table 3.47: PTREP message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOO | 1 | On Behalf Of Originator flag |
| RFU | 7 | Reserved for Future Use |

*Table 3.48: PTREP.OPT field format*

The OBOO flag shall be set to 1 if PTREP is a reply to a PTREQ originated by a Delegate Path Originator; otherwise, it shall be set to 0.

The PD field shall be set to the primary unicast address of the Path Destination or of the Low Power node associated with the Delegate Path Destination replying to a PTREQ on behalf of the Low Power node.

If OBOO is set to 0, PO shall be set to the primary unicast address of the Path Originator; otherwise, PO shall be set to the primary unicast address of the Delegate Path Originator.

OPM represents the Originator Path Metric, and shall be set to the OPM value reported in the Discovery Table entry that corresponds to the PREQ message (see Section 3.6.8.4.2).

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the value of the SRC field in the Discovery Table entry in which either the PO field matches the PO field of the message (if OBOO is set to 0) or the DPO field matches the PO field of the message (if OBOO is set to 1) (see Section 3.6.8.4.2).
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

#### 3.6.5.16 PAREQ

The Path Assistant Request (PAREQ) is a Transport Control message used to invite another node in a subnet to be part of an existing path from a Path Originator to a Path Destination and assist in the message forwarding for the path. The PAREQ message may be segmented.

The PAREQ message parameters are defined in 錯誤! 找不到參照來源。.

| Field | Size (octets) | Notes |
|---|---|---|

| | | |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Primary unicast address of the Path Destination |
| PO | 2 | Primary unicast address of the Path Originator |
| FN | 1 | Forwarding Number generated by the Path Originator or by the Delegate Path Originator |
| PN | 2 | Primary unicast address of the peer node of the link that needs an assistant node |
| BID | 1 | Index of the bearer to be used for forwarding messages directed to the Path Originator or to the Path Destination through the peer node of the link that needs an assistant node |
| Path Remaining Time | 2 | Time remaining before the path entry expires |
| DPO (optional) | 2 | Primary unicast address of the Delegate Path Originator |
| DPD (optional) | 2 | Primary unicast address of the Delegate Path Destination |
| OAR (optional) | 1 | Range of secondary element addresses known to be assigned to the Path Originator or to the Low Power node associated with the Delegate Path Originator |
| DAR (optional) | 1 | Range of secondary element addresses known to be assigned to the Path Destination or to the Low Power node associated with the Delegate Path Destination |
| Subscription List (optional) | variable (2 * N) | List of group addresses and virtual addresses (N in total) that the Path Destination or the Low Power node associated with the Delegate Path Destination is subscribed to |

*Table 3.49: PAREQ message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| DIR | 1 | Direction of the link that needs an assistant node compared with the direction of the path from the Path Originator to the Path Destination |
| OBOO | 1 | On Behalf Of Originator flag |
| OBOD | 1 | On Behalf Of Destination flag |
| OEA | 1 | Originator Extended Address flag |

| | | |
|---|---|---|
| DEA | 1 | Destination Extended Address flag |
| SL | 1 | Subscription List flag |
| BPV | 1 | BackwardPathValidated flag |
| RFU | 1 | Reserved for Future Use |

*Table 3.50: PAREQ.OPT field format*

The DIR flag shall be set to 1 if the direction of the link from the PAREQ originator to the peer node indicated in the message is the same as the direction of the path from the Path Originator to the Path Destination (that is, if the peer node is the next-hop node for the PAREQ originator toward the Path Destination); DIR shall be set to 0 if the peer node is the next-hop node for the PAREQ originator toward the Path Originator.

The OBOO flag shall be set to 1 if the DPO value in the Forwarding Table entry associated with the path is a unicast address.

The OBOD flag shall be set to 1 if the DPD value in the Forwarding Table entry associated with the path is a unicast address.

The OEA flag shall be set to 1 if the OAR value in the Forwarding Table entry associated with the path is in the range of 0x01 to 0xFE.

The DEA flag shall be set to 1 if the DAR value in the Forwarding Table entry associated with the path is in the range of 0x01 to 0xFE.

The SL flag shall be set to 1 if the Subscription List field in the Forwarding Table entry associated with the path is not empty.

The BPV flag shall be set to the value of the least significant bit of the BackwardPathValidated field in the Forwarding Table entry associated with the path.

PD, PO, and FN shall contain the values of the fields with the same names in the Forwarding Table entry associated with the path.

The PN field shall be set to the primary unicast address of the peer node of the link that needs an assistant node.

The BID field shall be set to the BTD value in the Forwarding Table entry associated with the path if DIR is set to 1; if DIR is set to 0, BID shall be set to the BTO value in the Forwarding Table entry.

Path Remaining Time shall contain the difference between the Path Expiration Time value in the Forwarding Table entry associated with the path and the current time, and shall be represented as specified in 錯誤! 找不到參照來源。.

If present, DPO shall be set to the DPO value in the Forwarding Table entry associated with the path. DPO is present if OBOO is set to 1.

If present, DPD shall be set to the DPD value in the Forwarding Table entry associated with the path. DPD is present if OBOD is set to 1.

If present, OAR shall be set to the OAR value in the Forwarding Table entry associated with the path. OAR is present if OEA is set to 1.

If present, DAR shall be set to the DAR value in the Forwarding Table entry associated with the path. DAR is present if DEA is set to 1.

If present, Subscription List shall contain the list of addresses in the Subscription List field of the Forwarding Table entry associated with the path. Subscription List is present if SL is set to 1.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the primary unicast address of the candidate assistant node (see Section **錯誤! 找不到參照來源。**).
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

### 3.6.5.17 PAREP

The Path Assistant Reply (PAREP) message is a Transport Control message used to notify neighbors of the status of the processing of a previously received Path Assistance Request (PAREQ).

The PAREP message parameters are defined in **錯誤! 找不到參照來源。**.

| Field | Size (octets) | Notes |
| --- | --- | --- |
| Status | 1 | Status Code indicating whether the path assistant request was accepted or rejected (see Table 4.132) |
| OPT | 1 | Options |
| PD | 2 | Primary unicast address of the Path Destination |
| PO | 2 | Primary unicast address of the Path Originator or the Delegate Path Originator |
| FN | 1 | Forwarding Number generated by |

| | | |
|---|---|---|
| | | the Path Originator or the Delegate Path Originator |
| PAI | 2 | Primary unicast address of the initiator of the path assistance operation |
| PN | 2 | Primary unicast address of the peer node of the link that needs an assistant node |

*Table 3.51: PAREP message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| OBOO | 1 | On Behalf Of Originator flag |
| RFU | 7 | Reserved for Future Use |

*Table 3.52: PAREP.OPT field format*

The OBOO flag shall be set equal to the OBOO value in the PAREQ message.

The Status field shall be set according to the value defined in Table 4.132. If the PAREQ message is accepted and processed, Status shall be set to Success. If the PAREQ message is rejected, Status shall be set to the Status Code value that identifies the reason for rejection.

The PAI field shall be set to the SRC value in the PAREQ message.

The PN, PD, and FN fields shall be set to the values of the fields with the same names in the PAREQ message.

The PO field shall be set to the PO value in the PAREQ message if OBOO in the PAREQ message is set to 0; otherwise, it shall be set to the DPO value in the PAREQ message.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the all-directed-forwarding-nodes fixed group address.
- The TTL field shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

### 3.6.5.18 PERR

The Path Error (PERR) message is an unsegmented Transport Control message used to notify other nodes in a subnet that a path from the message sender to a Path Originator or to a Path Destination in the subnet is invalid, possibly as a result of a broken link.

The PERR message parameters are defined in 錯誤! 找不到參照來源。. The parameters shall be set by the message originator and shall be left unchanged by intermediate Directed Relay nodes.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| PD | 2 | Path Destination |
| FN | 1 | Forwarding Number generated by the Path Originator, the Delegate Path Originator, or the Path Destination |
| PO (optional) | 2 | Path Originator |
| DPO (optional) | 2 | Delegate Path Originator |

*Table 3.53: PERR message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| WPO | 1 | With Path Originator flag |
| OBOO | 1 | On Behalf Of Originator flag |
| RFU | 6 | Reserved for Future Use |

*Table 3.54: PERR.OPT field format*

The WPO flag shall be set to 1 if the Forwarding Table entry associated with the broken path has a unicast address as the Path Originator (that is, if the path was discovered with the exchange of PREQ-PREP messages). The WPO flag shall be set to 0, if the Forwarding Table entry associated with the broken path has the unassigned address as the Path Originator (that is, the path was discovered with the exchange of NINFO messages).

The OBOO flag shall be set to 1 if WPO is set to 1 and the broken path was originated by a Delegate Path Originator; otherwise, WPO shall be set to 0.

The PD field shall be set to the primary unicast address of the Path Destination or of the Low Power node whose friendship with a Delegate Path Destination was terminated.

If the transmitting node is the originator of the PERR message, the FN field shall be set to the FN value of the Forwarding Table entry associated with the broken path. If the node receives a PERR message and sends a new PERR message based on the received one, the FN field of the new message shall be set to the FN value of the received message.

If present, PO shall be set to the primary unicast address of the Path Originator. This field is present if WPO is set to 1.

If present, DPO shall be set to the primary unicast address of the Delegate Path Originator. This field is present if WPO is set to 1 and OBOO is set to 1.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the all-directed-forwarding-nodes fixed group address.
- The TTL shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

#### 3.6.5.18.1 Examples of PERR parameters

This section provides some examples with the aim of clarifying how to structure a PERR message according to characteristics of the path that is reported as invalid.

If the path is established with the exchange of PREQ and PREP messages, it is characterized by the presence of a Path Originator and possibly of a Delegate Path Originator and a Delegate Path Destination.

If the path is established between a Path Originator and a Path Destination without a Delegate Path Originator and a Delegate Path Destination, the message’s set of parameters is 6 octets long and includes the mandatory parameters and the PO field. The OPT field is set to 0x80.

If the path is established between a Path Originator and a Path Destination with a Delegate Path Originator, the set of parameters for the message is 8 octets long and includes the mandatory parameters, the PO field, and the DPO field. The OPT field is set to 0xC0.

If the path is established with the exchange of NINFO messages, it is characterized by the absence of both a Path Originator and delegates. In this case, the set of parameters for the message parameters is 4 octets long and includes only the mandatory parameters. The OPT field is set to 0x00.

### 3.6.5.19 NINFO

The Neighbor Information (NINFO) message is a Transport Control message used to announce the presence of a Directed Forwarding node in a subnet, and, if the node is a Directed Relay node, the presence of all of its neighbors that are Directed Forwarding nodes within an area in the subnet indicated by the Neighborhood state minus 1, as defined in Section 錯誤! 找不到參照來源。. A NINFO message may be segmented. The presence of one-hop neighbors of the node, if any exist, can be reported if the Neighborhood state is set to 0x02. The NINFO contains path information per each neighbor—i.e., “(PD, DFN, DAR)”—that the Directed Relay node registered as a path update, as described in Section 錯誤! 找不到參照來源。.

The NINFO message parameters are defined in 錯誤! 找不到參照來源。.

| Field | Size (octets) | Notes |
|---|---|---|
| OPT | 1 | Options |
| NFN | 1 | Node Forwarding Number |
| Wanted RSSI | 1 | Wanted RSSI |
| NAR (optional) | 1 | Node Address Range |
| i-th PD (optional) | 2 | i-th Path Destination |
| i-th DFN (optional) | 1 | i-th Destination Forwarding Number |
| i-th DAR (optional) | 1 | i-th Destination Address Range |

*Table 3.55: NINFO message parameters*

The OPT field format is defined in 錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| NEA | 1 | Node Extended Address flag |
| NDEA | 6 | Number of neighbors that have secondary elements |
| RFU | 1 | Reserved for Future Use |

*Table 3.56: NINFO.OPT field format*

The index “i” in 錯誤! 找不到參照來源。 is an integer from 1 to the number of the neighbors of the node that are reachable in a single hop (given by the value of theNeighborhood state minus 1). However, the reported information in a NINFO shall not exceed the maximum size of an Upper Transport Control PDU (see Section 3.6.3).

The NEA flag shall be set to 1 if the node supports secondary element addresses; otherwise, the NEA flag shall be set to 0.

The NDEA field shall be set to the number of neighbors of the node that are announced in the message that support secondary element addresses. An NDEA value greater than 0 indicates the number of the first neighbor information tuples that are reported in the message (see Section 3.6.8.4), associated with neighbors that support secondary element addresses; the following neighbor information tuples are associated with neighbors that support only the primary unicast address. If the NDEA field is set to 0, all the nodes announced in the message support only primary unicast addresses.

The NFN field shall be set to the last Forwarding Number generated by the node.

The Wanted RSSI field shall be set to the minimum RSSI that the node requires at its antenna, and it is represented as a signed integer from -127 to 127. The assigned value of the Wanted RSSI is an implementation detail.

If present, the NAR field shall be set to the number of secondary element addresses of the node. This field is present if NEA is set to 1.

If present, the i-th PD field shall be set to the primary unicast address of the i-th Path Destination, which is a one-hop neighbor of the node.

If present, the i-th DFN field shall be set to the last Forwarding Number known to have been generated by the i-th Path Destination. This field is present if the i-th PD field is present.

If present, the i-th DAR field shall be set to the number of secondary element addresses of the i-th Path Destination. This field is present if the i-th PD field is present, the NDEA value is greater than 0, and the index “i” is not greater than the NDEA value.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the all-directed-forwarding-nodes fixed group address.
- The TTL shall be set to 0.
- The NID value is derived from the network key of the subnet over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet over which the message is sent.

##### *3.6.5.19.1 Examples of NINFO parameters*

This section provides some examples to demonstrate how to structure a NINFO message according to the values of the Directed Forwarding states and the actual network topology.

錯誤! 找不到參照來源。 illustrates a network of five Directed Forwarding nodes, numbered 1–5 for simplicity, and with corresponding addresses of 0x0010, 0x0020, 0x0030, 0x0040, and 0x0050. The dashed lines indicate neighborhood relationships. For nodes 1 and 2, the following relationships exist within a two-hop distance:

- Node 1 is a one-hop neighbor of node 2 and a two-hop neighbor of nodes 3 and 4.
- Node 2 is a one-hop neighbor of nodes 1, 3, and 4 and a two-hop neighbor of node 5.

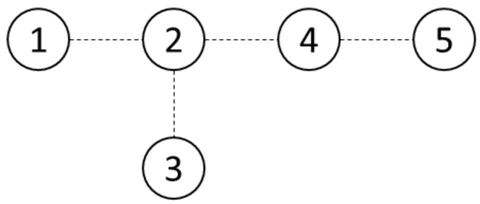


*Figure 3.17: Simple multi-hop network topology*

If the Neighbor Information Options state in node 2 is set to either 0x01 or 0x03 (see Section 錯誤! 找不到參照來源。), this node can send periodic NINFO messages structured according to the values of the Relay state (see Section 4.2.8), the Neighborhood state (see Section 錯誤! 找不到參照來源。), the NEA flag, the NDEA field, and the entries listed in the Forwarding Table state (see Section 錯誤! 找不到參照來源。).

If any one of the following conditions applies …

- The Relay feature is not supported (that is, Relay state is set to 0x02),
- The Relay feature is supported but is not enabled (that is, Relay state is set to 0x00),
- The Relay feature is enabled (that is, Relay state is set to 0x01) and the Neighborhood state is set to either 0x00 or 0x01,
- The Relay feature is enabled, the Neighborhood state is set to 0x02, and the Forwarding Table state does not contain any entries for one-hop neighbors,

… then the NINFO message is structured as any of the following ways:

- If node 2 does not support secondary element addresses, then both the NEA flag and the NDEA field are set to 0, the message fields occupy 3 octets and include only the mandatory fields (that is, OPT, NFN, and Wanted RSSI).
- If node 2 supports secondary element addresses, then the NEA flag is set to 1 and the NDEA field is set to 0, the message fields occupy 4 octets and include the mandatory fields and the NAR field.

If the Relay feature is enabled on the node (that is, the node is a Directed Relay node), the Neighborhood state is set to 0x02, and the Forwarding Table state contains all entries of one-hop neighbors (each one with the DHC field set to 1, according to Section 3.6.8.2), then the NINFO message is structured as any of the following ways:

- If node 2, node 1, node 3, and node 4 do not support secondary element addresses, then both the NEA flag and the NDEA field are set to 0, the message fields occupy 9 octets and include the mandatory fields and a pair of fields (PD, DFN) for each of the one-hop neighbor node 1 (that is, PD set to 0x0010), neighbor node 3 (that is, PD set to 0x0030), and neighbor node 4 (that is, PD set to 0x0040).
- If node 2 supports secondary element addresses, then the NEA flag is set to 1 and the message fields occupy 10 octets and include the mandatory fields, the NAR field, and a pair of fields (PD, DFN) for each of the one-hop neighbors 1 (that is, PD set to 0x0010), 3 (that is, PD set to 0x0030), and 4 (that is, PD set to 0x0040).
- If nodes 1 and 3 support secondary element addresses, then the NDEA field is set to 2 and the message fields occupy 11 octets and include the mandatory fields, a set of fields (PD, DFN, DAR) for each of the one-hop neighbors 1 (that is, PD set to 0x0010) and 3 (that is, PD set to 0x0030), and a pair of fields (PD, DFN) for the one-hop neighbor 4 (that is, PD set to 0x0040).
- If node 1, node 2, node 3, and node 4 support secondary element addresses, then the NEA flag is set to 1, the NDEA field is set to 3, and the message fields occupy 13 octets and include the mandatory fields, the NAR field, and a set of fields (PD, DFN, DAR) for each of the one-hop neighbors 1 (that is, PD set to 0x0010), 3 (that is, PD set to 0x0030), and 4 (that is, PD set to 0x0040).

By extension, if the Forwarding Table state contains only one or two entries associated with one-hop neighbors, then the NINFO message is structured as any of the following ways:

- If both the NEA flag and the NDEA field are set to 0, the message fields occupy 5 octets (if there is one one-hop neighbor entry) or 7 octets (if there are two one-hop neighbor entries).
- If the node 2 supports secondary element addresses, the NEA flag is set to 1 and the message fields occupy 6 octets (if there is one one-hop neighbor entry) or 8 octets (if there are two one-hop neighbor entries).
- If one of the one-hop neighbors supports secondary element addresses, the NDEA field is set to 1 and the message fields occupy 6 octets (if there is one one-hop neighbor entry) or 8 octets (if there are two one-hop neighbor entries).
- If all the one-hop neighbors support secondary element addresses, the NDEA field is set to 2 and the message fields occupy 6 octets (if there is one one-hop neighbor entry) or 9 octets (if there are two one-hop neighbor entries).
- If the node 2 and all the one-hop neighbors support secondary element addresses, the NEA flag is set to 1, the NDEA field is set to 2, and the message fields occupy 7 octets (if there is one one-hop neighbor entry) or 10 octets (if there are two one-hop neighbor entries).

### 3.6.5.20 LREQ

The Link Quality Measurement Request (LREQ) message is an unsegmented Transport Control message that may be used by a Directed Forwarding node to initiate a Link Quality Measurement operation as defined in Section 錯誤! 找不到參照來源。.

The LREQ message parameters are defined in 錯誤! 找不到參照來源。.

| Field | Size (octets) | Notes |
|---|---|---|
| Session ID | 1 | The number used to identify the Link Quality Measurement session |
| PTX | 1 | Transmitted Power Level |
| Wanted RSSI | 1 | The minimum required RSSI value |

*Table 3.57: LREQ message parameters*

The Session ID field is used to distinguish each individual session for a given node (see Section 錯誤! 找不到參照來源。).

The PTX field shall be set to the value in dBm of the transmitted power of the LREQ message sender and is represented as a signed integer from -127 to 127. The assigned value of the PTX field is an implementation detail.

The Wanted RSSI field shall be set to the value, in dBm, of the minimum RSSI that the LREQ message sender requires at its antenna. This valueis represented as a signed integer from -127 to 127. The assigned value of the Wanted RSSI field is an implementation detail.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the primary unicast address of the selected neighbor.
- The TTL shall be set to 0.
- The NID value is derived from the network key of the primary subnet of the node over which the message is sent.

This message shall be sent using the directed security material derived from the network key of the subnet of the node over which the message is sent.

### 3.6.5.21 LREP

The Link Quality Measurement Reply (LREP) message is an unsegmented Transport Control message that shall be generated in response to an Link Quality Measurement Request (LREQ) message.

The LREP message parameters are defined in 錯誤! 找不到參照來源。.

| Field | Size (octets) | Notes |
|---|---|---|
| Session ID | 1 | The number used to identify the Link Quality Measurement session |
| PTX | 1 | Transmitted Power Level |
| Wanted RSSI | 1 | Wanted RSSI |
| Measured RSSI | 1 | RSSI measured upon receiving the LREQ message |

*Table 3.58: LREP message parameters*

The Session ID field shall be set to the value of the corresponding field in the LREQ message.

The PTX field shall be set to the value, in dBm, of the transmitted power of the LREP message sender, is represented as a signed integer from -127 to 127. The assigned value of the PTX field is an implementation detail.

The Wanted RSSI field shall be set to the value, in dBm, of the minimum RSSI that the LREP message sender requires at its antenna, represented as a signed integer from -127 to 127. The assigned value of the Wanted RSSI field is an implementation detail.

The Measured RSSI field shall be set to the value, in dBm, of the RSSI measured by the LREP message sender upon receiving the LREQ message.

This message shall be sent with the following configuration of the Network PDU:

- The SRC field shall be set to the primary unicast address of the transmitting node.
- The DST field shall be set to the value of the SRC field of the LREQ message.
- The TTL shall be set to 0.
- The NID value is derived from the network key of the subnet from which the LREQ message is received.

This message shall be sent using the directed security material derived from the network key of the subnet from which the LREQ message is received.

### 3.6.5.22 Summary of opcodes

Table 3.59 provides a summary of opcodes used by Transport Control messages.

| Value | Opcode | Notes |
|---|---|---|
| 0x00 | – | Reserved for the lower transport layer. |
| 0x01 | Friend Poll | Sent by a Low Power node to its Friend node to request any messages that it has stored for the Low Power node. |
| 0x02 | Friend Update | Sent by a Friend node to a Low Power node to inform it about security updates. |
| 0x03 | Friend Request | Sent by a Low Power node to the all-friends fixed group address to initiate a search for  a friend. |
| 0x04 | Friend Offer | Sent by a Friend node to a Low Power node to offer to become its friend. |
| 0x05 | Friend Clear | Sent to a Friend node to inform a previous friend of a Low Power node about the removal of a friendship. |
| 0x06 | Friend Clear Confirm | Sent from a previous friend to a Friend node to confirm that a prior friend relationship has been removed. |
| 0x07 | Friend Subscription List Add | Sent to a Friend node to add one or more addresses to the Friend Subscription List. |
| 0x08 | Friend Subscription List Remove | Sent to a Friend node to remove one or more addresses from the Friend Subscription List. |
| 0x09 | Friend Subscription List Confirm | Sent by a Friend node to confirm Friend Subscription List updates. |
| 0x0A | Heartbeat | Sent by a node to let other nodes determine topology of a subnet. |
| 0x0B-0x2F | RFU | Reserved for Future Use |
| 0x30 | PREQ | Sent by a Path Originator, a Delegate Path Originator, or a Directed Relay node to members of the all-directed-forwarding-nodes fixed group address to request to discover a path toward a Path Destination. |
| 0x31 | PREP | Sent by a Path Destination, a Delegate Path Destination, or a Directed Relay node to a Directed Forwarding node that is included  in a path toward a Path Originator, to reply to a PREQ and confirm that a one-way path exists from the Path Originator to the Path Destination. |
| 0x32 | PCONF | Sent by a Path Originator, a Delegate Path Originator, or a Directed Relay node to a Directed Forwarding node that is included in a path toward a Path Destination, to reply to a PREP and confirm that a two-way path exists from the Path Originator to the Path Destination. |
| 0x33 | PTREQ | Sent by a Path Originator, a Delegate Path Originator, or a |

| Value | Opcode | Notes |
|---|---|---|
| | | Directed Relay node to the all-directed-forwarding-nodes fixed group address to request to trace and validate a path toward a Path Destination. |
| 0x34 | PTREP | Sent by a Path Destination, a Delegate Path Destination, or a Directed Relay node to a Directed Forwarding node that is included ina path toward a Path Originator, to reply to a PTREQ and confirm that an existing path toward a Path Originator is still valid. |
| 0x35 | PAREQ | Sent by a Directed Forwarding node to one of its neighbors that is a Directed Forwarding node to request to join an existing path from a Path Originator to a Path Destination and assist in message forwarding for the path. |
| 0x36 | PAREP | Sent by a Directed Forwarding node to the all-directed-forwarding-nodes fixed group address to reply to a PAREQ and notify nodes in the group whether the path assistant request is accepted or rejected. |
| 0x37 | PERR | Sent by a Directed Forwarding node to the all-directed-forwarding-nodes fixed group address to notify nodes in the group that a path toward a Path Originator or a Path Destination has become invalid |
| 0x38 | NINFO | Sent by a Directed Forwarding node to the all-directed-forwarding-nodes fixed group address to announce its presence and, if the transmitting node is a Directed Relay node, to announce path updates related to its neighbors. |
| 0x39 | LREQ | Sent by a Directed Forwarding node to one of its neighbors that is a Directed Forwarding node to request to measure the quality of the link with the neighbor. |
| 0x3A | LREP | Sent by a Directed Forwarding node to reply to an LREQ message and report measurements of the quality of the link with the neighbor that sent the LREQ. |
| 0x3B-0x7F | RFU | Reserved for Future Use |

*Table 3.59: Summary of Transport Control message opcodes*

### 3.6.6 Friendship

A Friend node can store messages for a Low Power node.

### 3.6.6.1 Functional overview

In order to optimize the power consumption of a Low Power node, a polling mechanism is used to minimize the Low Power node's receive window. This allows a Low Power node to receive updates from a Friend node by indicating when it is available to receive messages.

Friendship defines timing parameters that are static for the duration of a friend relationship between a Low Power node and a Friend node.

The following timing parameters are used in a friendship:

- ReceiveDelay
- ReceiveWindow
- PollTimeout

The ReceiveDelay is the time between the Low Power node sending a request and listening for a response. This delay allows the Friend node time to prepare the response.

The ReceiveWindow is the time that the Low Power node listens for a response. When the Low Power node receives a message from its Friend node, it can stop listening for additional messages.

The request can be a Friend Poll message, a Friend Subscription List Add message, or a Friend Subscription List Remove message. The response to a Friend Poll message can be a Friend Update message or a stored message. The response to a Friend Subscription List Add message or a Friend Subscription List Remove message is a Friend Subscription List Confirm message.

The timing parameters are illustrated in Figure 3.18.

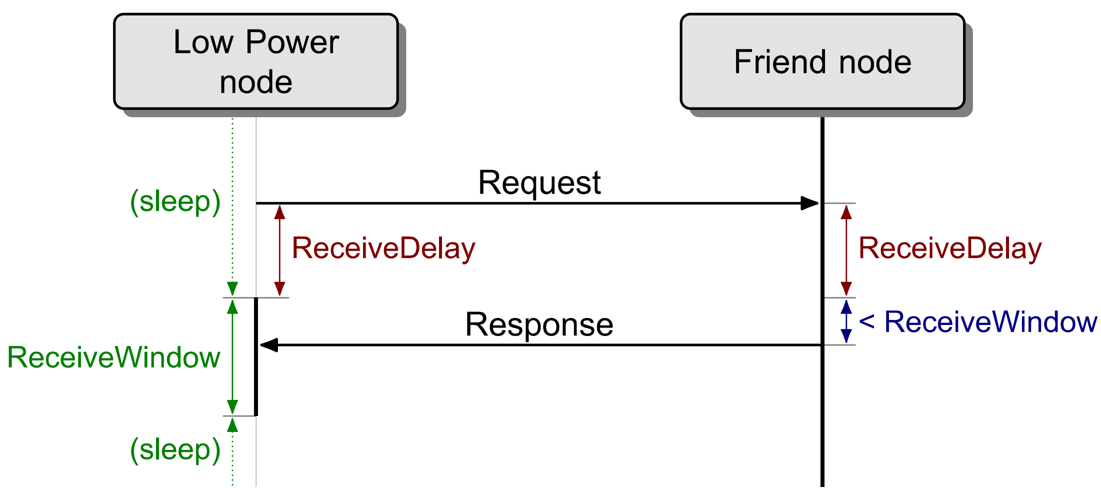


*Figure 3.18: Friendship timing parameters*

PollTimeout timer is used to measure time between two consecutive requests sent by the Low Power node. If no requests are received by the Friend node before the PollTimeout timer expires, then the friendship is considered terminated. This is illustrated in Figure 3.19.

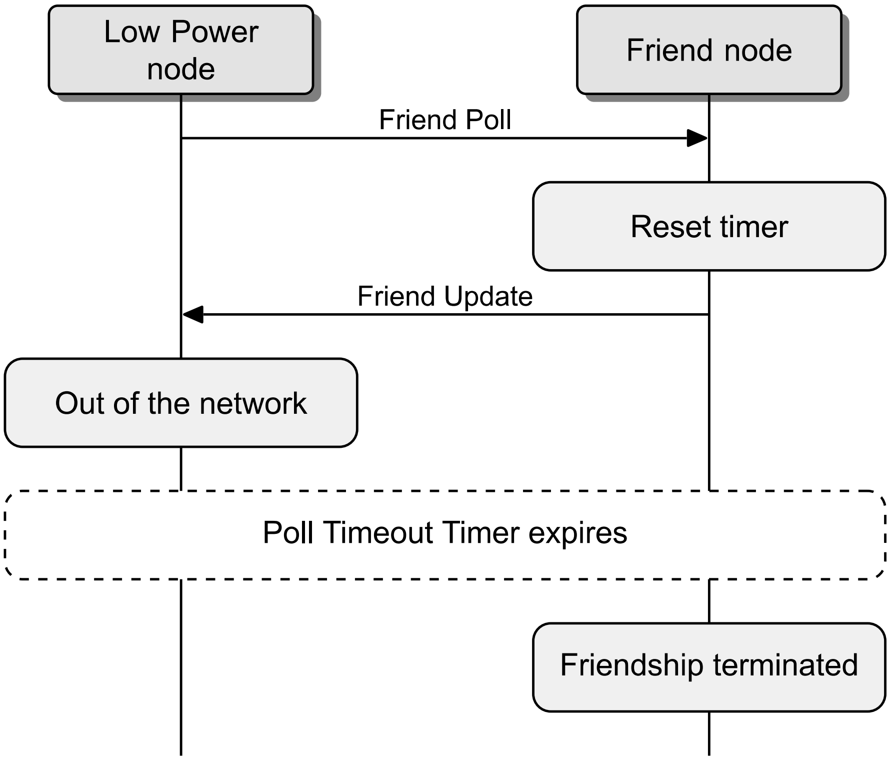


*Figure 3.19: Poll Timeout timer illustration*

To establish a friendship, a node that supports the Low Power feature sends a Friend Request to the all-friends address. The message is picked up by all nodes within radio range of this node that support the Friend feature.

The Friend Request message includes a number of parameters that define the requirements that this node requires any future Friend node to support. Each of the nodes that support the Friend feature and that can support the requirements of the Friend Request message responds by sending back a Friend Offer message to the requesting node. The offers also include additional information about the capabilities of each of the offering nodes. This allows the Low Power node to determine which of these offers it will accept.

The Low Power node then sends a Friend Poll message to its chosen Friend node, and the Friend node responds with a Friend Update message. At this point, the friendship is established, as illustrated in Figure 3.20.

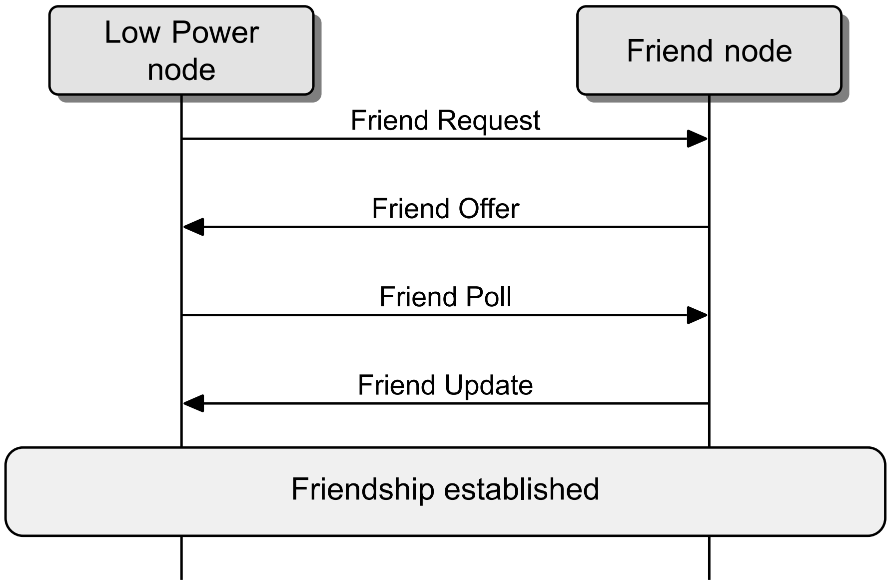


*Figure 3.20: Establishment of a friendship*

If the Low Power node was previously a friend of another Friend node, then the new Friend node informs the old Friend node that it is now the current friend of this Low Power node (see Section 3.6.6.3.1).

After a friendship is established, the Friend node stores a Friend Subscription List for the Low Power node, which is a collection of group and virtual addresses to which the Low Power node is subscribed. This list allows the Friend node to store the messages that the Low Power node is subscribed to.

The values of the IV Index, IV Update flag, and Key Refresh Flag for a given NetKey are known as security parameters. Whenever at least one of the values of the security parameters is changed, the new security parameters are known as security updates.

The Friend node stores all messages for the Low Power node, and also the most recent security updates for the Low Power node, in the Friend Queue. Collectively these are known as stored messages.

To obtain stored messages, the Low Power node sends Friend Poll messages and the Friend node replies with stored messages.

Messages stored in the Friend Queue are delivered to the Low Power node with acknowledgment and in order. To enable this, a Friend Sequence Number is used. This value is stored on the Low Power node and is sent in each Friend Poll message. When the Low Power node receives a valid message in response to a Friend Poll, the Low Power node shall change this Friend Sequence Number so that the next time this Low Power node polls, the Friend node can send the next message. If there was no response to the Friend Poll message, then the Low Power node does not change the Friend Sequence Number and the Friend node can then determine that the last message it sent was not received and resend it.

### 3.6.6.2 Friendship security

When a friendship has been established between a Friend node and a Low Power node, all network messages exchanged by the two parties are secured using the friendship security material created from the friendship security credentials (see Section 3.8.6.3.1). The friendship security credentials are exchanged during the Low Power Establishment procedure (see Section 3.6.6.4.1).

The Friend Poll, Friend Update, and Friend Subscription List Add/Remove/Confirm messages, as well as stored messages that the Friend node delivers to the Low Power node, are always encrypted using the friendship security credentials. If the Friend node supports the Directed Forwarding feature and has the feature enabled, friendship security credentials are also always used to encrypt messages that the Low Power node sends to the Friend node to be directed over a path toward a Path Destination (see Sections 3.6.8 and 錯誤! 找不到參照來源。). The Friend Clear and Friend Clear Confirm messages are always encrypted using the master security credentials.

Depending on the value of the Publish Friendship Credentials Flag (see Section 4.2.2.4), the Low Power node sends a message using either the friendship security credentials or the master security credentials (see Section 3.8.6.3.1).

Figure 3.21 illustrates messages secured using the friendship security credentials (dashed lines) and with the master security credentials (solid lines). The Low Power node starts by sending the Friend Request message using the master security credentials. A Friend node responds with a Friend Offer message, again using the master security credentials. Both the Low Power node and the Friend node must use the master security credentials as neither device is in a friendship with the other and therefore cannot use the friendship security credentials. The Low Power node accepts the offer of friendship and sends a Friend Poll to confirm this using the friendship security credentials. The Friend node will respond to this using a Friend Update message. The Low Power node can now configure the friend subscription list by using the Friend Subscription List Add message, confirmed using the Friend Subscription List Confirm message from the Friend node. Both of these messages are sent using the friendship security credentials.

Sometime later, the Friend node receives a message (InMsg) from another device that needs to be delivered to the Low Power node, so it will store this message in the Friend Queue. The Low Power node sends a Friend Poll message, secured using the friendship security credentials, to which the Friend node will reply with the stored InMsg.

The Low Power node then decides to send two messages: OutMsg1 and OutMsg2. OutMsg1 is sent secured using the friend security credentials and therefore only the Friend node will receive and relay this message. When the Friend node relays OutMsg1, the message will be retransmitted using the master security credentials. OutMsg2 is sent secured using the master security credentials and therefore the Friend node and any other Relay node within range of the Low Power node can relay the message. OutMsg2, when it is relayed, will be retransmitted using the master security credentials.

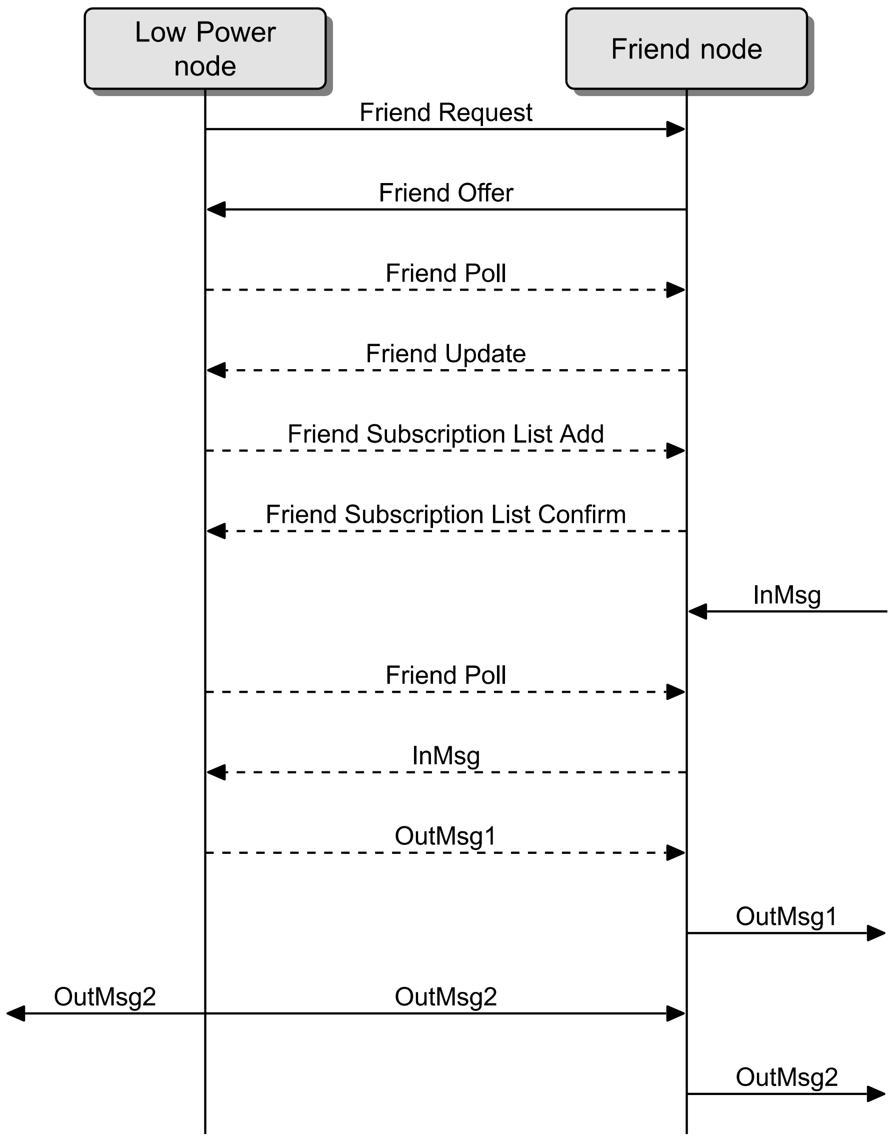


*Figure 3.21: Messages secured with friendship and master security credentials*

### 3.6.6.3 Friend feature

The Friend feature defines three mandatory operations: friend establishment, friend messaging, and friend management.

All Transport Control messages originated by a Friend node shall be sent as unsegmented Control messages with the SRC field set to the unicast address of the primary element of the Friend node.

#### 3.6.6.3.1 Friend establishment

A node that supports the Friend feature and has the feature enabled, and that receives a Friend Request message (see Section 3.6.5.3), that fulfills the minimum requirements specified by the message parameters shall respond with a Friend Offer message (see Section 3.6.5.4).

If the source address of the Friend Request message is an address of a Low Power node that is currently in a friendship with the Friend node, then the Friend node shall also consider the existing friendship with that Low Power node terminated.

The Friend Offer message shall be sent with the destination equal to the source address of the Friend Request message and the TTL value set to 0.

The node shall keep a Friend node counter (FriendCounter), which is a 2-octet value initialized to 0. This value shall be sent in the Friend Offer message and shall be used to derive the friendship security material if a friendship is established as a result of the Friend Offer message. After each Friend Offer message is sent, the FriendCounter value shall be incremented by 1. The FriendCounter may wrap.

The time between receiving the Friend Request message and sending the Friend Offer message is called the Friend Offer Delay and shall be computed based on the RSSIFactor field and the ReceiveWindowFactor field as defined in the Friend Request message on the supported ReceiveWindow and on the RSSI measured by the Friend node for the Friend Request message.

The Friend Offer Delay allows a Low Power node to receive Friend Offer messages from potential Friend nodes in order to determine how large the offered ReceiveWindow is, and how important the signal quality is. Some Low Power nodes will prefer Friend nodes with a very small ReceiveWindow and therefore set the ReceiveWindowFactor to be more important than the RSSIFactor. Other Low Power nodes will prefer Friends with a very good signal strength and therefore set the RSSIFactor to be more important than the ReceiveWindowFactor. This means that the Low Power node should receive Friend Offers from Friends quicker for those nodes that match the Low Power nodes requirements, reducing the power consumed by the Low Power node when searching for a Friend node.

A Local Delay is computed with the formula:

Local Delay = ReceiveWindowFactor * ReceiveWindow - RSSIFactor * RSSI

Where:

ReceiveWindowFactor is a number from the Friend Request message

ReceiveWindow is the value to be sent in the corresponding Friend Offer message

RSSIFactor is a number from the Friend Request message

RSSI is the received signal strength of the received Friend Request message on the Friend node

If the Local Delay value is greater than 100, then the Friend Offer Delay value in milliseconds shall be set to the Local Delay value. Otherwise, the Friend Offer Delay shall be set to 100 milliseconds.

If the node receives a Friend Poll message within 1 second after sending the Friend Offer message, the friendship is established and it shall save the FSN field value from this Friend Poll message; otherwise, the establishment has failed.

The Friend node shall respond with a Friend Update message after a minimum of ReceiveDelay milliseconds and before a maximum of the sum of ReceiveDelay and ReceiveWindow milliseconds, from the reception of the Friend Poll message from the Low Power node.

Figure 3.22 illustrates a friendship establishment where multiple nodes with the Friend feature enabled receive the same Friend Request message.

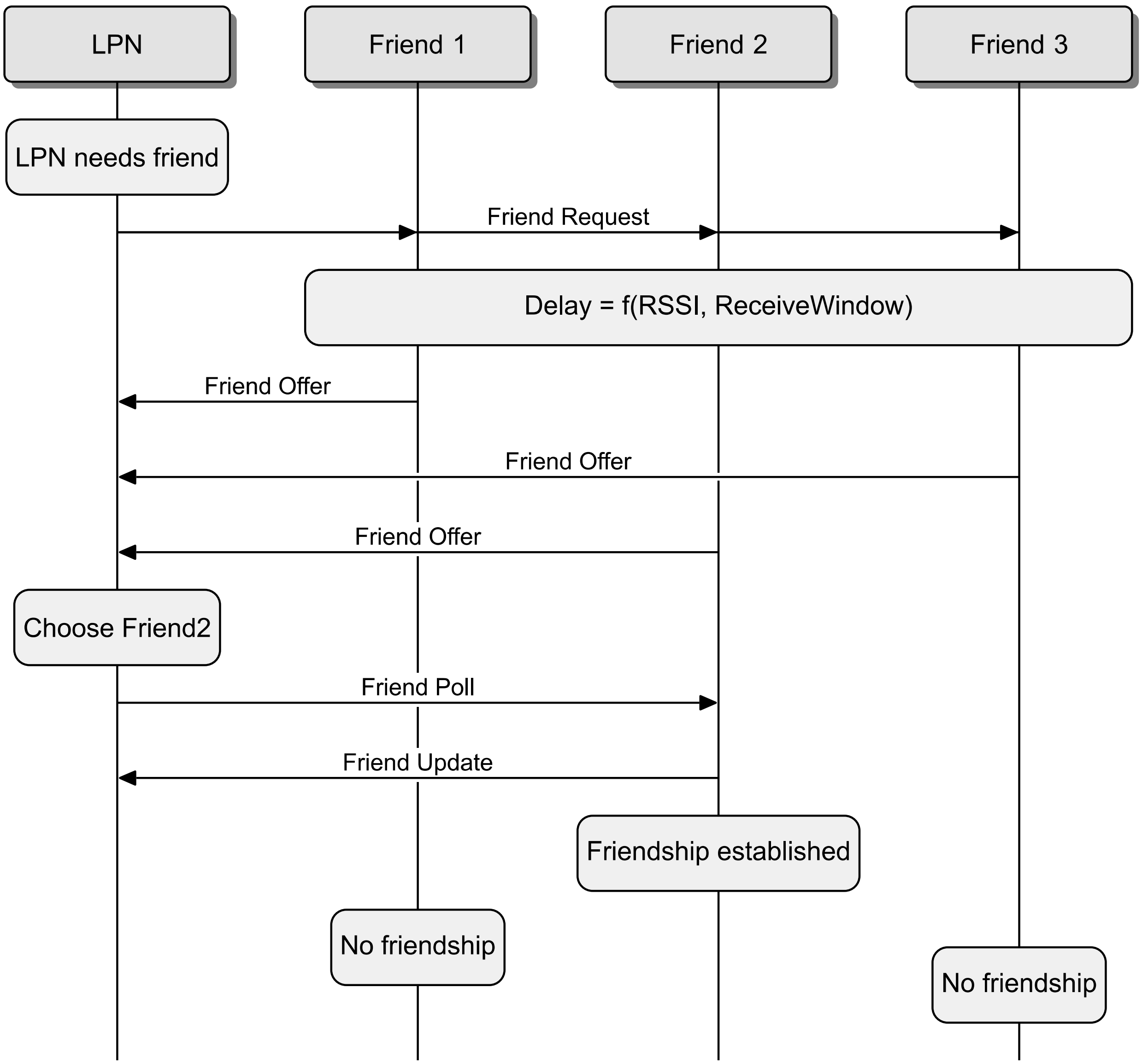


*Figure 3.22: Friend establishment example*

After a friendship is established, the Friend node shall initialize a Friend Subscription List to a zero-length (empty) list and start storing messages for the Low Power node in the Friend Queue.

After a friendship has been established, if the PreviousAddress field of the Friend Request message contains a valid unicast address that is not the Friend node's own unicast address, then the Friend node shall begin sending Friend Clear messages (see Section 3.6.5.5) to that unicast address according to the procedure below:

1. The TTL shall be set to the maximum valid value.
2. The first Friend Clear message shall be sent as soon as the friendship is established; at the same time, a Friend Clear Repeat timer shall be started with the period set to 1 second, and a Friend Clear Procedure timer shall be started with the period equal to two times the Friend Poll Timeout value.
3. If a Friend Clear Confirm message (see Section 3.6.5.6) is received in response to the Friend Clear message, both timers shall be canceled and the procedure is complete.
4. If the Friend Clear Repeat timer expires, a new Friend Clear message shall be sent and the timer shall be restarted with a period that is double the previous Friend Clear Repeat timer period. For example, after the first expiration, the period shall be set to two seconds; on the next expiration, it shall be set to four seconds, and so on.
5. If the Friend Clear Procedure timer expires, then the Friend Clear Repeat timer shall be cancelled and the procedure is complete.

An example of this procedure is illustrated in Figure 3.23.

Once friendship has been established the Friend node shall communicate with the Low Power node as defined in friend messaging (see Section 3.6.6.3.2), and may be managed as defined in Friend Management (see Section 3.6.6.3.3).

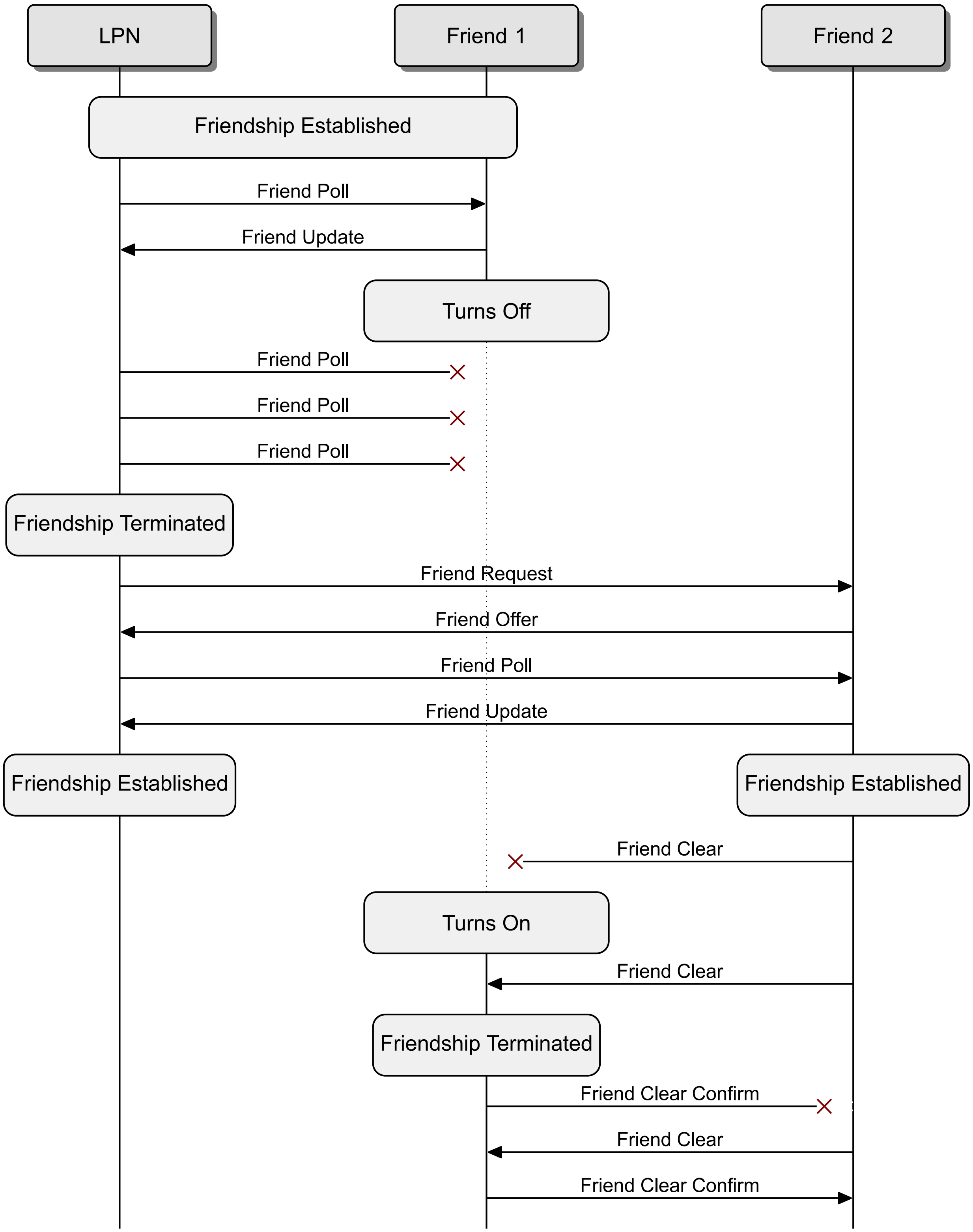
LPN
Friend 1
Friend 2
Friendship Established
Friend Poll
Friend Update
Turns Off
Friend Poll
Friend Poll
Friend Poll
Friendship Terminated
Friend Request
Friend Offer
Friend Poll
Friend Update
Friendship Established
Friendship Established
Friend Clear
Turns On
Friend Clear
Friendship Terminated
Friend Clear Confirm
Friend Clear
Friend Clear Confirm

*Figure 3.23: Friend Clear procedure example*

##### 3.6.6.3.2 Friend messaging

When the Friend node receives a Friend Poll message from a friend Low Power node that has the same FSN field value as the last Friend Poll message received from the Low Power node, the Friend node shall respond with exactly the same message it has previously sent, unless that message has been discarded. If the previously sent message has been discarded, then the oldest entry in the Friend Queue shall be sent.

When the Friend node receives a Friend Poll message from a friend Low Power node that has a different FSN field value as the last Friend Poll message from the Low Power node, then it shall send the oldest message from the Friend Queue.

When sending a stored message to the Low Power node, it shall be sent unchanged. If the IV Index on the Friend node has changed, for example the node is now transmitting with a new IV Index, the messages sent to the Low Power node shall still be sent in the context of the IV Index that the Friend node received for those messages.

Note: The above requirement implies that a Low Power node should collect all stored messages at least once every 96 hours, otherwise the Friend node may discard the stored messages before the Low Power node can receive them.

Each message shall be sent after a minimum of ReceiveDelay milliseconds and before a maximum of the sum of ReceiveDelay and ReceiveWindow milliseconds, from the reception of the Friend Poll message from a friend Low Power node.

If no Friend Poll, Friend Subscription List Add, or Friend Subscription List Remove messages are received by the Friend node before the PollTimeout timer expires, the friendship is terminated and the Friend node shall discard all entries in the Friend Queue.

The Friend Subscription List Confirm message shall be sent after a minimum of ReceiveDelay milliseconds and before a maximum of the sum of ReceiveDelay and ReceiveWindow milliseconds, from the reception of the Friend Subscription List Add message or the Friend Subscription List Remove message.

When a Friend node receives a Friend Clear message where the LPNAddress field is a friend Low Power node, and the LPNCounter field is within the range defined in (Section 3.6.5.6), the friendship is terminated and the Friend node shall discard all entries in the Friend Queue.

##### 3.6.6.3.3 Friend management

If the Friend node receives a Friend Subscription List Add message (see Section 3.6.5.7) from the Low Power node, it shall add the address or list of addresses contained in the message into the Friend Subscription List and respond with a Friend Subscription List Confirm message (see Section 3.6.5.9) setting the value of the TransactionNumber field to the same value as in the received Friend Subscription List Add message.

If the Friend node receives a Friend Subscription List Remove message (see Section 3.6.5.8) from the Low Power node, it shall remove the address or list of addresses contained in the message from the Friend Subscription List and respond with a Friend Subscription List Confirm message setting the value of the TransactionNumber field to the same value as in the received Friend Subscription List Remove message.

The Friend node shall respond with a Friend Update message after a minimum of ReceiveDelay milliseconds and before a maximum of the sum of ReceiveDelay and ReceiveWindow milliseconds, from the reception of the Friend Poll message from the Low Power node.

#### 3.6.6.4 Low Power feature

A node that supports the Low Power feature shall support the three mandatory operations: Low Power establishment, Low Power messaging, and Low Power management.

All transport control messages originated by a Low Power node shall be sent as Unsegmented Control messages with the SRC field set to the unicast address of the primary element of the node that supports the Low Power feature.

##### *3.6.6.4.1 Low Power establishment*

The Low Power establishment operation is used to establish friendship between a node supporting the Low Power feature and a node supporting the Friend feature.

This operation is started by sending a Friend Request message.

The Friend Request message shall be sent with the TTL set to 0 and the DST field set to the all-friends address. The node shall keep a Low Power node counter (LPNCounter), which is a 2-octet value initialized to 0. This value shall be sent in the Friend Request message and used to derive the friendship security material if a friendship is established as a result of the Friend Request message. After each Friend Request message is sent, this value shall be incremented by 1. The LPNCounter may wrap.

After 100 milliseconds have passed from the Friend Request, the node should listen for up to 1 second for the Friend Offer messages sent by potential Friend nodes, and it may select one of the Friend nodes to establish a friendship. The Low Power node may accept a received Friend Offer or continue to listen for other Friend Offer messages for comparison.

If no acceptable Friend Offer message is received, the node may send a new Friend Request message. The time interval between two consecutive Friend Request messages shall be greater than 1.1 seconds.

To establish a friendship with a potential Friend that has sent a Friend Offer message, the node shall set the Friend Sequence Number to zero and send a Friend Poll message to the selected Friend node within 1 second after the reception of the Friend Offer message. If a Friend Update message is received in response, the friendship is established, the Low Power feature of the node supporting it is in use and the Friend feature of the node supporting it is in use.

The node should restart the Low Power Establishment operation if it does not receive a Friend Update message after several attempts (e.g., 6 attempts).

Multiple failures of the Low Power Establishment operation may be an indication that the Low Power node no longer has a valid IV Index and it should initiate the IV Index Recovery procedure (see Section 3.10.6).

Once friendship has been established the Low Power node shall communicate with the Friend node as defined in Low Power messaging (see Section 3.6.6.4.2) and may manage the friendship as defined in Low Power management (see Section 3.6.6.4.3).

##### *3.6.6.4.2 Low Power messaging*

The Low Power messaging operation is executed by a Low Power node to receive stored messages and security updates from the Friend node.

The operation consists of asynchronous requests from the Low Power node to the Friend node and timed responses from the Friend node to the Low Power node.

A Low Power node that is friends with a Friend node shall send a Friend Poll message to the Friend node before the PollTimeout timer expires.

In a Friend Poll message, the TTL field shall be set to 0.

As a general rule, the Low Power node should continue sending Friend Poll messages until it receives a Friend Update message with the MD field set to 0. This is illustrated in Figure 3.24.

The Low Power node may terminate friendship with a Friend by sending a Friend Clear. The Friend Clear message should be sent using a TTL of 0.

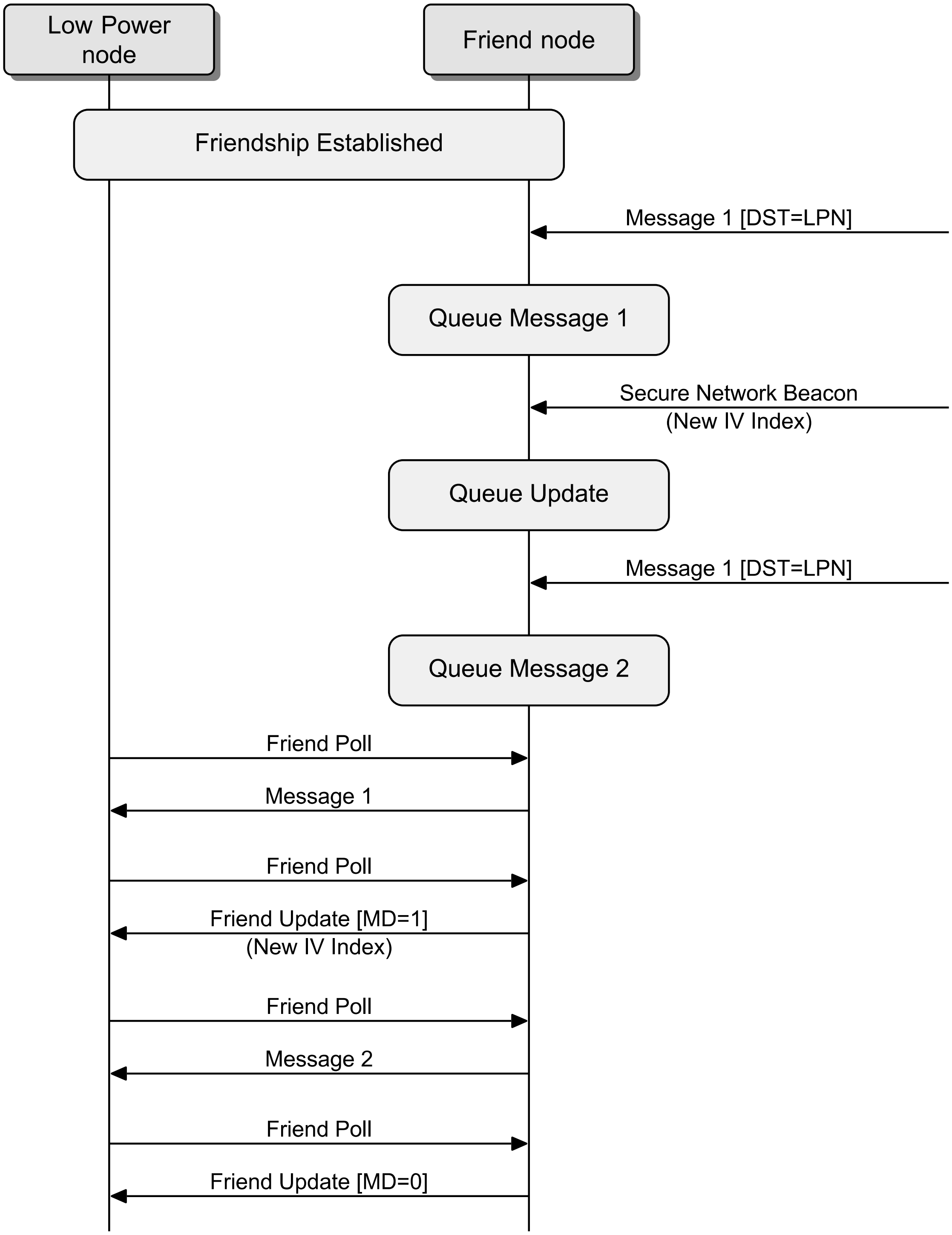


*Figure 3.24: Friend Update with security updates*

The FSN field shall be set to the value of the Friend Sequence Number.

If the Low Power node receives a response from the Friend node, then it shall toggle the Friend Sequence Number.

If the Low Power node does not receive a response within the ReceiveWindow, it should resend the Friend Poll message. It is recommended to resend this message 3 times, which assures a good balance between reliability and power consumption.

If no response has been received to multiple Friend Poll messages before the PollTimeout timer expires, the friendship is terminated. The Low Power node may repeat the Low Power establishment operation.

If the Low Power node receives a Friend Update message, it shall process the Flags and IV Index fields using the same rules as if they had been received in a Secure Network beacon (see Sections 3.9.3.1, 3.10.4, and 3.10.5).

##### *3.6.6.4.3 Low Power management*

The Low Power management operation is used to manage the subscription list in a Friend node.

The Low Power node may send one or more Friend Subscription List Add messages to the Friend node containing lists of group addresses or virtual addresses to which the Low Power node is subscribed. This type of message may be sent at any time by the Low Power node when its subscriptions change.

The Low Power node may send one or more Friend Subscription List Remove messages to the Friend node containing lists of group addresses or virtual addresses to which the Low Power node is no longer subscribed. This type of message may be sent at any time by the Low Power node when its subscriptions change.

The Low Power node shall start with a TransactionNumber value set to 0x00. It shall increment the TransactionNumber for each new Friend Subscription List Add or Friend Subscription List Remove such that the TransactionNumber can be matched with the TransactionNumber field of the Friend Subscription List Confirm message.

#### 3.6.6.5 **Examples of segmentation and reassembly**

The segmentation and reassembly behaviors defined in Sections 3.5.3.3 and 3.5.3.4 also apply to segmented messages sent to and from a Low Power node. The only difference is that since the Low Power node relies on the Friend Queue for all incoming messages, including segments and segment acknowledgments, the Friend node will acknowledge segmented transactions for the Low Power node.

This section provides two examples of segmentation and reassembly on the Low Power node.

##### *3.6.6.5.1 Incoming segmented message*

The message sequence chart (MSC) in Figure 3.25 is an example of a segmented message directed toward a Low Power node. The Friend node performs the reassembly separately and sends the acknowledgments needed until it receives all segments, at which point the Friend node places the segments in the Friend Queue so that they can be delivered to the Low Power node. An unsegmented message from another source is received in the middle of the transaction and is handled independently.

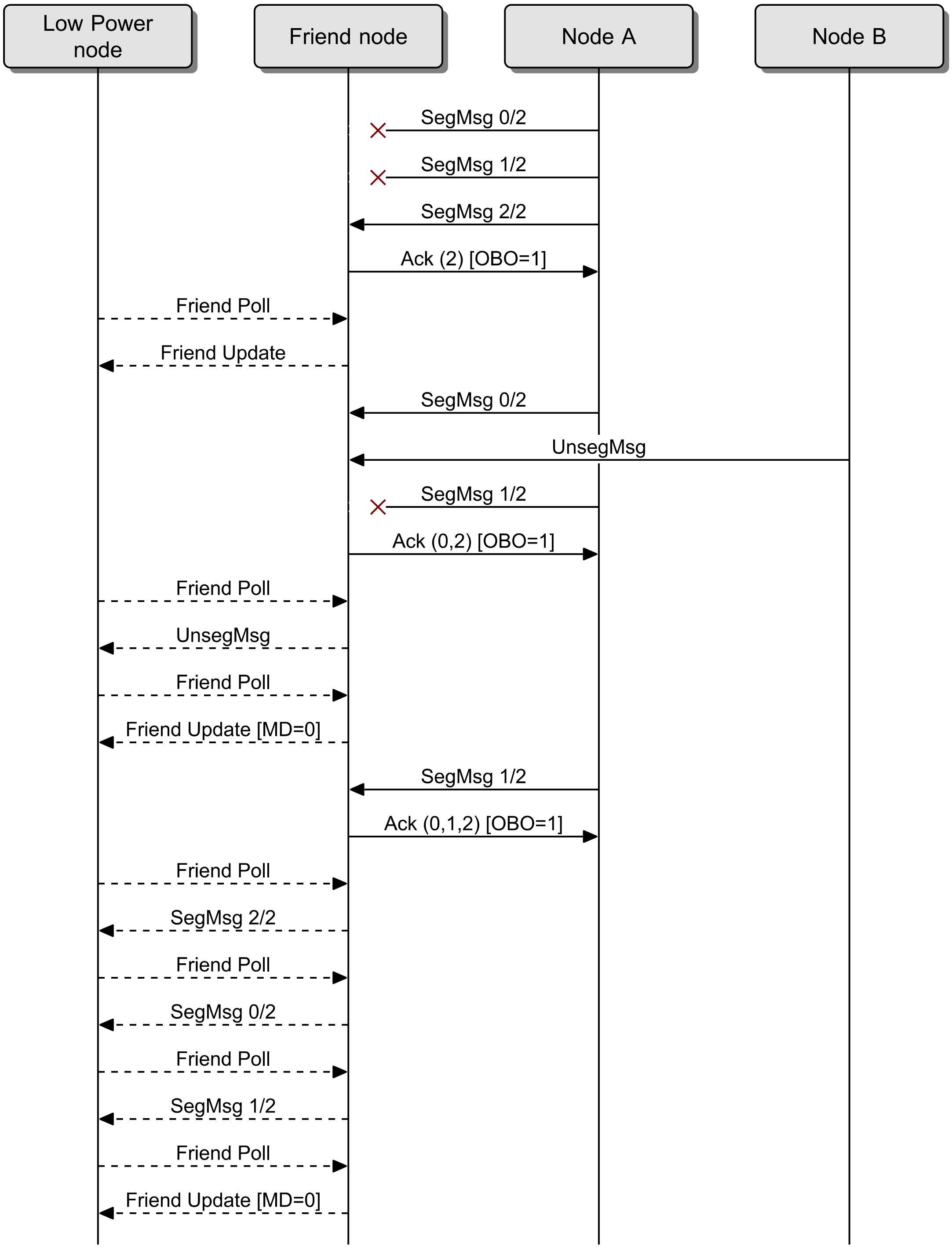


*Figure 3.25: Example of incoming segmented message directed to a Low Power node*

### 3.6.6.5.2 Outgoing segmented message

The MSC illustrated in Figure 3.26 shows an example of a segmented message sent by the Low Power node. Since the Low Power node relies on the Friend Queue for all incoming messages, it needs to poll for the acknowledgment as well. An unsegmented message from another source is received in the middle of the transaction and is handled independently.

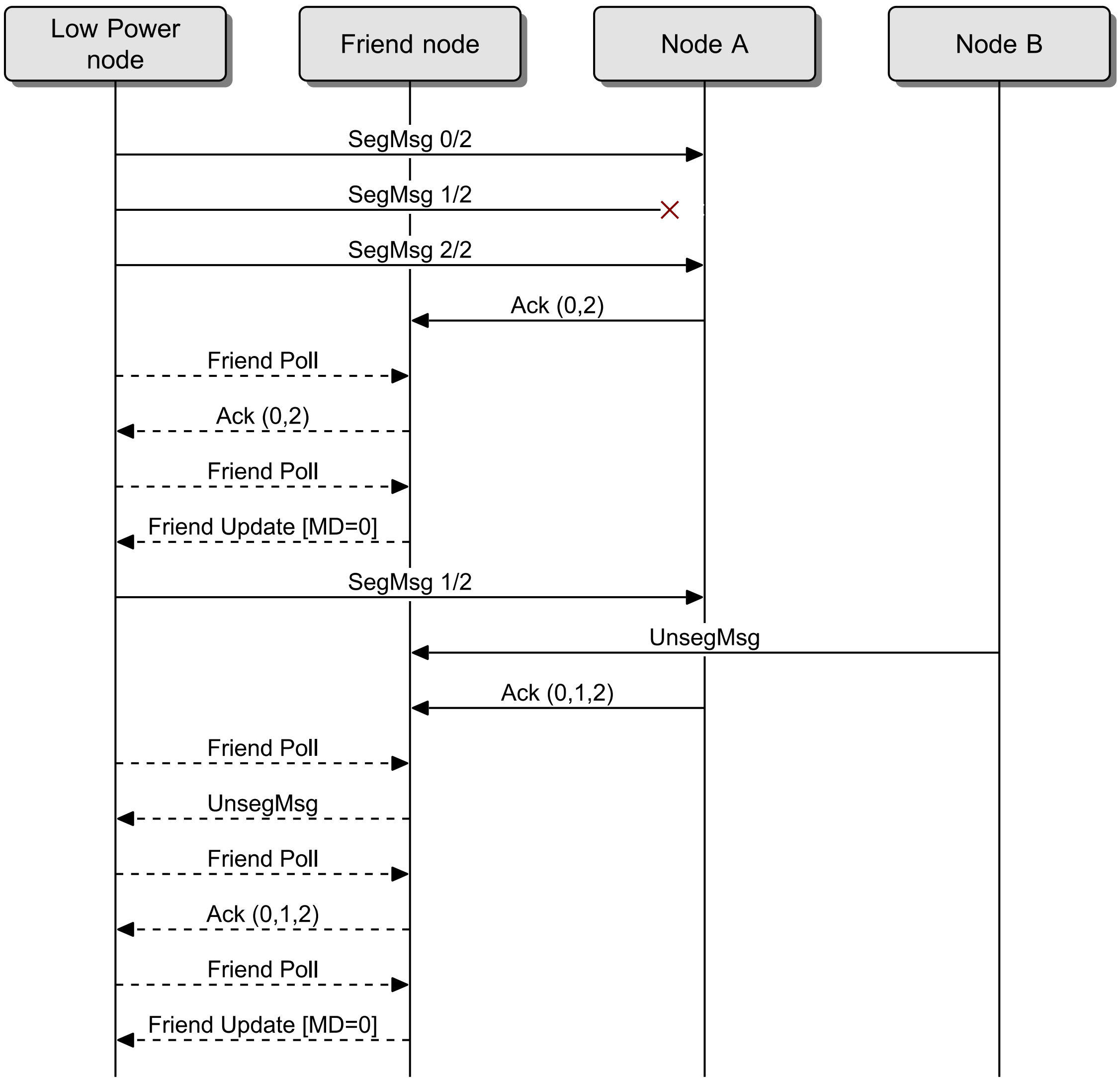


*Figure 3.26: Example of outgoing segmented message sent by a Low Power node*

## 3.6.7 Heartbeat

Heartbeat is used to monitor nodes on a network and discover how far nodes are apart from each other.

### 3.6.7.1 Functional overview

In order to determine if a node is still present and active within a mesh network, it is necessary to receive a message from this node. Sending a message to each and every node in the mesh network to elicit a response would be very wasteful of energy, and therefore each node can be configured to send a single message periodically. This message is called the Heartbeat message.

The Heartbeat message can be used for two main functions. The first function is the determination that a node is still active within a mesh network. The second function is the determination of how far a node is away.

The Heartbeat message is sent periodically, as configured by the Configuration Server model. The Heartbeat message can be sent a limited number of times or they can be sent indefinitely. Heartbeat messages are sent to a configured destination, and it is recommended that a group address is used for sending Heartbeat messages. The messages can also be configured with a specific TTL value.

When Heartbeat messages are received, they are counted. The number of Heartbeat messages received can help determine the reliability of the mesh network for delivering messages from the node sending Heartbeat messages.

Each Heartbeat message includes the initial TTL value used when sending the original Heartbeat message. This allows a receiving device to determine the number of times this message was retransmitted, known as the number of hops, and a record of the minimum and maximum number of hops can also be used to determine how reliable the mesh network is.

The use of Heartbeat messages can therefore be used to determine the best TTL value to use to address a given node.

Heartbeat messages also include the features of the node that are currently in use. A node can be configured to send a Heartbeat message when various features are enabled or disabled. This allows the features available on various nodes within a mesh network to be determined.

### 3.6.7.2 Publishing Heartbeat messages

Publishing of Heartbeat messages is controlled by the Heartbeat Publication state (see Section 4.2.17).

When the Heartbeat Publication Destination (see Section 4.2.17.1) address is set to an unassigned address, or when the value of the Heartbeat Publication Count state (see Section 4.2.17.2) is 0x0000, Heartbeat messages shall not be published.

Heartbeat messages shall be published with the DST field set to the value of the Heartbeat Publication Destination state and with the TTL field set to the value of the Heartbeat Publication TTL state (see Section 4.2.17.4).

Periodic publishing of Heartbeat messages is enabled by the Heartbeat Publication Count state (see Section 4.2.17.2). After publishing a Heartbeat message, if the Heartbeat Publication Count counter is less than 0xFFFF, the Heartbeat Publication Count counter shall be decreased by 1. The counter shall stop at 0x0000. The first Heartbeat message shall be published as soon as possible after the Heartbeat Publication Period state (see Section 4.2.17.3) has been configured for periodic publishing. The next Heartbeat message shall be published after the number of seconds defined by the Heartbeat Publication Period state (see Section 4.2.17.3).

Triggered publishing of Heartbeat messages is enabled by the Heartbeat Publication Features state (see Section 4.2.17.5):

- If the Relay bit is set to 1, a Heartbeat message shall be published when the Relay state of a node (see Section 4.2.8) changes.
- If the Proxy bit is set to 1, a Heartbeat message shall be published when the GATT Proxy state of a node (see Section 4.2.11) changes.
- If the Friend bit is set to 1, a Heartbeat message shall be published when the Friend state of a node (see Section 4.2.13) changes.
- If the Low Power bit is set to 1, a Heartbeat message shall be published when the node establishes or loses Friendship (see Section 3.6.6.1).
- If the Directed Forwarding bit is set to 1, a Heartbeat message shall be published when the Directed Forwarding state of a node (see Section 4.2.22) changes in the same subnet over which the message is sent.

#### 3.6.7.3 Receiving Heartbeat messages

Receiving of Heartbeat messages is controlled by the Heartbeat Subscription state (see Section 4.2.18).

The Heartbeat Subscription Period state is a countdown timer identifying the number of seconds remaining for the period when Heartbeat messages are received. When the timer reaches a value of 0x0000, the receiving of Heartbeat messages shall be disabled.

Heartbeat messages with the SRC field set to a value other than the Heartbeat Subscription Source state (see Section 4.2.18.1) or the DST field set to a value other than the Heartbeat Subscription Destination state (see Section 4.2.18.2) shall not be processed.

Upon receiving a Heartbeat message, the value of the Heartbeat Subscription Count state (see Section 4.2.18.3) shall be increased. The counter does not wrap. It stops counting at 0xFFFF.

Upon receiving the Heartbeat message, a hops value shall be calculated using the InitTTL value from the message, and the received Network PDU TTL field value, known as RxTTL, as follows:

hops = InitTTL - RxTTL +1

Note: If the message is received directly (for example, the InitTTL value and the received Network PDU TTL field value are the same), then the hops value would be 0x01. If the message has been delivered using the maximum length path, then InitTTL would be 0x7F and the received Network PDU TTL field value would be 0x01, and therefore hops would 0x7F.

If the hops value is lower than the Heartbeat Subscription Min Hops state, it shall be set as the new value of the Heartbeat Subscription Min Hops state. If the hops value is higher than the Heartbeat Subscription Max Hops state, it shall be set as the new value of the Heartbeat Subscription Max Hops state.

### 3.6.8 Directed Forwarding

#### 3.6.8.1 Functional overview of Directed Forwarding

The Directed Forwarding feature is based on a hybrid reactive and proactive protocol that is followed to perform discovery and maintenance of paths.

A Directed Forwarding Discovery operation (see Section **錯誤! 找不到參照來源。**) aims to find a valid path between two Directed Forwarding nodes, the node that originates messages (the Data Originator) and the node that is the destination of such messages (the Data Destination). This reactive procedure is initiated on demand by the Data Originator (see Section **錯誤! 找不到參照來源。**). The Data Destination either can be one or more elements of a single node or it can be a group of nodes that are subscribed to the same group address or virtual address. When a Data Originator needs to communicate with a target Data Destination, and does not have knowledge of a path to the latter, it broadcasts a Path Request (PREQ) message to find valid paths. In this case, the Data Originator performs as a Path Originator. The PREQ message are re-broadcast by Directed Relay nodes. A path is found when the PREQ message reaches the target Data Destination, which is then referred to as the Path Destination.

The path is made available by unicasting, hop by hop, a Path Reply (PREP) back to the Path Originator. This operation is named Directed Forwarding Establishment (see Section **錯誤! 找不到參照來源。錯誤! 找不到參照來源。**). Each Directed Forwarding node that receives the PREQ message stores information of a path back to the Path Originator (the Backward Path), so that the PREP message can be unicast along the Backward Path to the Path Originator from the Path Destination. In addition, each Directed Forwarding node that receives the PREP message stores information of a path toward the Path Destination (the Forward Path), which is used later by the Path Originator and the intermediate Directed Relay nodes along the Forward Path to forward messages addressed to the Path Destination.

A PREP may trigger the Path Originator to send a Path Confirmation (PCONF) message to confirm that a two-way path has been established from the Path Originator to the Path Destination.

Directed Forwarding nodes may use a Path Tracing Request (PTREQ) message to trace and validate a path from a Path Originator to a Path Destination. The message is processed similarly to a PREQ message, but it is propagated only along an existing path. The response is a Path Tracing Reply (PTREP) message sent back from the Path Destination to the Path Originator.

The Neighbor Information (NINFO) message, is used by Directed Forwarding nodes to exchange information proactively with their neighbors and to avoid the need for Directed Forwarding Discovery for local communication (see Section 錯誤! 找不到參照來源。).

Directed Forwarding nodes may monitor the status of valid paths (that is, paths whose information is stored), for example, by monitoring the status of links or NINFO messages from neighboring nodes. When a node finds that a path is invalid, the node may add a node to the path to assist message forwarding by performing a Directed Forwarding Assistance operation, with exchange of Path Assistant Request (PAREQ) and Path Assistant Reply (PAREP) messages (see Section 錯誤! 找不到參照來源。). If the Directed Forwarding Assistance operation is not initiated or if fails, the node sends out a Path Error (PERR) message to notify other Directed Forwarding nodes that the associated Path Destination is no longer reachable (the Directed Forwarding Error Notification operation) (see Section 錯誤! 找不到參照來源。).

In Directed Forwarding, the overall effect of combining reactive (for example, the Directed Forwarding Initialization operation) and proactive (for example, the Neighbor Information Management operation) approaches is that a path is followed from the Path Originator to the neighborhood of the Path Destination. All nodes within the neighborhood of the destination potentially participate in the message forwarding.

The Directed Forwarding feature uses a control sequence number that is defined as Forwarding Number (see Section 錯誤! 找不到參照來源。), which is separate from the sequence numbers used for the message caching and the replay protection procedures (that is, the SEQ field defined in Section 3.4.4.5). The Forwarding Number is used for the prevention of path loops, as discussed in Section 錯誤! 找不到參照來源。, and for the establishment of paths toward group addresses, as described in Section 錯誤! 找不到參照來源。.

#### *3.6.8.1.1 Path metrics*

Directed Forwarding provides the way to configure a metric (for example, a distance, cost, or quality metric) that can be used to measure, rank, and select the best paths (for example, the shortest paths, the paths with the lowest cost, or the best-quality paths) that messages will traverse.

A path is composed of nodes and the links between them. A good metric allows the selection of the most stable paths for the following purposes:

- To avoid frequent discovery operations (that is, creation and use of different paths) because of paths that soon become invalid or congested.
- To improve network performance, mainly in terms of latency and use of bandwidth.

Hop-wise path length is a simple and widely used path metric, which counts the number of hops between the Path Originator and the Path Destination. However, hop-wise path length has the following shortcomings:

- It does not differentiate the quality of multiple paths that are the same length, which is a common occurrence in dense networks.

- It does not take into account the path stability (or longevity); only its current availability.

To overcome the shortcomings, a Path Metric (PM) in Bluetooth Mesh can be based on a Node Metric (NM) and a Link Metric (LM) (see Section **錯誤! 找不到參照來源。**), and can be calculated according to the composition of the path.

Figure 3.27 illustrates a path (p) composed of L links ($l_i$) between pairs of nodes ($n_i$ and $n_{i+1}$), with the index "i" in the range of (0, L-1). NM($n_i$) and LM($l_i$) are the metrics associated with the $i^{th}$ node and the $i^{th}$ link, respectively.

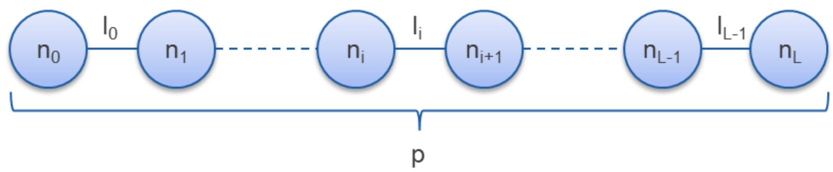


*Figure 3.27: Example of a path composed of L links*

The metric associated with the path is:

$$PM(p) = sum_{i=1...L-1}(NM(n_i)) + sum_{i=0...L-1}(LM(l_i))$$

### *3.6.8.1.2 Network interfaces and bearers*

The network layer supports sending and receiving messages via multiple bearers, and each instance of a bearer is connected to the network layer via a network interface. With the possibility that different bearers provide different link costs for the evaluation of the path metric, the Forwarding Table entries are associated with the bearers over which the messages are transmitted (see Section 3.6.8.2).

PREQ messages may be sent over multiple bearers. The Discovery Table contains an indicator of the selected bearers over which the PREP is expected to be sent (see Section 3.6.8.4.2). The Forwarding Table entries are only considered valid in the context of those bearers. Therefore, when transmitting a message by using the directed security credentials, the message is sent over a set of the valid bearers.

### *3.6.8.1.3 Group destinations*

Directed Forwarding supports multicasting by providing the ability to create paths toward group addresses.

During the configuration of the mesh node, models of elements within the node can subscribe to one or more group addresses. A message sent to a group address is processed by all the subscribed nodes. In order to use the Directed Forwarding feature for group-casting, all subscribed nodes are expected to be Directed Forwarding nodes or Low Power nodes in a friendship with a Directed Forwarding node. The support for the Directed Forwarding feature is published via Heartbeat messages (see Section 3.6.7).

Multicasting for Directed Forwarding nodes is enabled as follows. When a Data Originator communicates with a group (Group Destination) and does not have knowledge of a path to the latter (see Section 錯誤! 找不到參照來源。), it broadcasts a PREQ with the corresponding group address as Path Destination. Each of the (elements of) nodes subscribed to the group address issues a PREP with its unicast address as a Path Destination. Intermediate Directed Relay nodes can uniquely associate the received PREP with the forwarded PREQ by looking for matching entries in the Discovery Table (see Section 錯誤! 找不到參照來源。). By means of the Directed Forwarding Discovery operation, the Path Originator receives a PREP for the primary element of each node that is subscribed to the group address.

If the number of nodes that are subscribed to a group address is known at the Path Originator, a path toward a group is successfully established if the expected number of responses is reached. Otherwise, the path is considered established if at least one PREP is received (see Section 3.6.8.2).

In any case, the number of PREP messages received can be used at the Path Originator to estimate the number of subscribed nodes, which can be used by, for example, the access layer.

In summary, by doing multicasting using Directed Forwarding nodes:

- The originator gets an estimate of the number and identity of subscribed nodes.
- Individual paths for each of the destinations are stored in the Forwarding Table without additional overhead.
- Acknowledgments may be sent by each destination of a group message along backward paths.

##### 3.6.8.1.4 Virtual destinations

A virtual address is a hash value derived from a Label UUID and logically represents the Label UUID. An access message directed to a Label UUID is addressed to the corresponding virtual address, whereas the Label UUID is used by the publisher as the Additional Data field of the message integrity check value in the upper transport layer. When an access message addressed to a virtual address is received by a node that is subscribed to the Label UUID from which the virtual address was derived, the node can authenticate the message at the upper transport layer and process it.

For the Directed Forwarding feature, a virtual address is treated as a group address, following the procedure described in Section 錯誤! 找不到參照來源。. The Transport Control messages, including those defined for Directed Forwarding, are authenticated only at the network layer, and do not require any Additional Data field for the message integrity check. Consequently, if a PREQ is addressed to a virtual address, every Directed Forwarding node receiving and authenticating the PREQ message at the network layer, and subscribed to a Label UUID that the virtual address can be derived from, is considered a valid PREQ destination and can reply to the PREQ with a PREP, even if the Label UUID of the found match does not correspond to the Label UUID targeted by the node that has sent the PREQ.

There is a potential risk of establishing paths toward nodes that are subscribed to a different Label UUID that is associated with the same virtual address. In this case, some messages may be forwarded also toward destinations that are not the intended destinations.

#### 3.6.8.1.5 Bi-directional forwarding

To support forwarding of bi-directional traffic on a mesh—for example, the return of a message in response to an acknowledged message—a Directed Forwarding node can forward packets from the path originator to the path destination and vice versa by using a single Forwarding Table entry.. This minimizes additional traffic overhead and avoids the need for additional Forwarding Table memory to establish the return path for a forwarding path. After a path is established, that path can be used immediately by the originator. However, before using the path in the backward direction from the destination, the destination validates the path. For example, the destination may mark a path as validated if it receives a message on that path using the directed security credentials or receives a PCONF message from the originator in response to a PREP message (as defined in Section 3.6.8.2). The same validation is performed also for a multicasting scenario where individual unicast paths toward the group subscribers are established (see Section **錯誤! 找不到參照來源。**).

#### 3.6.8.1.6 Friendship and Directed Forwarding

To enable Directed Forwarding operations for a Low Power node that does not support Directed Forwarding, the Low Power node establishes friendship with a Friend node that supports the Directed Forwarding feature and has the feature enabled. Then the Low Power node uses the friendship security credentials for outgoing messages.

Friend nodes that support the Directed Forwarding feature provide discovery, management, and message forwarding for the associated Low Power nodes in the following ways:

- A Friend node acts on behalf of a Low Power node to generate and process Directed Forwarding control messages and to forward outgoing data messages using the directed security credentials.
- When the Friend node receives an outgoing message from the Low Power node using the friendship security credentials, if a valid path to the destination already exists, the message is delivered using the directed security credentials.
- If a valid path to the destination does not exist yet, the outgoing message may be delivered using the master security credentials, and the Friend node initiates a Directed Forwarding Discovery operation as a Delegate Path Originator (see Section **錯誤! 找不到參照來源。**).
- The Friend node also processes PREQ, PERR, and NINFO messages from other nodes for the Low Power node and generates PREP messages as a Delegate Path Destination in response to PREQ messages.

Heartbeat messages published by a Friend node in a subnet can be used to inform a Low Power node on whether the Friend node supports Directed Forwarding in that subnet..

#### 3.6.8.1.7 Fixed paths

A Directed Forwarding node stores information about paths in its Forwarding Table state, which it consults to decide whether or not message forwarding is allowed. As per default behavior, paths are established and their information updated by the node itself while executing any of the described algorithms for discovery and maintenance of paths (see Section 錯誤! 找不到參照來源。). In addition, Provisioner can read and configure the Forwarding Table state of a node, thus providing useful capabilities, such as checking the correctness or the quality of established paths or installing preferred path entries to have fine control over message traffic throughout the network.

In the Forwarding Table state, a distinction is made between fixed path entries and non-fixed path entries. A fixed path entry can be installed only by a Provisioner and, conversely, a Provisioner can install only fixed path entries. Fixed path entries never expire, and they can be removed only by a Provisioner. A Provisioner can query or remove any of fixed and non-fixed path entries.

Non-fixed path entries are  the ones installed with Directed Forwarding operations. A non-fixed path entry extends the set of significant information reported in a fixed path entry in support of Directed Forwarding operations.

#### *3.6.8.1.8 Loop-free forwarding*

A path discovery algorithm should avoid the formation of both permanent and temporary forwarding loops. A *forwarding loop* occurs when a message that is directed over a network traverses the same node more than once, before reaching or without reaching the target destination. This is mainly caused by inconsistent updates of Forwarding Table states on the nodes, leading to the selection of next-hop nodes that have already been visited. Consequently, nodes should simply ignore announcements of path updates (that is, the latest changes to path information) that lead to forwarding loops.

In Directed Forwarding, a received path update is accepted and incorporated in a Forwarding Table state if any of the following conditions are met (see Section 錯誤! 找不到參照來源。):

- The Path Destination is unknown (that is, its address has not yet been inserted in Forwarding Table).
- The Path Destination is known, and the received Forwarding Node (FN) value generated by the Path Destination is greater than a known FN (that is, an FN previously stored and generated by the same Path Destination).
- The Path Destination and FN both are known, and the received Path Metric (PM) value is associated with a better path to the Path Destination (see Sections 錯誤! 找不到參照來源。 and 錯誤! 找不到參照來源。).

In a static network with no topology changes, the above requirements will help to minimize the potential for forwarding loops. Nonetheless, multiple concurrent discovery and maintenance algorithms running over the network might create temporary inconsistencies between the Forwarding Table states of different nodes. This is particularly true because of the hybrid nature of Directed Forwarding, which combines an on-demand approach to discovery of paths toward farther destinations, based on the exchange of PREQ and PREP messages, with a proactive announcement of paths toward nearer destinations, carried out with NINFO messages.

The first approach allows finding a path in a reasonably short time, conditioned only by the latency of the delivery of the PREQ and PREP messages in the forward and backward directions, respectively.

The second approach usually is slower because it depends on the schedule of the NINFO transmissions. If NINFO transmission is periodic, with a period T, a Directed Relay node sends out an extract of its Forwarding Table state at every T interval. In the first step, it announces only its presence; in the second step, it also announces the information about the reachability of its one-hop neighbors. In response to the first step, each one-hop neighbor announces its presence, which provides information on their reachability. By extension, in the $N^{th}$ step, th node will be able to announce the information of reachability of its (N-1)-hop neighbors. In this approach, a path to a destination N hops away is found after an (N*T) interval.

This interval calculation represents the shortest possible interval because it assumes that during an interval T, a node receives a NINFO from each of its neighbors, which is not generally true. For the same reason, Forwarding Table states of different nodes get updated differently at different times, so the proactive scheme may suffer from temporary forwarding loops. To help reduce this problem or to help make its side effects more acceptable, Directed Forwarding requires that NINFO messages be used to report path updates for destinations up to a configurable number of hops from the message sender (known as the “radius of the neighborhood of the node”).

If the network topology changes (for example, because of node mobility, energy depletion in battery-powered nodes, broken nodes, or the presence of obstacles), countermeasures need to be taken to react promptly and avoid Forwarding Table inconsistencies and forwarding loops as much as possible.

In Directed Forwarding, the following options are available:

- Every path entry in a Forwarding Table is used for a limited time, as indicated by the Path Lifetime state (see Section 4.2.23.2). When this time elapses, the entry is considered invalid and is removed.
- NINFO messages are broadcast periodically and report only path updates..
- Every node collects a number of path updates for the same destination during a “settling time” before sending out its own path updates..

Because of the absence of any explicit next-hop indication, the presence of a loop in a path does not prevent the propagation of a message from source to destination.

##### 3.6.8.1.9 Path resilience

When many nodes contend for the use of a common channel for their transmissions, using Directed Forwarding nodes allows for a reduction of interference on the managed-flood mesh network.  Directed Forwarding involves only the nodes along the path from the Data Originator to the Data Destination in forwarding messages between any two nodes, and the sequence of transmissions among such nodes is ordered. On the other hand, the reliability of Directed Forwarding depends on the validity of the established paths.

The benefits of Directed Forwarding can be achieved at the cost of injecting in the network additional control traffic to check whether the discovered paths are valid.. A path is assumed to be valid at the time of its creation, but it could be difficult to estimate its stability. Several factors, such as node mobility, energy depletion, or the presence of obstacles, may change the network topology and break a path, often in an unpredictable way. Such an event usually triggers finding an alternative path, but this takes time, and the time required varies depending on the strategy that is used to detect the path failure and notify the node to start discovering a new path. The efficiency of such a strategy seriously affects the path resilience against the occurrence of path breakages, and is to be balanced with the detriment of the large number of control messages that are generated under this strategy.

Directed Forwarding provides a fair strategy that combines prediction of path longevity, monitoring (or tracing) of established paths, and notification of path failures.

During the Directed Forwarding Discovery operation, every path is evaluated according to a given Path Metric, which is lower if the path is considered “shorter” in the sense of the used metric and is associated with a lower probability of being broken. If multiple paths are found toward a Path Destination, they are measured with such a metric and then ranked, and the path that corresponds to the lowest value of the metric is selected. The Path Lifetime  is defined as the time during which the path is considered active and usable; after that time, the path is considered expired, and a new discovery is required.

After a path is discovered and before it expires, the Path Originator verifies that the path has not been interrupted by a topology change and is still viable. An interruption may happen for many reasons and is unpredictable. Consequently, the lifetime of a specific path can only be estimated; it cannot be guaranteed and it needs to be verified periodically. To confirm the existence of a path, a Directed Forwarding Tracing operation may be initiated by the Path Originator (see Section **錯誤! 找不到參照來源。**). This operation is very similar to a Directed Forwarding Discovery operation (Section **錯誤! 找不到參照來源。**), but it uses a different control message and the message is not forwarded by all nodes; instead, the message flows only along the selected path until it reaches the Path Destination. The Path Destination will reply (that is, will echo) to the Path Originator, thus confirming the existence of the path.

### *3.6.8.1.10 Coexistence of multiple subnets*

If a Directed Forwarding node belongs to multiple subnets—that is, it can receive and process messages from different directed security domains—then each subnet is treated separately. The node creates and maintains different path entries for different subnets. When messages are directed to a Path Destination in a subnet, the directed security material derived from the network key of the subnet is used.

As a consequence, the node maintains different tables to store information about paths discovered (see Section 3.6.8.4.2) and established (see Section 3.6.8.2) in different subnets. The states associated with the Directed Forwarding feature (see Sections 4.2.22, **錯誤! 找不到參照來源。**, **錯誤! 找不到參照來源。**, 4.2.25 and **錯誤! 找不到參照來源。**) can also be set to different values for different subnets. For example, a node that supports the Directed Forwarding feature can have the feature enabled in one subnet and disabled in another subnet. However, the Neighbor Table used to store information about links with neighbors (see Section 3.6.8.5) is common to all the subnets to which the node belongs.

To reduce the potential risk of anomalies while executing Directed Forwarding operations in the subnet (see Section 3.6.8.1), all Directed Forwarding nodes belonging to a given subnet should support and use the same value of the Path Metric state (see Section 錯誤! 找不到參照來源。).

#### 3.6.8.2 Directed Forwarding operations

This section defines the requirements for Directed Forwarding operations. The Directed Forwarding feature defines six mandatory operations: Directed Forwarding Initialization, Directed Forwarding Discovery, Directed Forwarding Establishment, Directed Forwarding Confirmation, Directed Forwarding Error Notification, Neighbor Information Management. It also defines three optional operations: Directed Forwarding Tracing, Directed Forwarding Assistance, and Link Quality Measurement.

##### *3.6.8.2.1 Directed Forwarding Initialization*

Directed Forwarding Initialization shall be executed by a Path Originator or a Delegate Path Originator to establish a new path or to replace an existing path toward a Path Destination. The address of the Path Destination shall be a unicast address, a dynamically assigned group address, or a virtual address.

If the number of entries in the Discovery Table exceeds the value of the Concurrent Initialization state in the Path Request Transmit composite state (see Section 錯誤! 找不到參照來源。), or if the time from the last Directed Forwarding Initialization operation for the same Path Destination is shorter than the value indicated in the Path Request Interval state in the Path Request Transmit composite state (see Section 錯誤! 找不到參照來源。), then the Directed Forwarding Initialization operation shall not be executed.

錯誤! 找不到參照來源。 shows the flow chart of the Directed Forwarding Initialization operation.

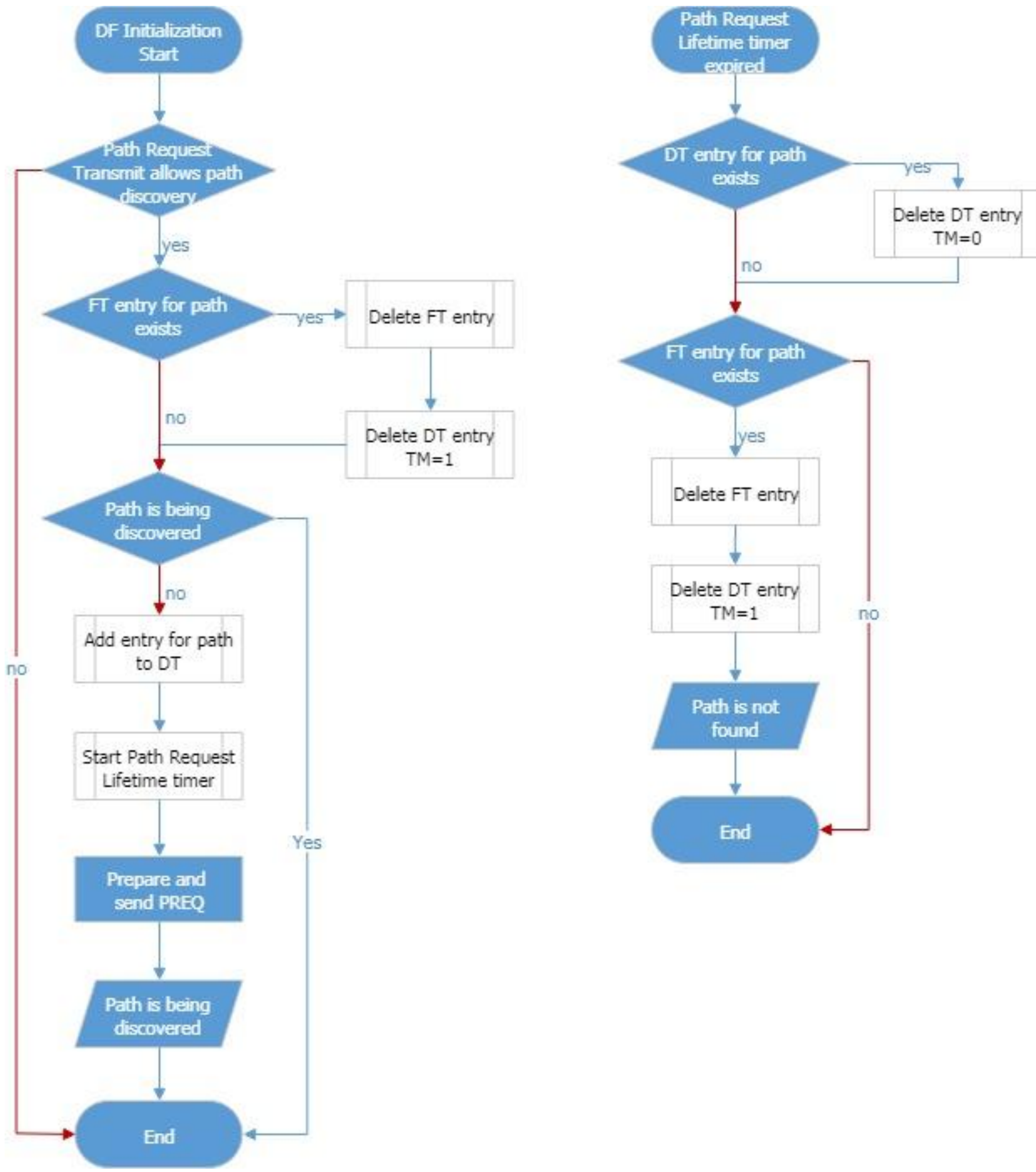


*Figure 3.28: Directed Forwarding Initialization operation*

First, the Path Originator or the Delegate Path Originator checks whether or not an entry associated with the same path exists in the Forwarding Table state (see Section 3.6.8.2). If an entry exists and the BackwardPathValidated flag in the entry is set to 0 (that is, the entry is associated with a one-way path), the entry shall be deleted. If an entry exists and the BackwardPathValidated flag in the entry is set to 1 (that is, the entry is associated with a two-way path), the BackwardPathValidated flag shall be set to 0 (that is, the entry is invalidated in the backward direction).

The Forwarding Table state may contain both a one-way path entry from the Path Originator to the Path Destination and a corresponding two-way path entry from the Path Destination to the Path Originator. For example, the Path Originator and the Path Destinationmight have established paths toward each other at different times, and the Path Originator confirmed the path originated by the Path Destination (see Sections **錯誤!** 找不到參照來源。, **錯誤!** 找不到參照來源。, and **錯誤!** 找不到參照來源。). If both entries exist, the one-way path entry is deleted and the two-way path entry is invalidated in the backward direction.

If the Path Destination is a group address or a virtual address, the Path Originator or the Delegate Path Originator checks whether or not at least one entry with the Path Destination in its Subscription List exists in the Forwarding Table. If so, the corresponding group address or virtual address shall be removed from the Subscription List of all the entries. Then, the Path Originator or the Delegate Path Originator checks whether or not the path is being discovered, as defined in Section 3.6.8.4.2. If so, no further operation is performed.

If the path is not being discovered, a new entry associated with the path shall be added to the Discovery Table, as defined in Section **錯誤!** 找不到參照來源。, and the corresponding Path Request Lifetime timer shall be started. Then a PREQ message shall be created according to Section 3.6.5.11.

If a Discovery Table entry associated with a path being discovered still exists after the Path Request Lifetime timer has expired, then the entry shall be deleted and the path is indicated as not found.

#### *3.6.8.2.2 Directed Forwarding Discovery*

The Directed Forwarding Discovery operation shall be executed by any Directed Forwarding node that receives a PREQ. To increase the probability of a successful path discovery, a Directed Forwarding node should process a PREQ only if the message has an RSSI well above the sensitivity of the receiver, and the link is estimated to be bi-directional, according to the information in the Neighbor Table (see Section 3.6.8.5). **錯誤!** 找不到參照來源。 shows the flow chart of the Directed Forwarding Discovery operation.

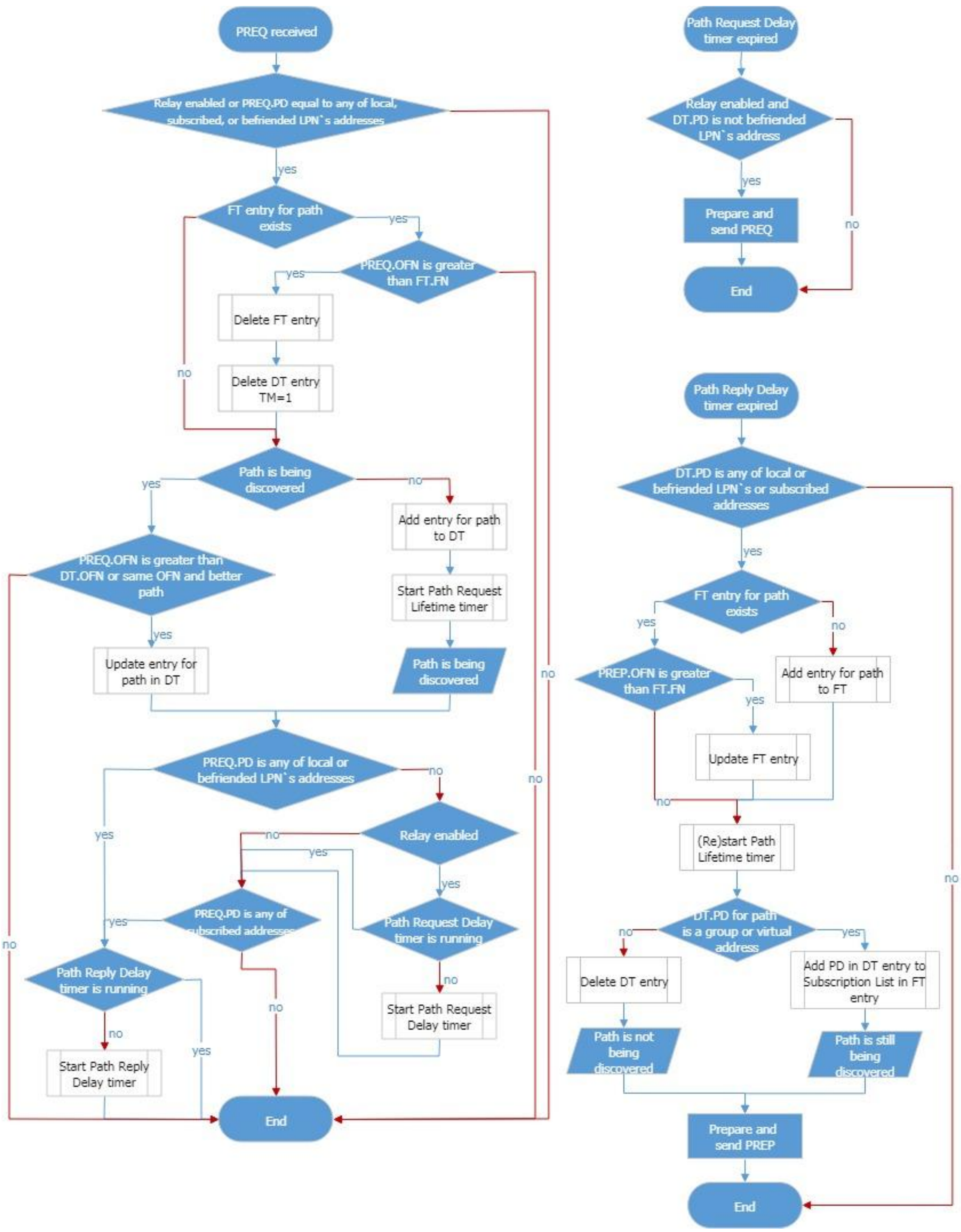


*Figure 3.29: Directed Forwarding Discovery operation*

When a Directed Forwarding node receives a PREQ message, the node checks whether to process or discard the message.

The PREQ message shall be discarded if any of the following conditions is met:

- The node is not a Relay node, and the PD value in the PREQ message is not either an element address of the node or a group address or virtual address that the node subscribed to.
- The node is not a Relay node and is a Friend node, and the PD value in the PREQ message is not either an element address of the associated Low Power nodes or a group address or virtual address that the Low Power node subscribed.
- A Forwarding Table entry exists so that OFN in PREQ is set not greater than FN in the entry, PO in PREQ is set equal to PO in the entry, DPO in PREQ is set equal to DPO in the entry if DPO in PREQ is present, and PD in PREQ is set equal to any value in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry, if PD in PREQ is a unicast address, or PD in PREQ is included in Subscription List in the entry, if PD in PREQ is a group address or a virtual address.
- After the Neighbor Table is updated, the Neighbor Table entry associated with the SRC value in the PREQ message either does not exist or, if it exists, the Neighbor flag (NF) value is 0.

If the PREQ message can be processed, the node checks whether a copy of the message has already been received based on the (PO, DPO, PD) triplet (i.e., the Path Originator, Delegate Path Originator, and Path Destination) and the current entries of the Discovery Table. Message caching (see Section 3.4.6.5) suppresses duplicates of a PREQ message with same SRC and SEQ values but does not filter out messages encapsulating the same PREQ as a transport payload.

If a PREQ message includes a (PO, DPO, PD) triplet that is not present in the Discovery Table, a new entry shall be added to the Discovery Table (see Section 3.6.8.4.2) and the corresponding Path Request Lifetime timer shall be started.

If a PREQ message includes a (PO, DPO, PD) triplet that is stored in a Discovery Table entry (that is, the associated path is being discovered), and the PREQ message came from a better path to the Path Originator or to the Delegate Path Originator than in the existing entry, then the entry shall be updated (see Section 3.6.8.4.2). The selection of the path and the calculation of Originator Path Metric (OPM) to be stored are based on the value of the Path Metric state as specified in Section **錯誤! 找不到參照來源。**.

If a PREQ message includes a (PO, DPO, PD) triplet that is stored in a Discovery Table entry, and the Originator Forwarding Number (OFN) in the PREQ message is lower than the OFN in the table entry or the OFN value in PREQ is set equal to the OFN value in the entry and the PREQ message does not come from a better path to the Path Originator or to the Delegate Path Originator, the PREQ message shall be discarded. The selection of the path and the calculation of Originator Path Metric (OPM) to be stored are based on the value of the Path Metric state as specified in Section **錯誤! 找不到參照來源。**.

When a Directed Relay node receives a PREQ message, and the PD value in the PREQ message is not an element address of the node, the node shall prepare and send a new PREQ message in which the fields of the received PREQ message are modified according to the requirements in Section 3.6.5.11.. In order to collect a number of PREQ copies, PREQ forwarding is delayed by the Path Request Delay interval after the receipt of the first PREQ. To avoid collisions with PREQ messages sent by other nodes, a random perturbation should be added to the Path Request Delay interval.

Before sending the PREQ, the node should verify that the transmission of PREQ messages is enabled. For example, the configuration of the node (such as the values of the Relay state) may be changed during the Path Request Delay time.

If the PD value in the PREQ message is either an element address of the node or a group address or virtual address that the node subscribed to, the node shall add an entry to the Forwarding Table state, copying relevant information from the Discovery Table (see Section 3.6.8.4.2), and shall prepare and send a PREP message according to the requirements in Section **錯誤! 找不到參照來源。**.

If the PREQ message is received by a Directed Forwarding node that is also a Friend node, and the PD value in the message either an element address of the Low Power node or a group address or virtual address that the Low Power node subscribed to, the node shall add an entry to the Forwarding Table state, copying relevant information from the Discovery Table (see Section 3.6.8.4.2), and shall prepare and send a PREP as Delegate Path Destination according to the requirements in Section **錯誤! 找不到參照來源。**.

In order to collect a number of PREQ copies and reply only to the one coming from the best path to the Path Originator or to the Delegate Path Originator (that is, the Backward Path), the PREP transmission shall be delayed by the Path Reply Delay interval after the reception time of the first PREQ message. If a PREP message is sent in response to a PREQ message directed to a group address or to a virtual address, the node should transmit the PREP message with an additional random delay of 0 to 500 milliseconds. This reduces the probability of multiple nodes responding to the PREQ at the same time, and therefore increases the probability of message delivery rather than message collisions.Before sending the PREP, the node should check whether all the conditions to reply to the PREQ message are still met. For example, the configuration of the node (such as the values of the Relay state and the Subscription List state or the status of a friendship with any Low Power node) may change during the Path Reply Delay interval.

If the Path Destination (PD) in the PREQ message is a unicast address, the Discovery Table entry that corresponds to PREQ shall be deleted after the associated Path Reply Delay timer expires. In all the other cases, the entry shall be deleted after the associated Path Request Lifetime time expires.

##### *3.6.8.2.3 Directed Forwarding Establishment*

The Directed Forwarding Establishment operation shall be executed by any Directed Forwarding node that receives a Path Request Reply (PREP).

**錯誤! 找不到參照來源。** illustrates the Directed Forwarding Establishment operation.

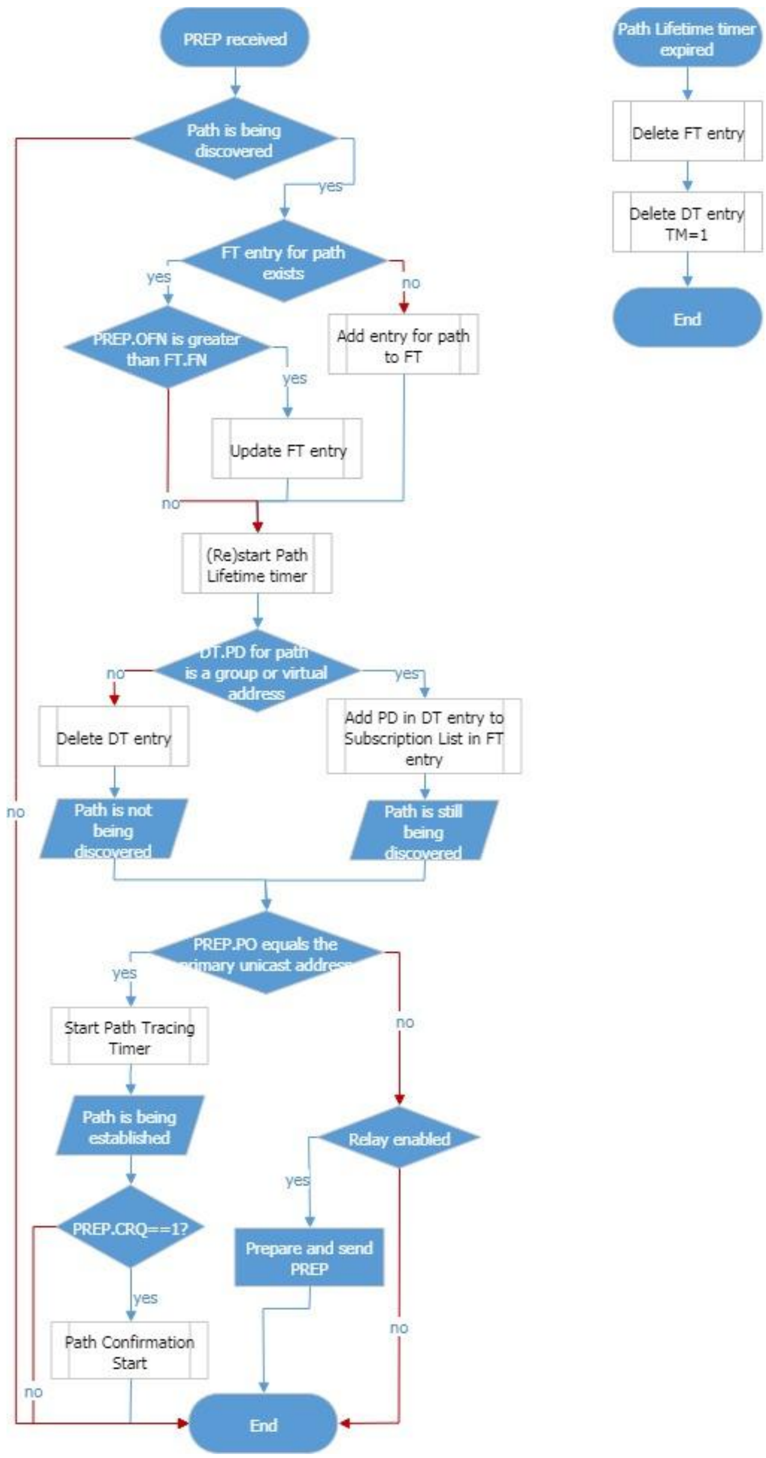


*Figure 3.30: Directed Forwarding Establishment operation*

When a node receives a PREP and the message can be processed (see Section 3.6.8.5), the node checks whether a Discovery Table entry that corresponds to the PREP message exists (see Section 3.6.8.4.2). If the Discovery Table contains a corresponding entry, the values of some fields of the entry shall be copied to a new entry in the Forwarding Table state (see Section 3.6.8.2). A Path Lifetime timer shall be also started for the Forwarding Table entry.  If the Path Destination (PD) in the Discovery Table entry is a unicast address, then the Discovery Table entry shall be removed. If the Discovery Table entry does not exist, the PREP message shall be discarded.

If the PO in a PREP message is the primary unicast address of the node (indicating that the message has reached the Path Originator or the Delegate Path Originator), then a path toward the Path Destination is considered established. If the node is a Directed Relay node, it shall prepare and send a PREP message toward the Path Originator or the Delegate Path Originator according to Section **錯誤! 找不到參照來源。**.

#### *3.6.8.2.4 Directed Forwarding Confirmation*

The Directed Forwarding Confirmation operation shall be initiated by a Path Originator or by a Delegate Path Originator that receives a Path Reply (PREP) message with the Confirmation Request flag (CRQ field) set to 1, and shall be executed by any Directed Forwarding node that receives a Path Confirmation (PCONF) message.

**錯誤! 找不到參照來源。** is a flow chart illustrating the Directed Forwarding Confirmation operation.

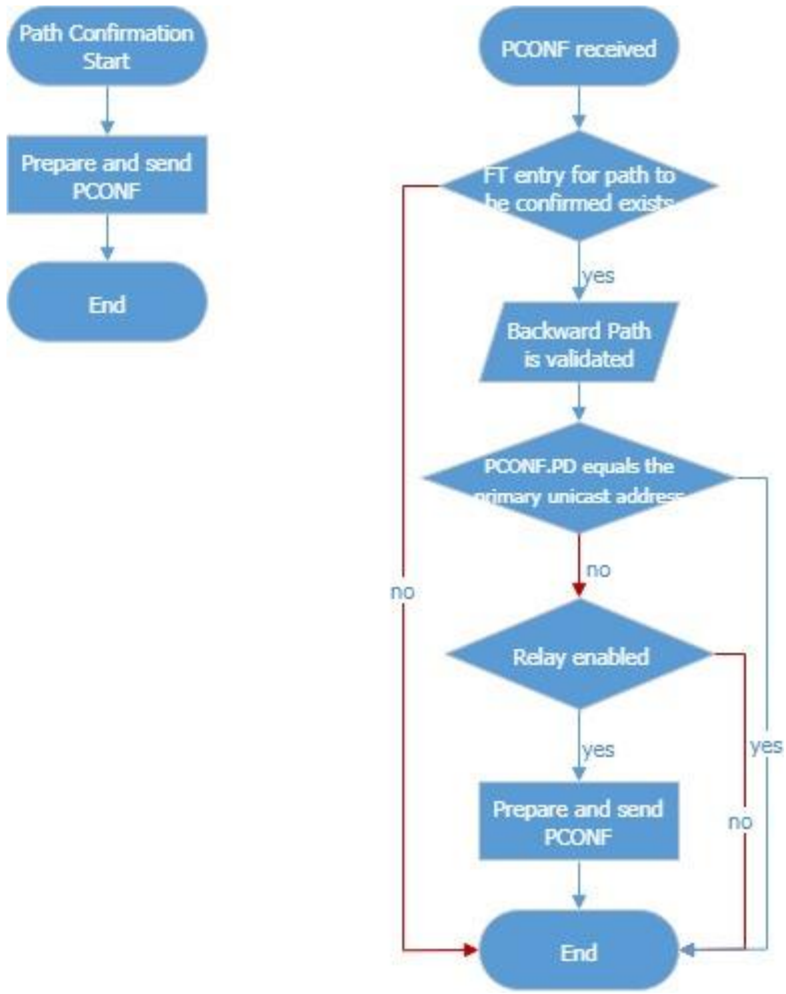


*Figure 3.31: Directed Forwarding Confirmation operation*

When a Path Originator or a Delegate Path Originator receives a PREP message with the CRQ field set to 1 and establishes the related path, the node shall prepare and send a PCONF message directed to the Path Destination or the Delegate Path Destination according to the requirements in Section 3.6.5.12.

When a Directed Forwarding node receives the PCONF message, and the message can be processed (see Section 3.6.8.5), the node checks whether the Forwarding Table has an existing entry that corresponds to the PCONF message (see Section 3.6.8.2). If there is no such entry, the message shall be discarded.

If the table entry exists and the node is either the Path Destination or the Delegate Path Destination, the BackwardPathValidated flag of the entry shall be set to 1, and the entry can be used later to send messages toward the Path Originator.

If the table entry exists and the node is a Directed Relay node that is neither the Path Destination nor the Delegate Path Destination, the BackwardPathValidated flag of the entry shall be set to 1, and the node shall prepare and send a PCONF message toward the Path Destination or the Delegate Path Destination according to the requirements in Section 3.6.5.12.

The BackwardPathValidated flag of a path entry should be set to 1 if there is no other path entry in the same Forwarding Table state that the Path Destination can use to send messages toward the Path Originator.

#### *3.6.8.2.5 Directed Forwarding Error Notification*

The Directed Forwarding Error Notification operation shall be executed by any Directed Forwarding node that finds a path to be invalid or receives a Path Error (PERR) message.

These scenarios, which result in the generation of a PERR message, shall include but are not limited to the following:

- A Directed Forwarding Assistance operation fails (see Section **錯誤! 找不到參照來源。**).
- A Directed Forwarding Assistance operation is not implemented, and a node marks a link with any of its one-hop neighbors as broken by, for example, setting the Neighbor flag in the Neighbor Table to 0 (see Section 3.6.8.5), and the primary unicast address of the neighbor is equal to either the NTO or NTD value in the Forwarding Table state.
- A Friend node that supports the Directed Forwarding feature detects that its friendship with any of the associated Low Power nodes has been lost, and the Friend node is either the Delegate Path Originator or the Delegate Path Destination—that is, the primary unicast address of the Low Power node is equal to the value of either PO or PD in the Forwarding Table state.

**錯誤! 找不到參照來源。** is a flow chart of the Directed Forwarding Error Notification operation.

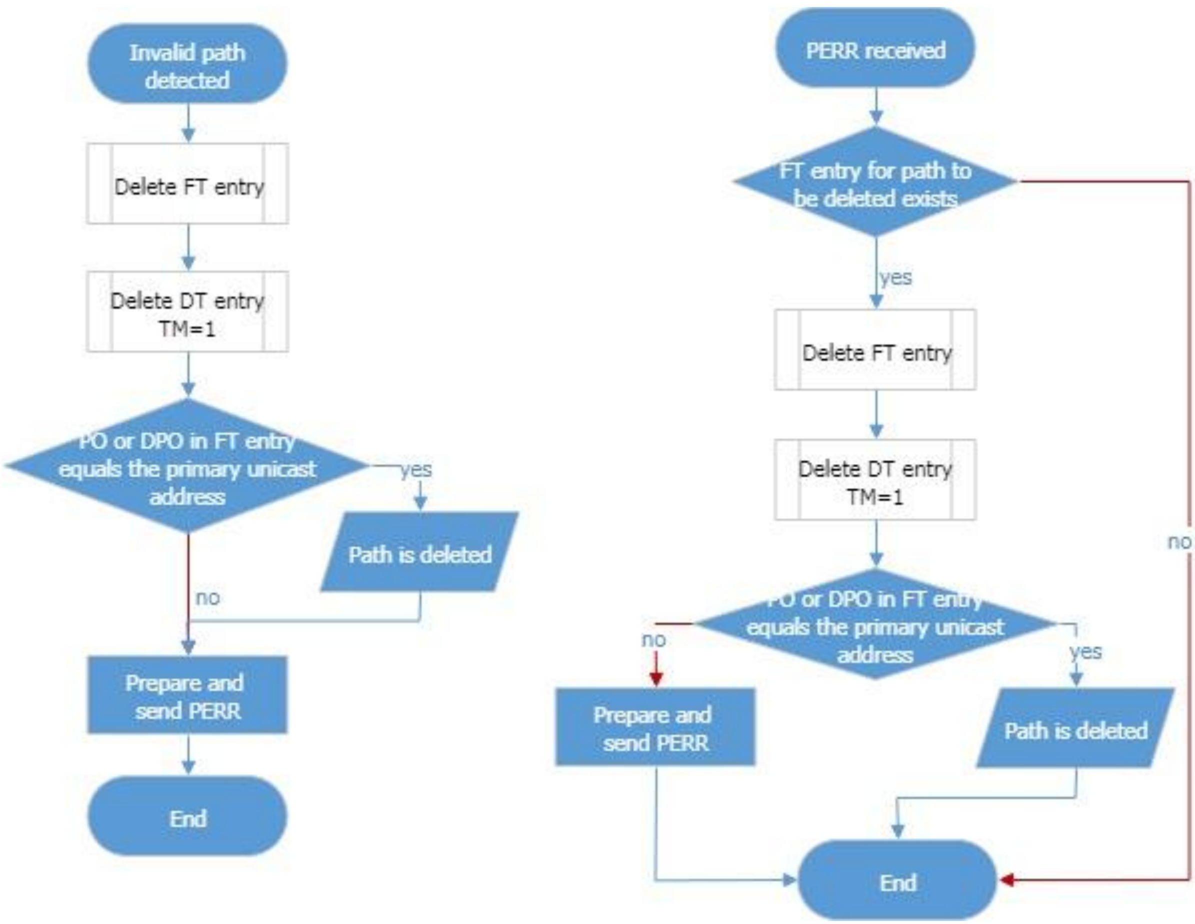


*Figure 3.32: Directed Forwarding Error Notification operation*

A Directed Forwarding node that finds that a path is not valid anymore shall delete the corresponding Forwarding Table entry, shall remove the group addresses and virtual addresses included in the Subscription List field in the entry from the Subscription List field of all other entries in the Forwarding Table state, and shall prepare and send a PERR message according to the requirements in Section 3.6.5.17.

When a node receives a PERR, and the message can be processed (see Section 3.6.8.5), the node checks whether or not an entry that corresponds to the PERR message exists in the Forwarding Table state (see Section 3.6.8.2).

If an entry exists, it shall be removed; if the Subscription List field in the entry is not empty, the group addresses or virtual addresses included in the field shall be removed from the Subscription List field of all other entries in the Forwarding Table state. The node shall then send a new PERR prepared on the basis of the received one according to Section 3.6.5.17.

If a Forwarding Table entry corresponding to the PERR message does not exist (for example, the FN value in the PERR message is less than the stored FN value), the message shall be discarded.

Note: PERR messages should be given high priority for transmission at Directed Forwarding nodes.

##### 3.6.8.2.6 Neighbor Information Management

The Neighbor Information Management operation shall be executed by any Directed Forwarding node if the Neighbor Information Options state is not set to 0x00.

**錯誤! 找不到參照來源。** is a flow chart of the Neighbor Information Management operation.

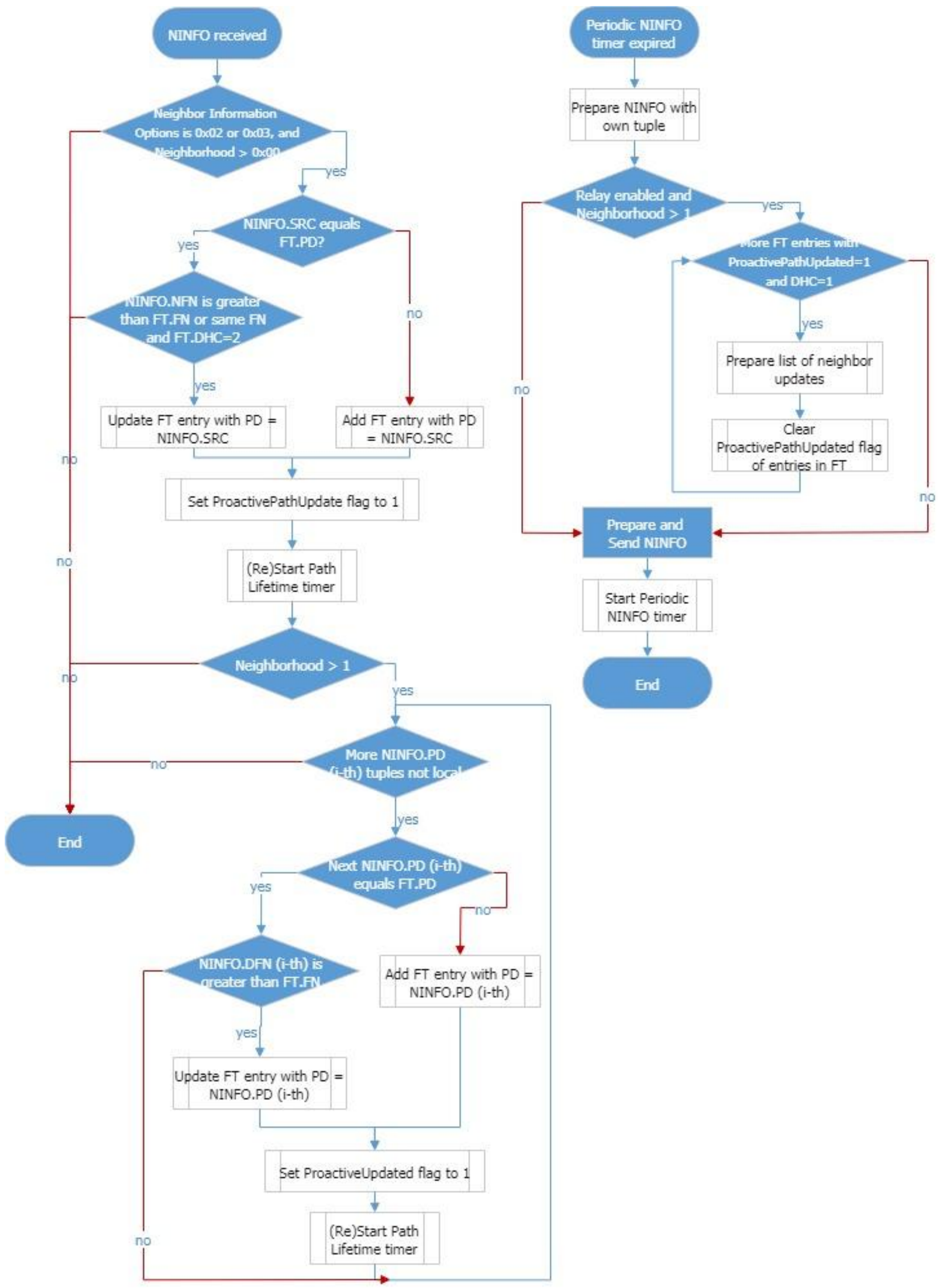


*Figure 3.33: Neighbor Information Management operation*

A node shall send a Neighborhood Information (NINFO) message periodically at the end of every Neighbor Information Period if such a transmission is enabled (that is, if the Neighbor Information Options state is set to 0x01 or 0x03, as defined in Section **錯誤! 找不到參照來源。**) to announce the presence of the node, and, if the node is a Directed Relay node, to announce path updates related to its neighbors that are Directed Forwarding nodes located within a number of hops given by the Neighborhood state minus 1, according to Section **錯誤! 找不到參照來源。**.

When a node receives a NINFO, and NINFO processing is enabled (that is, the Neighbor Information Options state is set to 0x02 or 0x03 according to Section **錯誤! 找不到參照來源。**), the node updates the Neighbor Table and checks for an entry with a Neighbor Address matching the SRC value in the NINFO message. If no such entry exists or the entry exists and its Neighbor flag is set to 0, the NINFO message shall be ignored. Otherwise, the node checks whether or not the NINFO message reports path information updates for the message sender and for its neighbors whose existence is announced by the message (see Section 3.6.8.2). If the value of the Neighborhood state is greater than 0x00, and the NINFO reports a path information update for the message sender or the value of the Neighborhood state is greater than 0x01 and NINFO reports a path information update for a neighbor of the message sender, then the path information update shall be stored in the Forwarding Table state. If any update is stored, and the node is a Directed Relay node, the node shall prepare a NINFO message with only the applied updates according to Section 3.6.5.18 and shall send the message at the end of the current Neighbor Information Period.

Neighbors of the node may use the Wanted RSSI value from a received NINFO message to determine whether the node is able to receive messages from them and to evaluate whether the link is bi-directional.

Note: Information about paths that are established *proactively* through the exchange of NINFO messages cannot be merged with information about paths that are discovered when needed, with the exchange of PREQ, PREP, and PCONF messages. These classes of paths are different in several aspects. For example, different operations are performed to discover the paths, and the absence of a Path Originator in a path is established proactively.

Information about paths established via the exchange of a PREQ and PREP message cannot be updated; it can only be validated. This information may include addresses of secondary elements, groups, and Low Power nodes.

In contrast, information about paths toward nodes in the neighborhood that is obtained by processing NINFO messages can be updated. However, this path information includes only information about Directed Forwarding nodes and the addresses of only the primary elements.

#### *3.6.8.2.7 Directed Forwarding Tracing*

The Directed Forwarding Tracing operation may be executed by any Directed Forwarding node to validate a path in the Forwarding Table state that was discovered by the node; in addition, the operation shall be executed by any Directed Forwarding node that receives a Path Tracing Request (PTREQ) message or a Path Tracing Reply (PTREP) message.

錯誤! 找不到參照來源。 is a flow chart of the Directed Forwarding Tracing operation.

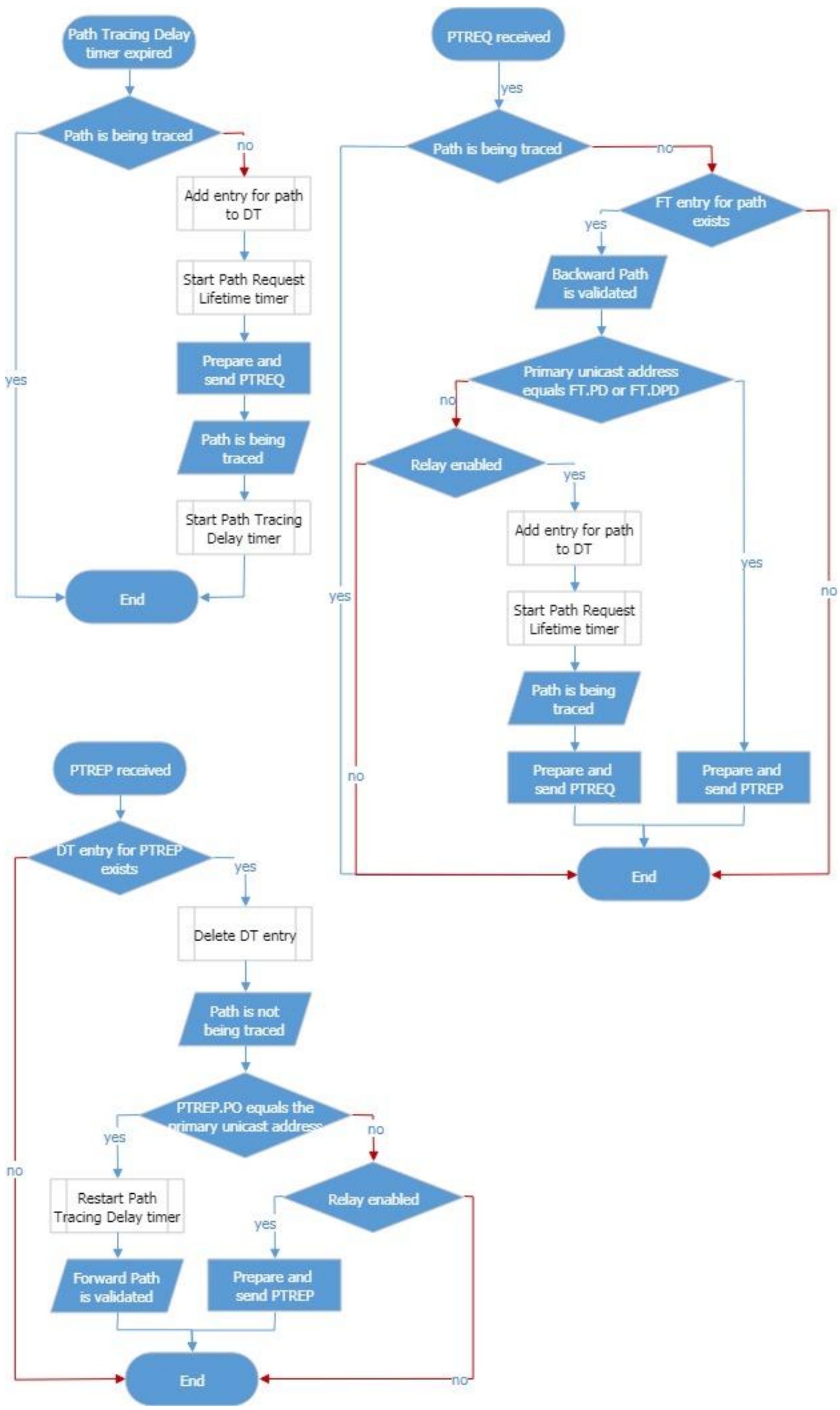


*Figure 3.34: Directed Forwarding Tracing operation*

When a Path Originator or a Delegate Path Originator creates a Forwarding Table entry (see Section 錯誤! 找不到參照來源。), the node may start a Path Tracing Delay timer. When this timer expires, the Path Originator or the Delegate Path Originator shall send a PTREQ message directed to the Path Destination in order to continue validating the associated path. After the Path Tracing Delay timer expires, the timer may be scheduled again for a later tracing operation.

The value of the Path Tracing Delay timer is left to the implementation. In static networks, the Path Tracing Delay timer should be set to more than 10 minutes.

The Path Originator or the Delegate Path Originator checks whether a Discovery Table entry associated with the path that is indicated as being traced exists (see Section 3.6.8.4.2). If the entry exists, another Directed Forwarding Tracing operation shall not be started for the associated path.

If the entry does not exist (that is, the path is not being traced), a new entry associated with the path shall be added to the Discovery Table (see Section 3.6.8.4.2), the corresponding Path Request Lifetime timer shall be started, and a PTREQ message shall be prepared and sent according to Section 3.6.5.13. If the entry associated with the path exists after the Path Request Lifetime timer expires, then both the Discovery Table entry and the Forwarding Table entry associated with the path shall be removed.

When a Directed Forwarding node receives the PTREQ message, the node checks whether to process or to discard the PTREQ message.

The PTREQ message shall be discarded if any of the following conditions is met:

- The node is not a Relay node, and the PD value in the PTREQ message is not any of the element addresses of the node or a group address or virtual address that the node is subscribed to.
- The node is not a Relay node and is a Friend node, and the PD value in the PTREQ message is not any of the element addresses of the associated Low Power nodes or a group address of virtual address that the Low Power nodes are subscribed to.
- The Discovery Table contains an entry in which the FN value is less than or equal to the OFN value in the PTREQ message, the PO value in the PTREQ message is equal to the PO value in the table entry and, if the PTREQ contains a DPO field, the DPO value in the PTREQ message is equal to the DPO value in the table entry.
- The Forwarding Table contains an entry in which the FN value is not equal to the OFN value in the PTREQ message, the PO value is equal to the PO value I the PTREQ message, the DPO value is equal to the DPO value in the PTREQ message if the message contains a DPO field, and the PD value is equal to the PD value in the PTREQ message.
- After the Neighbor Table is updated, the Neighbor Table entry associated with the SRC value in the PTREQ message either does not exist or, if it exists, the Neighbor flag (NF) value is 0.

If the PTREQ message is processed and the node is neither the Path Destination nor the Delegate Path Destination of PTREQ, a new PTREQ message shall be prepared and sent according to the requirements in Section 3.6.5.13. This allows propagating PTREQ messages only along the path that is being traced. If the PTREQ message is propagated, a new entry shall be added to the Discovery Table, and the corresponding Path Request Lifetime timer shall be started. If the entry associated with the path exists after the Path Request Lifetime timer expires, then both the Discovery Table entry and the Forwarding Table entry associated with the path shall be removed.

If the PTREQ message is processed, and the node is the Path Destination or the Delegate Path Destination of PTREQ (see Section 3.6.8.2), then the node shall prepare and send a PTREP message according to the requirements in Section 3.6.5.14.

When a node receives a PTREP message, and the message can be processed (see Section 3.6.8.5), the node checks whether an entry that corresponds to the PTREP message exists in the Discovery Table (see Section 3.6.8.4.2). If no such entry exists, then the PTREP message shall be discarded.

If the Discovery Table contains an entry corresponding to the PTREP message, the node checks whether the PO value in the PTREP message is primary unicast address of the node (that is, the message has reached the Path Originator or the Delegate Path Originator). If so, the path is indicated as validated. If the node is a Directed Relay node, the node shall prepare and send a new PTREP message toward the Path Originator or the Delegate Path Originator according to the requirements in Section 3.6.5.14.

#### *3.6.8.2.8 Directed Forwarding Assistance*

The Directed Forwarding Assistance operation may be executed by any Directed Forwarding node to request that another node in the same subnet join an existing path and shall be executed by a Directed Forwarding node that receives a PAREQ.

**錯誤! 找不到參照來源。** is a flow chart of the Directed Forwarding Assistance operation.

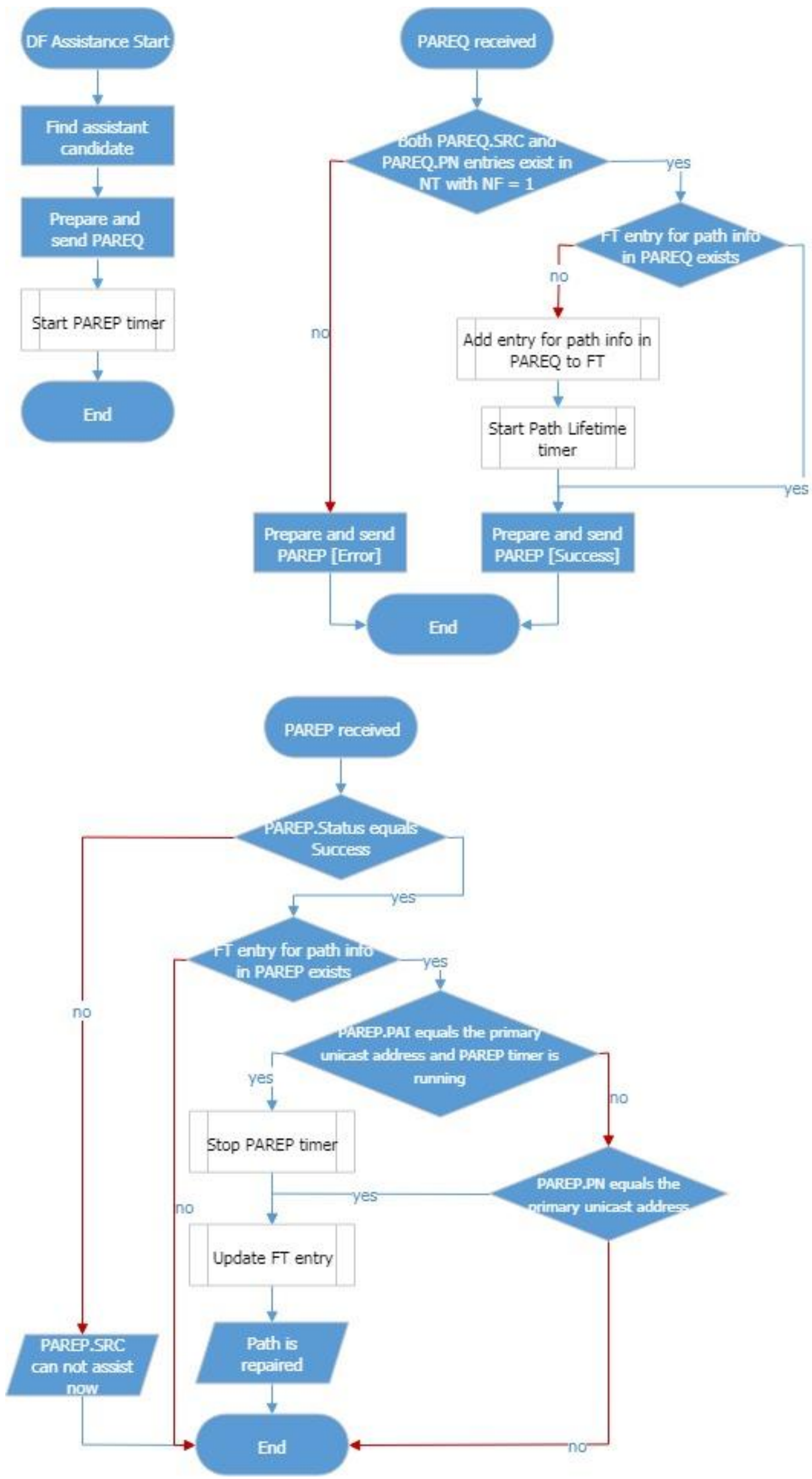
DF Assistance Start
Find assistant candidate
Prepare and send PAREQ
Start PAREP timer
End
PAREQ received
Both PAREQ.SRC and PAREQ.PN entries exist in NT with NF = 1
yes
FT entry for path info in PAREQ exists
no
Add entry for path info in PAREQ to FT
Start Path Lifetime timer
no
yes
Prepare and send PAREP [Error]
Prepare and send PAREP [Success]
End
PAREP received
PAREP.Status equals Success
yes
FT entry for path info in PAREP exists
yes
no
PAREP.PAI equals the primary unicast address and PAREP timer is running
yes
no
Stop PAREP timer
PAREP.PN equals the primary unicast address
yes
no
Update FT entry
PAREP.SRC can not assist now
Path is repaired
End
no

*Figure 3.35: Directed Forwarding Assistance operation*

This operation may be initiated by a node that marks a link with any of its one-hop neighbors as broken, for example, setting the associated Neighbor flag in the Neighbor Table to 0 (see Section 3.6.8.5), and the primary unicast address of the neighbor is equal to the value of either NTO or NTD in the Forwarding Table state. The neighbor of the initiator node that is no more reachable directly is indicated as peer node (PN).

The initiator node shall send a PAREQ message to the primary unicast address of a candidate assistant node. The candidate assistant node should be a Directed Relay node that is one-hop neighbor of both the initiator node and the peer node. The choice of the candidate assistant node is an implementation detail.

Note: The initiator node can send multiple PAREQ messages reporting information of different path entries that become invalid after that a common intermediate link breaks. The choice of the number of different PAREQ messages to be sent is left to implementation.

After sending the PAREQ message, a PAREP timer shall be started. The timer should be set to 500 ms. If the expected PAREP message is not received before the timer expires or the PAREP message is received with a Status value different from Success (0x00), the Directed Forwarding Assistance operation is considered failed.

If the Directed Forwarding Assistance operation fails, the node may repeat the operation with the same or another candidate assistant node. Alternatively, the node shall initiate a Directed Forwarding Error Notification operation.

If a node receives a PAREQ message, but either the SRC address or the PN address in the message is not present in the Neighbor Table with Neighbor flag set to 1, the node shall send a PAREP message with the Status field set to 0x13 (see Table 4.132).

If a node receives a PAREQ message and a corresponding Forwarding Table entry exists, the node shall send a PAREP message with the Status field set to Success.

If a node receives a PAREQ message and accepts the request, it shall add a new entry into the Forwarding Table state with information reported in the message (see Section 3.6.8.2). The node shall also send a PAREP message with Status set to Success and prepared according to Section 3.6.5.16. The PAREP message should be sent multiple times to increase the probability of correct delivery.

If a PAREP message with Status set to Success is received and either the PAI value or the PN value in the PAREP message is set equal to the primary unicast address of the receiving node, the node checks whether a Forwarding Table entry that corresponds to the PAREP message exists (see Section 3.6.8.2). If the entry exists, it shall be modified as defined in Section 3.6.8.2 and the associated path is considered locally repaired (that is, a path is established from the initiator node to the peer node via the assistant node).

If a PAREP message with Status set to a value different from Success is received or both the PAI value and the PN value in the PAREP message are set to values different from the primary unicast address of the receiving node, the PAREP message shall be discarded.

#### 3.6.8.2.9 Link Quality Measurement

The Link Quality Measurement operation may be executed by any Directed Forwarding node to measure the quality of the link with a neighbor that is a Directed Forwarding node and shall be executed by any Directed Forwarding node that receives a LREQ message or a LREP message.

錯誤! 找不到參照來源。 is a flow chart of the Link Quality Measurement operation.

A Directed Forwarding node should avoid using links characterized by significant signal loss, because they would compromise the reliability of the Directed Forwarding operations. Consequently, a Directed Forwarding node may evaluate the quality of the link with one of its neighbors before sending messages over the link. The information about the quality of links with neighbors may be already available in the Neighbor Table of the node, as defined in Section 3.6.8.5, but this information may be considered outdated or not properly rated. The node may then initiate a Link Quality Measurement operation involving a selected neighbor; as a consequence, the node shall send a LREQ message directed to the neighbor.

The LREQ message shall be created according to Section 3.6.5.19. The node shall start with a Session ID field value set to 0x00 and shall increment the Session ID field value for each new Link Quality Measurement operation, regardless of the destination of the LREQ message.

A Link Reply Wait timer associated with the reception of a LREP message from the selected neighbor shall be also started and set to expire after 150 milliseconds. It the Link Reply Wait timer is already running when the node initiates a new Link Quality Measurement operation for the same neighbor, then the node shall abort the operation. However, the node may initiate multiple Link Quality Measurement operations toward different neighbors concurrently.

When a node receives a LREQ message, the node shall prepare and send a LREP including the RSSI measured upon receiving the LREQ message. The LREP message shall be created according to Section 3.6.5.20.

When a node receives a LREP message, the node checks whether a Link Reply Wait timer associated with the SRC value of the LREP message is running and the Session ID field in the LREP message is set equal to the Session ID field in the LREQ message sent previously. If the timer is running and the field values are equal, the timer shall be stopped and the entry in the Neighbor Table associated with the SRC value of the LREP message shall be updated according to the information reported in the LREP message.

If the Link Reply Wait timer associated with a given neighbor of the node expires, the entry in the Neighbor Table associated with that neighbor may also be updated: the information recorded in this case is an implementation choice.

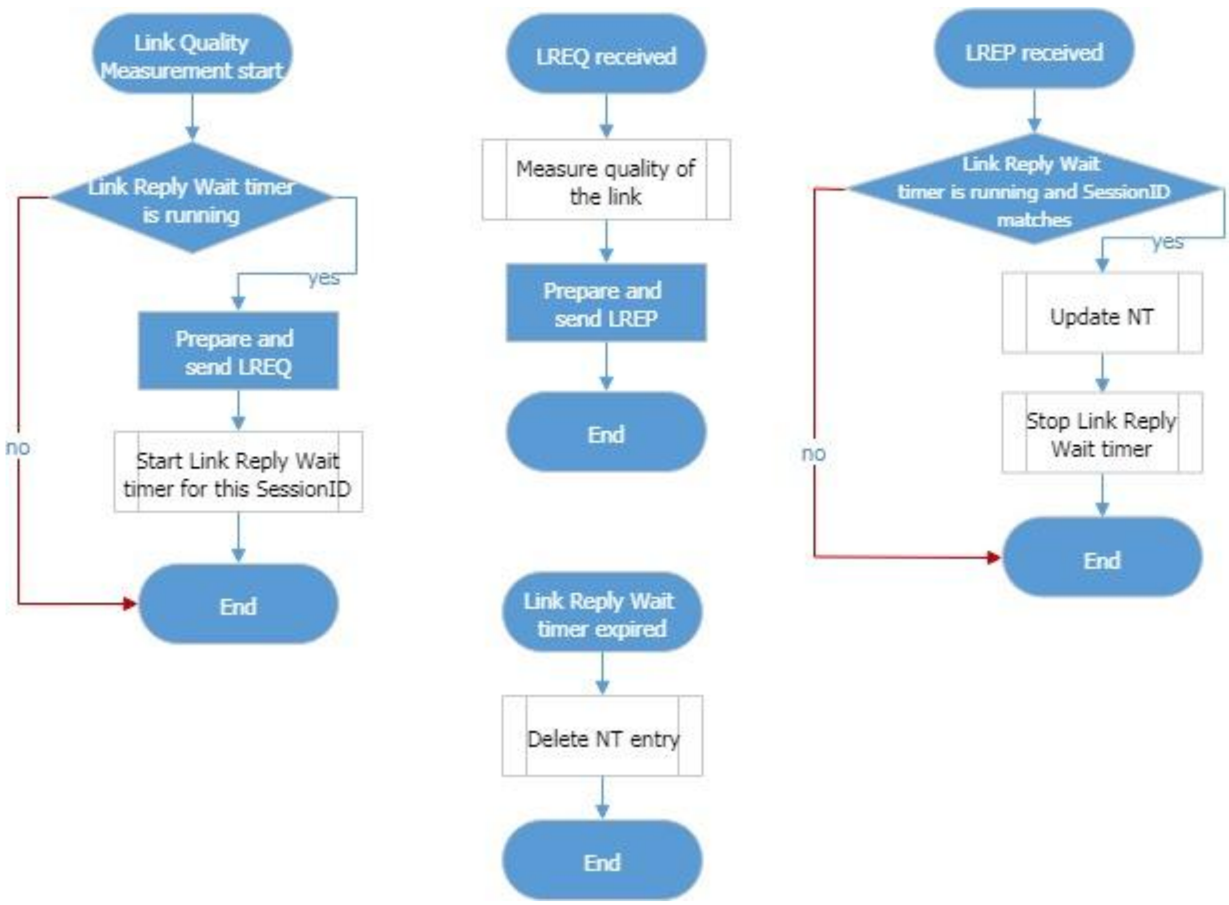


*Figure 3.36: Link Quality Measurement operation*

### 3.6.8.3 **Forwarding Number**

The Forwarding Number (FN) used in Directed Forwarding control messages is initialized to zero.

FN shall be incremented with each new transmission of a PREQ message or of a NINFO message: The Forwading Number becomes the Originator Forwarding Number (OFN) in a Path Requst (PREQ), and it becomes the Node Forwarding Number (NFN) in a Neighbor Information (NINFO) message. A NINFO may also report FN values generated by other nodes, which are indicated as Destination Forwarding Number (DFN) values.

Considering the generally very different frequency of transmissions of control messages that are used for path discovery and establishment (that is, PREQ, PREP, and PCONF messages) and of periodic control messages (that is, NINFO messages), and taking into account variability of an FN, different FN value spaces should be defined for the PREQ message and the NINFO message. To allow a node that belongs to multiple subnets to reuse FN values that have been used in a different subnet, a separate FN value space should be defined for each subnet.

PREP shall report the OFN value of the PREQ message generated by the Path Originator or by the Delegate Path Originator for the discovery of the same path.

PCONF, PTREQ, PTREP, and PERR shall report the value of the FN field in the entry of the Forwarding Table state associated with the path.

#### 3.6.8.4 Forwarding Table

Each Directed Forwarding node filters messages to be forwarded based on its Forwarding Table state, as defined in Section 錯誤! 找不到參照來源。. The Forwarding Table state is updated by Directed Forwarding operations that the node is involved in (see Section 3.6.8.1) or as an effect of processing of configuration messages issued by a Provisioner (see Section 4.3.4). This section describes how the processing of messages exchanged in the context of Directed Forwarding operations affects the management of non-fixed path entries in Forwarding Table state. Every insertion, modification, and deletion specified in this section for a Forwarding Table entry shall be executed only if the Fixed field in the entry is set to 0.

The Forwarding Table state is initially empty. It shall be updated whenever any of the following messages is received and processed.

When a PREP is received (see Section 錯誤! 找不到參照來源。), the node checks whether a Discovery Table entry that corresponds to the PREP message exists (see Section 3.6.8.4.2).

If a Discovery Table entry that corresponds to the PREP message exists, the node checks whether a Forwarding Table entry that corresponds to the PREP message exists.

The Forwarding Table entry that corresponds to the PREP message exists if all the following conditions are met:

- OBOO in PREP is set to 0 and PO in PREP is set to the PO value in the entry, or OBOO in PREP is set to 1 and PO in PREP is set to the DPO value in the entry.
- PD in PREP is set to the PD value in the entry.
- OFN in PREP is set to a value that is not lower than the FN value in the entry or a Discovery Table entry corresponding to the PREP message exists (see Section 3.6.8.4.2).

If a Forwarding Table entry that corresponds to the PREP message does not exist, a new entry shall be added to the Forwarding Table state and shall be set as follows:

- Fixed, BackwardPathValidated, ProactivePathUpdated, DHC: Set to 0.
- PO, DPO, OAR, BTO: Set to the values of the identically named fields in the Discovery Table entry that corresponds to the PREP message.
- FN: Set to the OFN value in the Discovery Table entry that corresponds to the PREP message.

- PD: Set to the PD value in PREP.
- DPD: Set to the DPD value in PREP if OBOD in PREP is set to 1; set to the unassigned address otherwise.
- DAR: Set to the DAR value in PREP if EA in PREP is set to 1; set to 0 otherwise.
- NTO: Set to the SRC value in the Discovery Table entry that corresponds to the PREP message.
- NTD: Set to the SRC value in PREP.
- Subscription List: Added the PD value in the Discovery Table entry that corresponds to the PREP message if the PD value is a group address or a virtual address.
- Path Expiration Time: Set to the sum of the current time and the time indicated by the Path Lifetime value in the Path Metric state (see Section **錯誤! 找不到參照來源。**).
- BTD: Set to the index of the bearer from which PREP is received.

If a Forwarding Table entry that corresponds to the PREP message exists, the following fields of the entry shall be updated:

- FN: Set to the greatest value between the FN value in the Forwarding Table entry and the OFN value in PREP.
- NTO: Set to the SRC value in the Discovery Table entry that corresponds to the PREP message.
- NTD: Set to the SRC value in PREP.
- Subscription List: Added the PD value in the Discovery Table entry that corresponds to the PREP message if the PD value is a group address or a virtual address.
- Path Expiration Time: Set to the sum of the current time and the time indicated by the Path Lifetime value in the Path Metric state (see Section **錯誤! 找不到參照來源。**).
- BTO: Set to the BTO value in the Discovery Table entry that corresponds to the PREP message.
- BTD: Set to the index of the bearer from which PREP is received.

When a PCONF (see Section 3.6.5.12) is received, the node checks whether a Forwarding Table entry that corresponds to the PCONF message exists.

The Forwarding Table entry that corresponds to the PCONF message exists if all the following conditions are met:

- OBOO in PCONF is set to 0 and PO in PCONF is set to the PO value in the entry, or OBOO in PCONF is set to 1 and PO in PCONF is set to the DPO value in the entry.
- PD in PCONF is set to the PD value in the entry.
- OFN in PCONF is set to the FN value in the entry.

If a Forwarding Table entry that corresponds to the PCONF message exists, the BackwardPathValidated field of the entry shall be set to 1.

Note: The BackwardPathValidated flag of a path entry should be set to 1 if there is no other path entry in the same Forwarding Table state that can be used by the Path Destination to send messages toward the Path Originator.

When a PTREQ (see Section 3.6.5.12) is received, the node checks whether a Forwarding Table entry that corresponds to the PTREQ message exists.

The Forwarding Table entry that corresponds to the PTREQ message exists if the following condition is met:

- PO and PD in PTREQ are set to the values of the fields with the same names in the entry.

If a Forwarding Table entry that corresponds to the PTREQ message exists, the BackwardPathValidated field of the entry shall be set to 1.

When a PAREQ (see Section 3.6.5.15) is received and can be processed (see Sections **錯誤! 找不到參照來源。** and 3.6.8.5), the node checks whether a Forwarding Table entry that corresponds to the PAREQ message exists.

The Forwarding Table entry that corresponds to the PAREQ message exists if all the following conditions are met:

- OBOO in PAREQ is set to 0 and PO in PAREQ is set to the PO value in the entry, or OBOO in PAREQ is set to 1 and DPO in PAREQ is set to the DPO value in the entry.
- PD in PAREQ is set to the PD value in the entry.

If a Forwarding Table entry that corresponds to the PAREQ message does not exist, a new entry is added to the Forwarding Table state and shall be set as follows:

- Fixed, ProactivePathUpdated, DHC: Set to 0.
- BackwardPathValidated: Set to the BPV value in PAREQ.
- PO, FN, PD: Set to the values of the identically named fields in PAREQ.

- DPO: Set to the DPO value in PAREQ if OBOO in PAREQ is set to 1; set to the unassigned address otherwise.
- DPD: Set to the DPD value in PAREQ if OBOD in PAREQ is set to 1; set to the unassigned address otherwise.
- OAR: Set to the OAR value in PAREQ if OEA in PAREQ is set to 1; set to 0 otherwise.
- DAR: Set to the DAR value in PAREQ if DEA in PAREQ is set to 1; set to 0 otherwise.
- NTO: Set to the SRC value in PAREQ if DIR in PAREQ is set to 1; set to the PN value in PAREQ otherwise.
- NTD: Set to the PN value in PAREQ if DIR in PAREQ is set to 1; set to the SRC value in PAREQ otherwise.
- BTO: Set to the index of the bearer from which PAREQ is received if DIR in PAREQ is set to 1; set to the BID value in PAREQ otherwise.
- BTD: Set to the BID value in PAREQ if DIR in PAREQ is set to 1; set to the index of the bearer from which PAREQ is received otherwise.
- Path Expiration Time: Set to the sum of the current time and the Path Remaining Time value in PAREQ.
- Subscription List: Set to the list of values in the Subscription List field in PAREQ if SL in PAREQ is set to 1; left empty otherwise.

When a PAREP (see Section 3.6.5.16) is received and can be processed (see Sections **錯誤! 找不到參照來源。** and 3.6.8.5), the node checks whether a Forwarding Table entry that corresponds to the PAREP message exists.

The Forwarding Table entry that corresponds to the PAREP message exists if all the following conditions are met:

- OBOO in PAREP is set to 0 and PO in PAREP is set to the PO value in the entry, or OBOO in PAREP is set to 1 and PO in PAREP is set to the DPO value in the entry.
- PD and FN in PAREP are set to the values of the fields with the same names in the entry.

If a Forwarding Table entry that corresponds to the PAREP message exists, either of the (NTO, BTO) and (NTD, BTD) field pairs of the entry shall be modified if either of the following conditions is met:

- If PN in PAREP is set to the primary unicast address of the node and the NTO value in the entry is equal to the PAI value in PAREP or PAI in PAREP is set to the primary unicast address of the node and the NTO value in the entry is equal to the PN value in PAREP:

- NTO: Set to the SRC value in PAREP.
- BTO: Set to the index of the bearer from which PAREP is received.

- If PN in PAREP is set to the primary unicast address of the node and the NTD value in the entry is equal to the PAI value in PAREP or PAI in PAREP is set to the primary unicast address of the node and the NTD value in the entry is equal to the PN value in PAREP:
  - NTD: Set to the SRC value in PAREP.
  - BTD: Set to the index of the bearer from which PAREP is received.

When a NINFO is received (see Section 3.6.5.18), the node checks whether a Forwarding Table entry associated with each of the node addresses reported in the message (that is, SRC and i-th PD) exists.

A Forwarding Table entry associated with a node address in the message exists if the node address is equal to the PD value in the entry and the PO value in the entry is set to the unassigned address.

If a Forwarding Table entry associated with a node address in the message does not exist, a new entry shall be added to the Forwarding Table state and shall be set as follows:

- Fixed, BackwardPathValidated: Set to 0.
- ProactivePathUpdated: Set to 1.
- PO, DPO, DPD, NTO: Set to the unassigned address.
- OAR: Set to 0xFF.
- NTD: Set to the SRC value in NINFO.
- Subscription List: Left empty.
- Path Expiration Time: Set to the sum of the current time and the time indicated by the Path Lifetime value in the Path Metric state (see Section **錯誤! 找不到參照來源。**).
- BTO: Set to the unassigned bearer index.
- BTD: Set to the index of the bearer from which NINFO is received.

If the entry is associated with the SRC value in NINFO, the FN, PD, DHC, and DAR fields in the entry shall be set as follows:

- FN: Set to the NFN value in NINFO.
- PD: Set to the SRC value in NINFO.

- DHC: Set to 1.
- DAR: Set to the NAR value in NINFO if the NEA value in NINFO is set to 1; set to 0 otherwise.

If the entry is associated with the i-th PD value in NINFO, the FN, PD, DHC, and DAR fields in the entry shall be set as follows:

- FN: Set to the i-th DFN value in NINFO.
- PD: Set to the i-th PD value in NINFO.
- DHC: Set to 2.
- DAR: Set to the i-th DAR value in NINFO if the i-th DAR field is present (see Section 3.6.5.18); set to 0 otherwise.

If a Forwarding Table entry associated with a node address in the message exists, the node checks whether NINFO reports an updated information associated with the node address.

If the entry is associated with the SRC value in NINFO, an updated information is reported if either of the following conditions is met:

- The NFN value in NINFO is greater than the FN value in the entry.
- The NFN value in NINFO is equal to the FN value in the entry and DHC in the entry is set to 2.

If an updated information is reported, the ProactivePathUpdated, FN, DHC, DAR, NTD, Path Expiration Time, and BTD fields in the entry shall be set as follows:

- ProactivePathUpdated: Set to 1.
- FN: Set to the NFN value in NINFO.
- DHC: Set to 1.
- DAR: Set to the NAR value in NINFO if the NEA value in NINFO is set to 1; set to 0 otherwise.
- NTD: Set to the SRC value in NINFO.
- Path Expiration Time: Set to the sum of the current time and the time indicated by the Path Lifetime value in the Path Metric state (see Section **錯誤! 找不到參照來源。**).
- BTD: Set to the index of the bearer from which NINFO is received.

If the entry is associated with the i-th PD value in NINFO, an updated information is reported if the i-th DFN value in NINFO is greater than the FN value in the entry.

If an updated information is reported, the ProactivePathUpdated, FN, DHC, DAR, NTD, Path Expiration Time, and BTD fields in the entry shall be set as follows:

- ProactivePathUpdated: Set to 1.
- FN: Set to the i-th DFN value in NINFO.
- DHC: Set to 2.
- DAR: Set to the i-th DAR value in NINFO if the i-th DAR field is present (see Section 3.6.5.18); set to 0 otherwise.
- NTD: Set to the SRC value in NINFO.
- Path Expiration Time: Set to the sum of the current time and the time indicated by the Path Lifetime value in the Path Metric state (see Section **錯誤! 找不到參照來源。**).
- BTD: Set to the index of the bearer from which NINFO is received.

When a NINFO is transmitted and for each of the node addresses reported in the message, the node shall setthe ProactivePathUpdated field in the Forwarding Table entry associated with the node address to 0.

When a PERR is received (see Section 3.6.5.17), the node checks whether a Forwarding Table entry that corresponds to the PERR message exists.

The Forwarding Table entry that corresponds to the PERR message exists if FN in PERR is set to a value that is greater than or equal to the FN value in the entry and either of the following conditions is met:

- WPO in PERR is set to 0, PD in PERR is set to the PD value in the entry, and PO in the entry is set to the unassigned address.
- WPO in PERR is set to 1, PD in PERR is set to the PD value in the entry, PO in PERR is set to the PO value in the entry and, if OBOO in PERR is set to 1, DPO in PERR is set to the DPO value in the entry.

If a Forwarding Table entry that corresponds to the PERR message exists, the entry shall be removed.

A Forwarding Table entry shall be also removed if any of the following conditions is met:

- The current time is greater than the Path Expiration Time value in the entry.
- A PREQ is received or prepared to be sent in the context of a Directed Forwarding Initialization operation (see Section **錯誤! 找不到參照來源。**), BackwardPathValidated in the entry is set to 0, OFN in PREQ is set greater than FN in the entry, PO in PREQ is set equal to PO in the entry, DPO in PREQ is set equal to DPO in the entry if DPO in PREQ is present, and PD in PREQ is set equal to

any value in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry, if PD in PREQ is a unicast address, or PD in PREQ is included in Subscription List in the entry, if PD in PREQ is a group address or a virtual address.

- The entry corresponds to a Discovery Table entry that is deleted because the current time is greater than the Path Request Expiration value in the Discovery Table entry (see Section 3.6.8.4.2).
- The node infers that the path associated with the entry is not valid anymore (for example, after detecting that a link is broken).

If a PREQ is received or prepared to be sent in the context of a Directed Forwarding Initialization operation (see Section 錯誤! 找不到參照來源。), BackwardPathValidated in the entry is set to 1, PO in PREQ is set equal to either PD or DPD in the entry, and PD in PREQ is set equal to either PO or DPO in the entry, then the BackwardPathValidated flag shall be set to 0.When an access layer message or a Transport Control message that is not a Directed Forwarding control message is received, the node checks whether an entry associated with a path for the message forwarding exists in the Forwarding Table state.

A Forwarding Table entry associated with a path for the forwarding of a message with DST set to a unicast address exists if either of the following conditions is met:

- PO in the entry is set to a unicast address, SRC in the message is set to any address in the range from the PO value in the entry to the sum of the PO value and the OAR value in the entry, and DST in the message is set to any address in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry.
- DPO in the entry is set to a unicast address, SRC in the message is set to the DPO value, and DST in the message is set to any address in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry.
- PO and DPD in the entry are set to unicast addresses, SRC in the message is set to any address in the range from the PO value in the entry to the sum of the PO value and the OAR value in the entry, and DST in the message is set to the DPD value.
- DPO and DPD in the entry are set to unicast addresses, SRC in the message is set to the DPO value, and DST in the message is set to the DPD value.
- PO in the entry is set to a unicast address, BackwardPathValidated in the entry is set to 1, SRC in the message is set to any address in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry, and DST in the message is set to any address in the range from the PO value in the entry to the sum of the PO value and the OAR value in the entry.
- DPO in the entry is set to a unicast address, BackwardPathValidated in the entry is set to 1, SRC in the message is set to any address in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry, and DST in the message is set to the DPO value.

- PO and DPD in the entry are set to unicast addresses, BackwardPathValidated in the entry is set to 1, SRC in the message is set to the DPD value, and DST in the message is set to any address in the range from the PO value in the entry to the sum of the PO value and the OAR value in the entry.

- DPO and DPD in the entry are set to unicast addresses, BackwardPathValidated in the entry is set to 1, SRC in the message is set to the DPD value, and DST in the message is set to the DPO value.

- PO in the entry is set to the unassigned address, and DST in the message is set to any address in the range from the PD value in the entry to the sum of the PD value and the DAR value in the entry.

A path for the forwarding of a message with DST set to a unicast address exists if a Forwarding Table entry associated with the path exists.

A Forwarding Table entry associated with a path for the forwarding of a message with DST set to either a group address or a virtual address exists if the DST value is included in the Subscription List field in the entry.

If the node is the originator of a message with DST set to either a group address or a virtual address, that is, SRC in the message is set to the address of the node, the node assumes a path for the forwarding of the message exists if either of the following conditions is met:

- The number of nodes (or elements) subscribed to DST is known and is equal to the number of Forwarding Table entries associated with the path.

- The number of nodes (or elements) subscribed to DST is unknown and at least one Forwarding Table entry associated with the path exists.

If the node is not the originator of a message with DST set to either a group address or a virtual address, the node assumes a path for the forwarding of the message exists if at least one Forwarding Table entry associated with the path exists.

If a path for the forwarding of a message exists, the message can be forwarded.

When establishing a path towards a group address or a virtual address, there is a possibility that a bi-directional path from one of the nodes subscribed to the group towards the Path Originator already exists. This may result in having multiple entries in the Forwarding Table associated with the same path. An implementation should be able to merge duplicated information contained in these entries into a single Forwarding Table entry.

##### *3.6.8.4.1 Examples of Forward Path and Backward Path entries*

This section provides some examples of tuples extracted from Forwarding Table entries associated with Forward Paths and corresponding Backward Paths, in presence and in absence of delegate nodes, and in correspondence of use of either a reactive approach (that is, when a Path Originator exists) or a proactive approach (that is, when a Path Originator does not exist) of path establishment. The examples are based on the use of the network illustrated in 錯誤! 找不到參照來源。 and on the assumption that all nodes support only the primary unicast address.

If node 1 is the Path Originator and node 5 is the Path Destination, and the path between them does not have delegate nodes, the Forward Path is represented by the tuple (PO=0x0010, DPO=0x0000, DPD=0x0000, PD=0x0050) and the Backward Path is represented by the tuple (PO=0x0050, DPO=0x0000, DPD=0x0000, PD=0x0010).

If node 1 is the Path Originator, node 2 is the Delegate Path Originator, and node 5 is the Path Destination, possible Forward Paths are represented by the tuples (PO=0x0010, DPO=0x0020, DPD=0x0000, PD=0x0050) and (PO=0x0020, DPO=0x0000, DPD=0x0000, PD=0x0050), and possible Backward Paths are represented by the tuples (PO=0x0050, DPO=0x0000, DPD=0x0020, PD=0x0010) and (PO=0x0050, DPO=0x0000, DPD=0x0000, PD=0x0020).

If node 1 is the Path Originator, node 4 is the Delegate Path Destination, and node 5 is the Path Destination, possible Forward Paths are represented by the tuples (PO=0x0010, DPO=0x0000, DPD=0x0040, PD=0x0050) and (PO=0x0010, DPO=0x0000, DPD=0x0000, PD=0x0040) and possible Backward Paths are represented by the tuples (PO=0x0050, DPO=0x0040, DPD=0x0000, PD=0x0010) and (PO=0x0040, DPO=0x0000, DPD=0x0000, PD=0x0010).

If node 1 is the Path Originator, node 2 is the Delegate Path Originator, node 4 is the Delegate Path Destination, and node 5 is the Path Destination, possible Forward Paths are represented by the tuples (PO=0x0010, DPO=0x0020, DPD=0x0040, PD=0x0050), (PO=0x0010, DPO=0x0020, DPD=0x0000, PD=0x0040), (PO=0x0020, DPO=0x0000, DPD=0x0040, PD=0x0050), and (PO=0x0020, DPO=0x0000, DPD=0x0000, PD=0x0040) and possible Backward Paths are represented by the tuples (PO=0x0050, DPO=0x0040, DPD=0x0020, PD=0x0010), (PO=0x0050, DPO=0x0040, DPD=0x0000, PD=0x0020), (PO=0x0040, DPO=0x0000, DPD=0x0020, PD=0x0010), and (PO=0x0040, DPO=0x0000, DPD=0x0000, PD=0x0020).

For each Forward Path to a node that is established through the use of the proactive approach to path establishment (the exchange of NINFO messages), there is no corresponding Backward Path. A Forward Path to node 3 in this category is represented by the tuple (PO=0x0000, DPO=0x0000, DPD=0x0000, PD=0x0030).

##### *3.6.8.4.2 Examples of Transport Control message exchange and updates to the Forwarding Table state*

錯誤! 找不到參照來源。 is an example of the Transport Control message exchange and updates to the Forwarding Table state that are reported during Directed Forwarding Initialization, Directed Forwarding Discovery, and Directed Forwarding Establishment operations between two nodes (A and B). In the illustration, “SL” indicates the Subscription List field of a Forwarding Table entry.

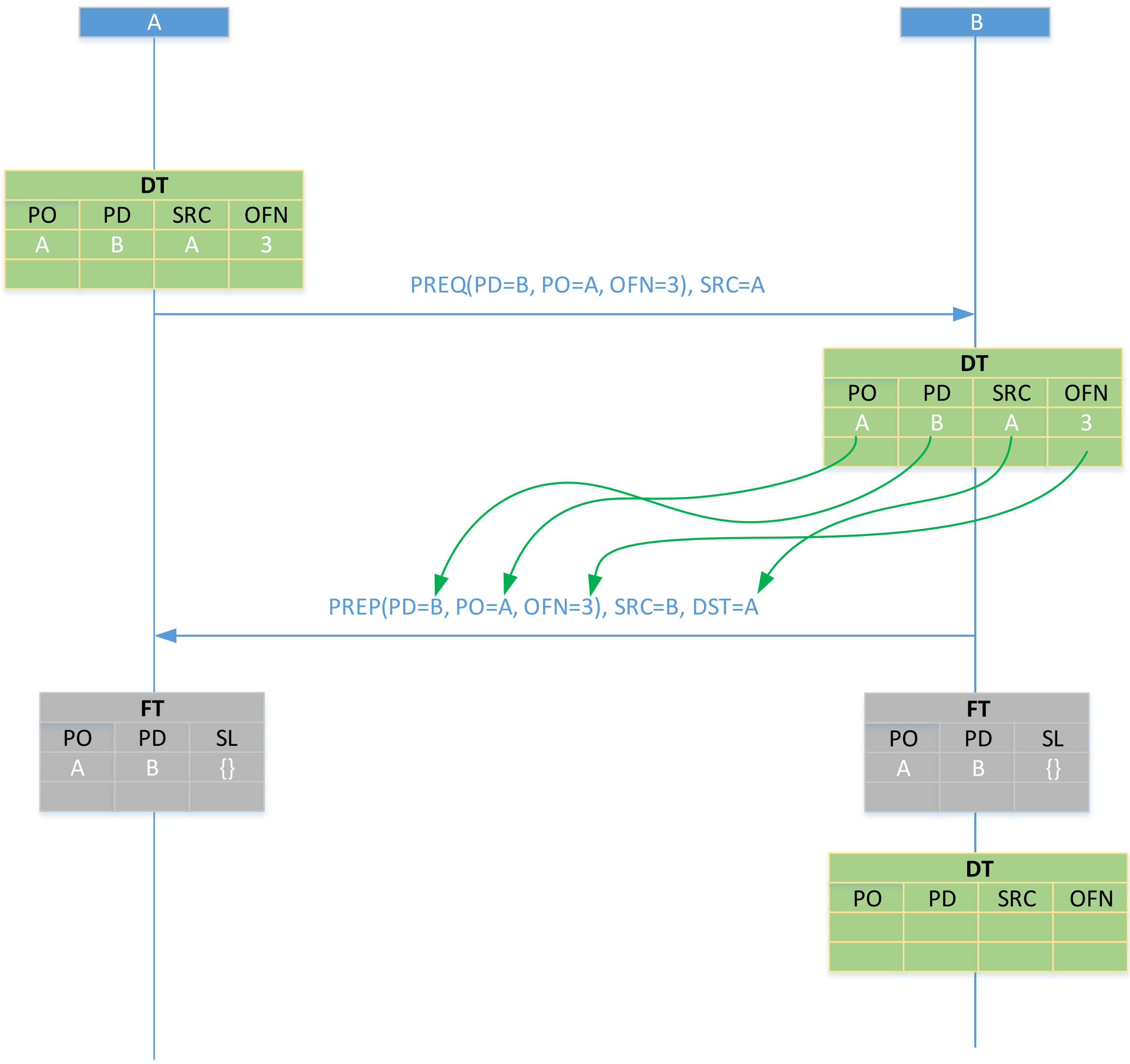


*Figure 3.37: Example of Directed Forwarding operations during the establishment of a path between two nodes*

Similarly, 錯誤! 找不到參照來源。 shows an example of the exchange of Transport Control messages and updates to Forwarding Table states that are reported during Directed Forwarding Initialization, Directed Forwarding Discovery, and Directed Forwarding Establishment operations among a node A and two other nodes (C and D), which are subscribed to a group address G1, via an intermediate Relay B.

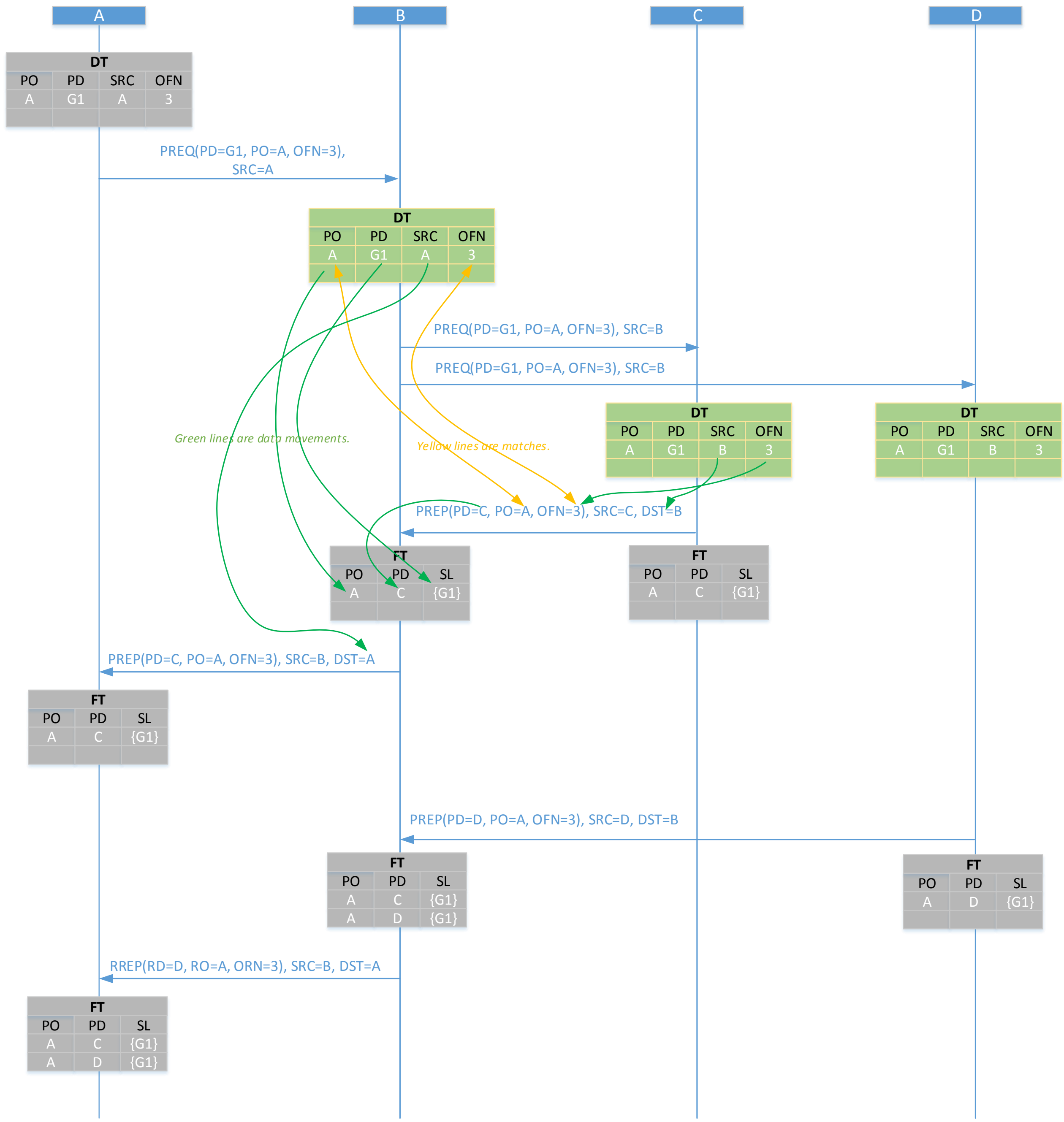


*Figure 3.38: Example of Directed Forwarding operations for the establishment of paths via a Relay node to two nodes that are subscribed to a group address*

#### 3.6.8.5 Discovery Table

Each node manages a Discovery Table containing a list of temporary entries associated with received PREQs and PTREQs. If the node belongs to multiple subnets, then a Discovery Table is maintained for each subnet with information about PREQs and PTREQs received on the associated subnet.

Each entry in a Discovery Table shall contain the following information:

- PO: Primary unicast address of the Path Originator.
- DPO: Primary unicast address of the Delegate Path Originator, if present.
- OFN: Last Originator Forwarding Number known to have been generated by either the Path Originator or the Delegate Path Originator.
- OPM: Path Metric associated with the path toward either the Path Originator or the Delegate Path Originator.
- OAR: Range of secondary addresses known to be assigned to the Path Originator or to the Low Power node associated with the Delegate Path Originator.
- TM: A Tracing Mode flag indicating whether the entry was created upon receiving a PREQ (the flag is set to 0) or upon receiving a PTREQ (the flag is set to 1).
- PD: The unicast address, group address, or virtual address of the Path Destination.
- SRC: Theh primary unicast address of the node selected as the next-hop toward either the Path Originator or the Delegate Path Originator.
- Path Request Expiration: Time until when the Discovery Table entry is considered valid.
- BTO: Index of the bearer from which a PREQ is received, to be used for messages towards the Path Originator.

The Discovery Table initially is empty. It shall be updated whenever a PREQ, PREP, PTREQ, or PTREP message is received and processed.

When a PREQ (see Section 3.6.5.11) is received, the node checks whether a Discovery Table entry that corresponds to the PREQ message exists.

A Discovery Table entry corresponds to the PREQ message if all the following conditions are met:

- The TM value in the entry is 0.
- The PO value in the table entry matches the PO value in the PREQ message and, if the OBOO value in the PREQ message is 1, the DPO value in the PREQ message matches the DPO value in the table entry.
- The OFN value in the table entry matches the OFN value in the PREQ message.

If no existing Discovery Table entry corresponds to the PREQcomm message, a new entry shall be added to the Discovery Table and shall be set as follows:

- PO, OFN, PD, and SRC: Set to the values of the identically named fields in the PREQ message.
- DPO: Set to the DPO value in the PREQ message if OBOO in the PREQ is set to 1; otherwise, set to the unassigned address.
- OPM: Calculated on the basis of the Path Metric Type value in the Path Metric state by using the method defined in Section **錯誤! 找不到參照來源。**.
- OAR: Set to the OAR value in the PREQ message if the EA value in the PREQ is set to 1; set to 0.
- TM: Set to 0.
- Path Request Expiration: Set to the sum of the current time and the Path Request Lifetime value.
- BTO: Set to the index of the bearer from which PREQ is received.

The path associated with the entry is indicated as being discovered.

If a Discovery Table entry that corresponds to the PREQ message exists and the OPM value calculated on the basis of the Path Metric Type value in the Path Metric state, by using the method defined in Section **錯誤! 找不到參照來源。**, is preferred to the OPM value in the entry, according to the selection criterion defined in Section **錯誤! 找不到參照來源。**, then the OPM field in the entry shall be set to the calculated OPM value.

When a PREP (see Section **錯誤! 找不到參照來源。**) is received, the node checks whether a Discovery Table entry that corresponds to the PREP message exists.

The Discovery Table entry that corresponds to the PREP message exists if all the following conditions are met:

- TM in the entry is set to 0.
- OBOO in PREP is set to 0 and PO in PREP is set to the PO value in the entry, or OBOO in PREP is set to 1 and PO in PREP is set to the DPO value in the entry.
- OFN in PREP is set to the OFN value in the entry.

If a Discovery Table entry that corresponds to the PREP message exists, the values of some fields of the entry shall be copied in the Forwarding Table state (see Section 3.6.8.2). If PD in the entry is set to a unicast address, then the entry shall be removed.

When a PTREQ (see Section 3.6.5.13) is received, the node checks whether a Discovery Table entry that corresponds to the PTREQ message exists.

The Discovery Table entry that corresponds to the PTREQ message exists if all the following conditions are met:

- TM in the entry is set to 1.
- PO in PTREQ is set to the PO value in the entry and, if OBOO in PTREQ is set to 1, DPO in PTREQ is set to the DPO value in the entry.

If a Discovery Table entry that corresponds to the PTREQ message does not exist, a new entry shall be added to the Discovery Table and shall be set as follows:

- PO, PD, SRC: Set to the values of the identically named fields in PTREQ.
- DPO: Set to the PO value in PTREQ if OBOO in PTREQ is set to 1; set to the unassigned address otherwise.
- OPM: Set to the OPM value calculated on the basis of the Path Metric Type value in the Path Metric state, by using the method defined in Section **錯誤! 找不到參照來源。**.
- TM: Set to 1.
- Path Request Expiration: Set to the sum of the current time and the Path Request Lifetime value.
- OFN: Set to 0x00 and ignored.
- OAR: Set to 0xFF and ignored.
- BTO: Set to the unassigned bearer index and ignored.

The path associated with the entry is indicated as being traced.

When a PTREP (see Section 3.6.5.14) is received, the node checks whether a Discovery Table entry that corresponds to the PTREP message exists.

The Discovery Table entry that corresponds to the PTREP message exists if all the following conditions are met:

- TM in the entry is set to 1.
- OBOO in PTREP is set to 0 and PO in PTREP is set to the PO value in the entry, or OBOO in PTREP is set to 1 and PO in PTREP is set to the DPO value in the entry.
- PD in PTREP is set to the PD value in the entry.

If a Discovery Table entry that corresponds to the PTREP message exists, the entry shall be removed.

A Discovery Table entry shall be also removed if any of the following conditions is met:

- The current time is greater than the Path Request Expiration value in the entry.
- The TM value in the entry is set to 1 and the entry corresponds to a Forwarding Table entry that is deleted (see Section 3.6.8.2).

### 3.6.8.6 Neighbor Table

Directed Forwarding nodes maintain a Neighbor Table to store information about one-hop neighbors. The Neighbor Table is used during the Directed Forwarding operations to filter PREQ, NINFO, and PTREQ messages (see Section 3.6.8.1) and to initiate a Directed Forwarding Error Notification upon detecting that a link with any one-hop neighbor is broken (see Section **錯誤! 找不到參照來源。**). This section describes the format and management of the Neighbor Table.

Each entry of the Neighbor Table shall contain the following field values:

- Neighbor Address (NA): The primary unicast address of the neighbor.
- Neighbor flag (NF): A flag indicating whether the node identified by NA is considered a neighbor (NF=1) or not (NF=0).
- Retransmission count (RET): A 3-bit value that controls the number of message retransmissions of a Network PDU relayed by the Directed Forwarding node, so that the Network PDU is transmitted a number of times equal to the RET value plus one. By default, the RET value shall be set to 3. The Directed Forwarding node may change the value of this field depending on the link quality. If the link quality is good, a lower value for RET may be used. If the link quality is bad, a higher value for RET may be used.

The Neighbor Table entry may contain the following field values:

- RSSI: The RSSI measured upon receiving messages originated by the neighbor. The value of this field should be calculated as an average of multiple RSSI measurements collected for the same neighbor.
- WantedRSSI: The minimum RSSI value that the neighbor requires when receiving messages.
- Last Updated Timestamp (LUT): A timestamp, either local or absolute, that records the time when the last message was received by the neighbor.

The Neighbor Table is initially empty. It shall be updated when a valid message is received by a neighbor. Only messages with TTL = 0 (for example, NINFO, PREQ, PREP, PCONF, PTREQ, PTREP, PAREQ, PAREP, PERR, LREQ, and LREP messages) or TTL = 0x7F should be used to update the Neighbor Table.

The NF field shall be set to 1 by default. A Directed Forwarding node may set NF to 0 for a neighbor that has poor radio link quality (for example, an RSSI below a defined threshold) in either direction. In such a case, the link is considered broken.

The methods used to decide whether the RSSI is good enough and whether the link is bi-directional are left to implementations. A long term RSSI measurement should be performed over the different advertisement channels to assess the quality of a link. For the Neighbor Table management, the Directed Forwarding node should also know the periodicity of messages exchanged with neighbors. The Directed Forwarding node may use this information to identify if a message has been missed completely during the Neighbor Table entry aging process. Once a Neighbor Table entry is aged, the node either may set NF to 0 or may remove the Neighbor Table entry to preserve the memory.

When any Transport Control message used for a Directed Forwarding operation is received (that is, any of NINFO, PREQ, PREP, PCONF, PTREQ, PTREP, PAREQ, PAREP, and PERR), except for the messages used for link quality measurement (see Section **錯誤! 找不到參照來源。**), and if a Neighbor Table entry associated with the SRC value of the message does not exist after updating the Neighbor Table or exists and the NF field is 0, then the message is discarded.

Note: The Neighbor Table is implemented and updated independently of the value of the Neighbor Information Option state in the node.

#### 3.6.8.7 Directed Forwarding parameters

The following parameters are defined in association with the Directed Forwarding feature. Because they are not sent in Transport Control messages, and they shall be consistent among nodes, these parameters are assigned by the following requirements:

- **Path Reply Delay**: Minimum delay that a node shall wait between the reception of a PREQ message directed to the node and the transmission of a PREP message. The Path Replay Delay shall be set to 500 milliseconds. This delay is introduced to let a node collect multiple PREQ messages coming from different paths and select the one with the best path metric before issuing a PREP message.
- **Path Request Lifetime**: Timeout for Discovery Table entries. The Path Request Lifetime shall be set to 30 seconds.
- **Path Request Delay**: Minimum delay that a node shall wait between the reception of a PREQ message not directed to the node and the generation of a new PREQ message associated with the same discovery. The Path Request Delay shall be set to 150 milliseconds. This delay is introduced to let a node collect multiple PREQ messages coming from different neighboring nodes and calculate the best path metric before issuing a PREQ message associated with the same discovery.
- **Neighbor Information Period**: Interval between periodic NINFO message transmissions. The Neighbor Information Period shall be set to 30 seconds.

#### 3.6.8.8 Coexistence rules

The following rules are defined to enable coexistence of Directed Forwarding nodes and nodes not supporting the Directed Forwarding feature or supporting the feature and with the feature disabled:

- A Directed Forwarding node shall discover a path (that is, transmit a PREQ message) toward a destination if the node does not have information of a valid path toward the destination.

- A Directed Forwarding node may establish a new path toward a destination, replacing an existing path toward the destination.
- A Directed Forwarding node shall use directed security credentials to send a message if a path toward the destination of the message exists.
- A Directed Forwarding node that is a Path Originator or a Delegate Path Originator shall use master security credentials to send access messages and Transport Control messages that are not Directed Forwarding control messages, during the Directed Forwarding Discovery and Directed Forwarding Establishment operations (that is, between the transmission of the PREQ message and the reception of the PREP message).
- A Directed Forwarding node that is a Path Originator or a Delegate Path Originator shall use master security credentials to send access messages and Transport Control messages that are not associated to Directed Forwarding operations if the node does not receive a PREP message within a Path Request Lifetime after sending a PREQ message. The Directed Forwarding node shall continue to send these messages until a new path toward the destination is established.
- A Directed Forwarding node shall use master security credentials to send access messages and Transport Control messages that are not Directed Forwarding control messages to a destination if a PERR message is received that invalidates an existing Forwarding Table entry associated with the destination. The Directed Forwarding node shall continue to send these messages until a new path toward the destination is established.

## 3.7 Access layer

The access layer defines how higher-layer applications can use the upper transport layer. It defines the format of the application data; it defines and controls the application data encryption and decryption performed in the upper transport layer; and it checks whether the incoming application data has been received in the context of the right network and application keys before forwarding it to the higher layer.

### 3.7.1 Endianness

All multiple-octet numeric values in this layer shall be “little endian” as described in Section 3.1.1.2.

### 3.7.2 Model identifier

A model is identified by a unique identifier. The identifier can be 16 bits in length for a Bluetooth SIG adopted model (SIG Model ID) or 32 bits in length for a Vendor Model (Vendor Model ID). See Section 4.4.5 for additional information on SIG Model IDs.

The Vendor Model ID is composed of two values: a 16-bit Bluetooth-assigned Company Identifier [6] and a 16-bit vendor-assigned model identifier. The format of each Vendor Model ID is defined in Table 3.60.

| Field | Size (octets) | Notes |
|---|---|---|
| 16-bit Company Identifier | 2 | See [6] |

| | | |
|---|---|---|
| 16-bit vendor-assigned model identifier | 2 | |

*Table 3.60: Vendor Model ID format*

### 3.7.3 Access payload

The access payload is logically composed of the fields shown in Table 3.61.

| Field Name | Size (octets) | Notes |
|---|---|---|
| Opcode | 1, 2, or 3 | Operation Code |
| Parameters | 0 to 379 | Application Parameters |

*Table 3.61: Access payload fields*

An access payload may be sent in up to 32 segments of 12 octets each. This implies that the maximum number of octets is 384 including the TransMIC.

With a 4-octet TransMIC, the maximum size of the access payload is 380 octets, and therefore with a single octet opcode, the parameters field can be up to 379 octets. With a 2-octet opcode, the parameters field can be up to 378 octets. With a 3-octet opcode, the parameters field can be up to 377 octets.

The lower transport layer may segment messages into smaller PDUs for delivery over the network layer. Table 3.62 shows the maximum useful application packet size depending on the number of packets and the size of the TransMIC.

| Number of Packets | Maximum useful access payload size (octets) | |
|---|---|---|
| | 32 bit TransMIC | 64 bit TransMIC |
| 1 | 11 (unsegmented) | n/a |
| 1 | 8 (segmented) | 4 (segmented) |
| 2 | 20 | 16 |
| 3 | 32 | 28 |
| n | (n×12)-4 | (n×12)-8 |
| 32 | 380 | 376 |

*Table 3.62: Maximum useful access payload sizes for various sizes of TransMIC*

#### 3.7.3.1 Operation codes

An operation code (opcode) is an array of octets comprising 1, 2, or 3 octets. The first octet of the opcode determines the number of octets that are part of the opcode.

If the most significant bit of the first octet of the opcode is zero, then the opcode contains a single octet. If the two most significant bits of the first octet are 0b10, then the opcode contains two octets. If the two most significant bits of the first octet are 0b11, then the opcode contains three octets. This is shown in Table 3.63.

| Opcode Format | Notes |
|---|---|
| 0xxxxxxx (excluding 01111111) | 1-octet Opcodes |
| 01111111 | Reserved for Future Use |
| 10xxxxxx xxxxxxxx | 2-octet Opcodes |
| 11xxxxxx zzzzzzzz | 3-octet Opcodes |

*Table 3.63: Opcode formats*

The 1-octet opcodes are used for Bluetooth SIG defined application opcodes. There are 127 1-octet opcodes that can be defined and allocated by the Bluetooth SIG. Opcode 0x7F is reserved for future possible extension.

The 2-octet opcodes are used for Bluetooth SIG defined application opcodes. There are 16384 2-octet opcodes that can be defined and allocated by the Bluetooth SIG.

The 3-octet opcodes are used for manufacturer-specific opcodes. There are 64 3-octet opcodes available per company identifier, identified using ”x” in Table 3.63, although a company may further sub-class opcodes if desired. The company identifiers are 16-bit values defined by the Bluetooth SIG and are coded into the second and third octets of the 3-octet opcodes, identified using ”z” in Table 3.63. The company-specific opcodes are managed by the company associated with the identifier.

#### 3.7.3.2 Application parameters

The Parameters field is defined individually for each opcode. The specific parameters are defined within the message definitions in Section 4.3. The Parameters field can be zero octets in length.

### 3.7.4 Access layer behavior

#### 3.7.4.1 Transmitting an access message

A message is transmitted by a model to a destination address that is a unicast address, a group address, or a virtual address.

A message is transmitted from a source address, which is the transmitting element’s unicast address.

The TTL field may be specified by the application by setting it to the number of hops required to transfer the message from the source element to all the destination addresses. However, if this is not specified, the Default TTL shall be applied by the access layer.

The SRC field shall be set to the unicast address of the element within the node that is originating the message.

The DST field shall be set to the unicast address, group address, or virtual address that the message is directed toward.

If the Directed Forwarding feature is supported and enabled in the subnet over which the message is transmitted, a node checks whether a Forwarding Table entry associated with the Path Destination exists (see Section 3.6.8.2). If the entry exists, the message shall be transmitted using the directed security credentials. If no entry exists, the node shall initiate a Directed Forwarding Discovery operation for the Path Destination, and may use the master security credentials while the Directed Forwarding Discovery operation is ongoing.

The access layer does not guarantee delivery of messages. Each model should decide if a message is to be retransmitted and how potential duplicates are handled.

If the message is sent in response to a received message that was sent to a unicast address, the node should transmit the response message with a random delay between 20 and 50 milliseconds. If the message is sent in response to a received message that was sent to a group address or a virtual address, the node should transmit the response message with a random delay between 20 and 500 milliseconds. This reduces the probability of multiple nodes responding to this message at exactly the same time, and therefore increases the probability of message delivery rather than message collisions.

Due to limited bandwidth available that is shared among all nodes and other Bluetooth devices, it is important to observe the volume of traffic a node is originating. A node should originate less than 100 Lower Transport PDUs in a moving 10-second window.

#### 3.7.4.2 Receiving an access message

A message is delivered to the model for processing if all the following conditions are met:

- The opcode belongs to the addressed model’s element.
- The destination address is set to one of the model’s element unicast address or a group or virtual address for which the model’s element is subscribed to, or the destination address is set to a fixed group address of the primary element of the node as defined in Section 3.4.2.4.
- The model is bound to the application or device key that was used to secure the transportation of the message.

#### 3.7.4.3 Security considerations

A message is encrypted and authenticated by the upper transport layer. Messages originated by a node shall use either the AppKey configured for the Model or the DevKey.

A response message shall always use the same DevKey or AppKey used by the corresponding request message.

#### 3.7.4.4 Message error procedure

When receiving a message that is not understood by an element, it shall ignore the message.

> Note: A message can be falsely identified as a valid message, passing the NetMIC and TransMIC authentication using a known network key and application key even though that message was sent using different keys. The decryption of that message using the wrong keys would result in a message that is not understood by the element. The probability of such a situation occurring is small but not insignificant.

A message that is not understood includes messages that have one or more of the following conditions:

- The application opcode is unknown by the receiving element.
- The access message size for the application opcode is incorrect.
- The application parameters contain values that are currently Prohibited.

> Note: An element that sends an acknowledged message that is not understood by a peer node will not receive any response message.

### 3.7.5 Unacknowledged and acknowledged messages

At the access layer, a message is defined as unacknowledged or acknowledged.

#### 3.7.5.1 Unacknowledged message

An unacknowledged message is transmitted whenever an application decides peer nodes should be notified about a state change using a status message. A status message is sent to the publish address of a model associated with the changed state. There is no response to an unacknowledged message, therefore it is not possible for the sending element to determine if that message has been delivered or processed.

#### 3.7.5.2 Acknowledged message

An acknowledged message is transmitted and acknowledged by each receiving element by responding to that message. The response is typically a status message.

If a response is not received within an arbitrary time period, the message originator may retransmit the message. The time period used is application specific.

If an element transmits a message to more than one element, for example it has set the destination address to a group address, the element may not know how many elements may respond to the message. It is not recommended to send an acknowledged message to the all-nodes address. To increase the probability of successful delivery of messages, the sending element should determine how many message retransmissions are required before it considers that all the nodes that should have received the message have actually received it.

If the element does not receive a response within a period of time known as the acknowledged message timeout, then the element may consider the message has not been delivered, without sending any additional messages.

The acknowledged message timeout should be set to a minimum of 30 seconds. The exact value is application specific.

When an acknowledged message is delivered to the model, it shall send the associated response message to acknowledge that message. The response message may include information such as state information. The response message is an unacknowledged message. The destination of the response message shall be set to the source address of the received acknowledged message. If the acknowledged message has TTL set to 0, it is recommended that the response message should have TTL set to 0.

### 3.7.6 Publish and subscribe

A higher layer specification can describe messages containing data as being published by a model or subscribed to by a model's element.

Publishing and subscribing is performed using destination addresses.

The configuration of the destination addresses used for publishing and subscribing is managed using a higher layer specification.

#### 3.7.6.1 Publish

A model publishes data if it transmits an unsolicited message to a destination address. Messages can be transmitted to destination addresses that can be unicast, group, or virtual, known as the publish addresses. Each model within a node has a single publish address.

##### *3.7.6.1.1 State transitions*

States within an element either can change instantaneously or can transition over time to a new state, as illustrated in Figure 3.39. The time from the initial state to the target state is the transition time. The time from the current state to the target state is the remaining time. When a message is received to set a new state value, this new value may not be immediately applied, but may instead be stored as a target state. The state will transition from the initial state to the target state. A status message can be sent at any time and will always include the current state even if the transition time has not elapsed. This status message may include the remaining time between the current state and the target state. When the current state reaches the target state, the state transition ends.

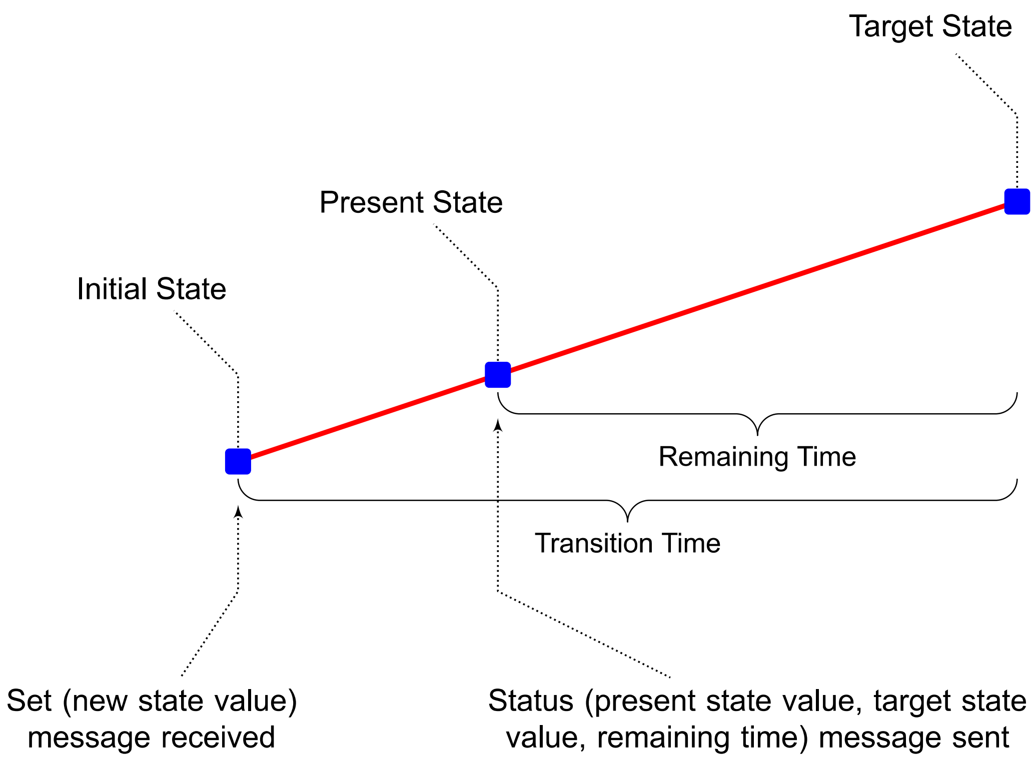


*Figure 3.39: State transition*

#### 3.7.6.1.2 State change publishing

Publishing a message on a state change is enabled by setting a Model Publication state (see Section 4.2.2) for a model. When publishing is enabled for a model, a corresponding Status message shall be published immediately after the state transition ends. For transitions that take more than 2 seconds, it is recommended to publish an additional Status message within 1 second from the start of the state transition.

#### 3.7.6.1.3 Periodic publishing

A model may be configured to send Status messages periodically regardless of whether the state has changed or not. This is done by using a Publish Period (see Section 4.2.2.2). When the Publish Period is set to a non-zero value, a Status message shall be published at least once every Publish Period. When the Publish Period is set to 0, the Status messages shall only be published on a state change when enabled.

### 3.7.6.2 Subscribe

Each model may have one or more subscription lists (see Section 4.2.3) of one or more addresses that determines which destination addresses this model's element subscribes to. These subscription addresses can be a group address or a virtual address.

### 3.7.7 Example message sequence charts

This section shows typical message sequence charts.

#### 3.7.7.1 Acknowledged Get

Figure 3.40 shows a client getting a state of a peer element using an acknowledged Get message. The server responds with the associated Status message.

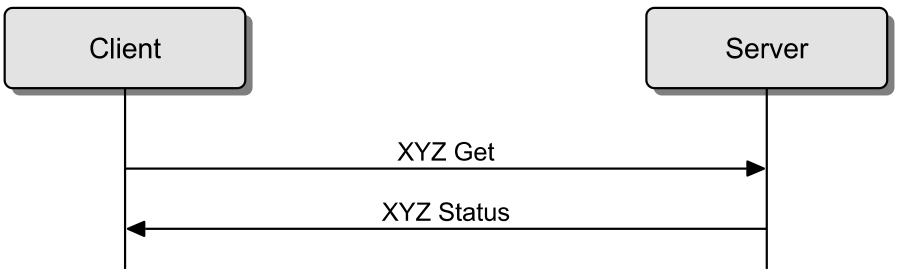


*Figure 3.40: Acknowledged Get message*

#### 3.7.7.2 Acknowledged Set

Figure 3.41 shows a client setting a state of a peer element with an acknowledged Set message. The server responds with the associated Status message. The server then publishes a Status message to the model's publish address according to the publishing rules (see Section 3.7.6.1.2). If the client is subscribed to this model's publish address, then it will receive both Status messages.

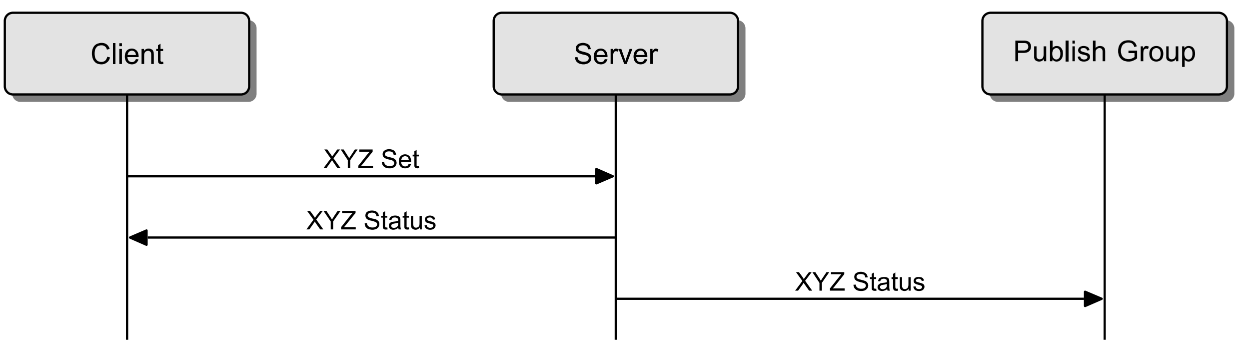


*Figure 3.41: Acknowledged Set message*

#### 3.7.7.3 Unacknowledged Set

Figure 3.42 shows a client setting a state of a peer element with an unacknowledged Set message. No response is sent, but the server publishes the new state information to the model's publish address according to the publishing rules (see Section 3.7.6.1.2). If the client is subscribed to this model's publish address, then it will receive the Status message.

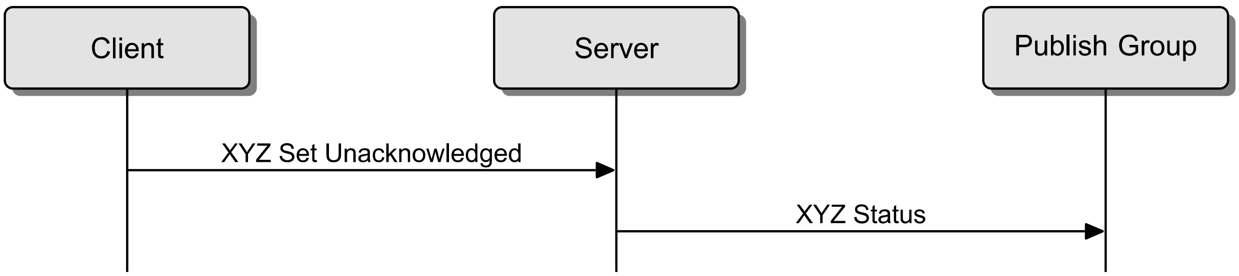


*Figure 3.42: Unacknowledged Set message*

#### 3.7.7.4 Acknowledged set with periodic publishing

Figure 3.43 shows a client setting a state of a peer element with an acknowledged Set message. The server responds with the associated Status message. The server then periodically publishes the new state information to the model's publish address, according to the periodic publishing rules (see Section 3.7.6.1.2).

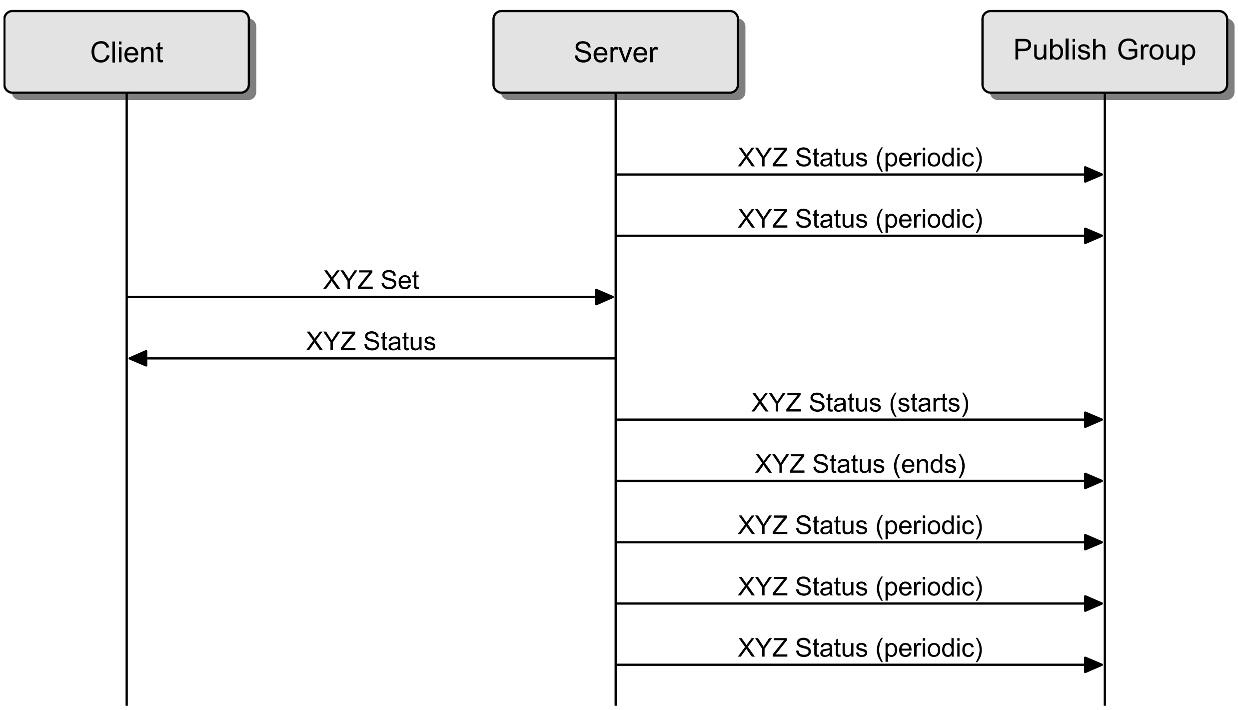


*Figure 3.43: Acknowledged Set message with periodic publishing*

## 3.8 Mesh security

This section describes how mesh security is implemented.

### 3.8.1 Endianness

All multiple-octet numeric values in this layer shall be marshalled in “big endian,”, as described in Section 3.1.1.1.

### 3.8.2 Security toolbox

This section describes the four functions that together provide a security toolbox for mesh networking.

#### 3.8.2.1 Encryption function

The same encryption function *e*, as defined in Volume 3, Part H, Section 2.2.1 of the Core Specification [1], shall be used. This can be summarized as:

ciphertext = e(key, plaintext)

#### 3.8.2.2 CMAC function

RFC4493 [9] defines the Cipher-based Message Authentication Code (CMAC) that uses AES-128 as the block cipher function, also known as AES-CMAC. The inputs to AES-CMAC are:

k is the 128-bit key

m is the variable length data to be authenticated

The 128-bit message authentication code (MAC) is generated[1] as follows:

$\text{MAC} = \text{AES-CMAC}_k(m)$

A node can implement AES functions in the host or can use the HCI_LE_Encrypt command (see Volume 2, Part E, Section 7.8.22 of the Core Specification [1]) in order to use the AES function in the controller.

[1] This is using the same notation used in other Bluetooth specifications. This is functionally the same as the notation as RFC 4493, where MAC = AES-CMAC(k, m).

#### 3.8.2.3 CCM function

RFC3610 [10] defines the AES Counter with CBC-MAC (CCM) (see Volume 6, Part E, Section 1 of the Core Specification [1]). This specification defines AES-CCM as a function that takes four inputs and results in two outputs:

The inputs to AES-CCM are:

k is the 128-bit key

n is a 104-bit nonce

m is the variable length data to be encrypted and authenticated – also known as “plaintext”

a is the variable length data to be authenticated – also known as “Additional Data”

The ciphertext and mic are generated as follows:

ciphertext, mic = $\text{AES-CCM}_k$ (n, m, a)

Where:

ciphertext is the variable length data after it has been encrypted

mic is the message integrity check value of m and a – also known as the “Message Authentication Code” or the encrypted authentication value U in RFC3610 [10].

If only the k, n, and m parameters are provided to the AES-CCM, then the additional data shall be zero length.

#### 3.8.2.4 s1 SALT generation function

The inputs to function s1 are:

M is a non-zero length octet array or ASCII encoded string

If M is an ASCII encoded string, it shall be converted into an octet array by replacing each string character with its ASCII code preserving the order. For example, if M is the string “MESH”, this is converted into the octet array: 0x4d, 0x45, 0x53, 0x48.

ZERO is the 128-bit value:

0x0000 0000 0000 0000 0000 0000 0000 0000

The output of the salt generation function s1 is as follows:

s1(M) = $\text{AES-CMAC}_{\text{ZERO}}$ (M)

### 3.8.2.5 k1 derivation function

The network key material derivation function k1 is used to generate instances of IdentityKey and BeaconKey.

The definition of this key generation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k1 are:

N is 0 or more octets

SALT is 128 bits

P is 0 or more octets

The key (T) is computed as follows:

T = AES-CMAC$_{SALT}$ (N)

The output of the key generation function k1 is as follows:

k1(N, SALT, P) = AES-CMAC$_T$ (P)

### 3.8.2.6 k2 network key material derivation function

The network key material derivation function k2 is used to generate instances of EncryptionKey, PrivacyKey, and NID for use as Master and Private Low Power node communication.

The definition of this key generation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k2 are:

N is 128 bits

P is 1 or more octets

The key (T) is computed as follows:

T = AES-CMAC$_{SALT}$ (N)

SALT is the 128-bit value computed as follows

SALT = s1(“smk2”)

The output of the key generation function k2 is as follows:

T0 = empty string (zero length)
T1 = AES-CMAC$_T$ (T0 || P || 0x01)
T2 = AES-CMAC$_T$ (T1 || P || 0x02)
T3 = AES-CMAC$_T$ (T2 || P || 0x03)

$k2(N, P) = (T1 || T2 || T3) \bmod 2^{263}$

#### 3.8.2.7 k3 derivation function

The derivation function k3 is used to generate a public value of 64 bits derived from a private key.

The definition of this derivation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k3 are:

N is 128 bits

The key (T) is computed as follows:

$T = \text{AES-CMAC}_{SALT} (N)$

SALT is a 128-bit value computed as follows:

SALT = s1(“smk3”)

The output of the derivation function k3 is as follows:

$k3(N) = \text{AES-CMAC}_T ( \text{“id64”} || 0x01 ) \bmod 2^{64}$

#### 3.8.2.8 k4 derivation function

The derivation function k4 is used to generate a public value of 6 bits derived from a private key.

The definition of this derivation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k4 are:

N is 128 bits

The key (T) is computed as follows:

$T = \text{AES-CMAC}_{SALT} (N)$

SALT is a 128-bit value computed as follows:

SALT = s1(“smk4”)

The output of the derivation function k4 is as follows:

$K4(N) = \text{AES-CMAC}_T ( \text{“id6”} || 0x01 ) \bmod 2^{6}$

### 3.8.3 Sequence number

The sequence number, a 24-bit value contained in the SEQ field of the Network PDU, is primarily designed to protect against replay attacks. Elements within the same node may or may not share the sequence number space with each other. Having a different sequence number in each new Network PDU for every message source (identified by the unicast address contained in the SRC field) is critical for the security of the mesh network.

With a 24-bit sequence number, an element can transmit 16,777,216 messages before repeating a nonce. If an element transmits a message on average once every five seconds (representing a fairly high frequency message for known use cases), the element can transmit for 2.6 years before the nonce repeats.

Each element shall use strictly increasing sequence numbers for the Network PDUs it generates. Before the sequence number approaches the maximum value (0xFFFFFF), the element shall update the IV Index using the IV Update procedure (see Section 3.10.5). This is done to ensure that the sequence number will never wrap around.

### 3.8.4 IV Index

The IV Index is a 32-bit value that is a shared network resource (i.e., all nodes in a mesh network share the same value of the IV Index and use it for all subnets they belong to).

The IV Index starts at 0x00000000 and is incremented during the IV Update procedure as described in Section 3.10.5. The timing when the value is incremented does not have to be exact, since the least significant bit is communicated within every Network PDU. Since the IV Index value is a 32-bit value, a mesh network can function approximately 5 trillion years before the IV Index will wrap.

The IV Index is shared within a network via Secure Network beacons (see Section 3.9.3). IV updates received on a subnet are processed and propagated to that subnet. The propagation happens by the device transmitting Secure Network beacons with the updated IV Index for that particular subnet. If a device on a primary subnet receives an update on the primary subnet, it shall propagate the IV update to all other subnets. If a device on a primary subnet receives an IV update on any other subnet, the update shall be ignored.

If a node is absent from a mesh network for a period of time, it can scan for Secure Network beacons (see Section 3.10.1) or use the IV Index Recovery procedure (see Section 3.10.6), and therefore set the IV Index value autonomously.

### 3.8.5 Nonce

The nonce is a 13-octet value that is unique for each new message encryption. There are four different nonces that are used, as shown in Table 3.64. The type of the nonce is determined by the first octet of the nonce, referred to as the Nonce Type.

| Nonce Type | Nonce | Description |
|---|---|---|
| 0x00 | Network nonce | Used with an encryption key for network authentication and encryption |
| 0x01 | Application nonce | Used with an application key for upper transport authentication and |

| | | |
|---|---|---|
| | | encryption |
| 0x02 | Device nonce | Used with a device key for upper transport authentication and encryption |
| 0x03 | Proxy nonce | Used with an encryption key for proxy authentication and encryption |
| 0x04–0xFF | RFU | Reserved for Future Use |

*Table 3.64: Nonce types*

Note: The TTL is used within the network nonce but not within the application nonce, device nonce, or proxy nonce. This means that when a message is relayed and the TTL is decremented, the application nonce or device nonce does not change; however, the network nonce does change, allowing the authentication of the TTL value.

Note: The DST is used within the application nonce and device nonce but not in the network nonce. This means that the destination of the application or device message may be authenticated, but at the network layer the destination address is encrypted.

#### 3.8.5.1 Network nonce

The network nonce is defined in Table 3.65 and illustrated in Figure 3.44.

| **Field** | **Size (octets)** | **Notes** |
|---|---|---|
| Nonce Type | 1 | 0x00 |
| CTL and TTL | 1 | See Table 3.66 |
| SEQ | 3 | Sequence Number |
| SRC | 2 | Source Address |
| Pad | 2 | 0x0000 |
| IV Index | 4 | IV Index |

*Table 3.65: Network nonce format*

| **Field** | **Size (bits)** | **Notes** |
|---|---|---|
| CTL | 1 | See Section 3.4.4.3 |
| TTL | 7 | See Section 3.4.4.4 |

*Table 3.66: CTL and TTL field format*

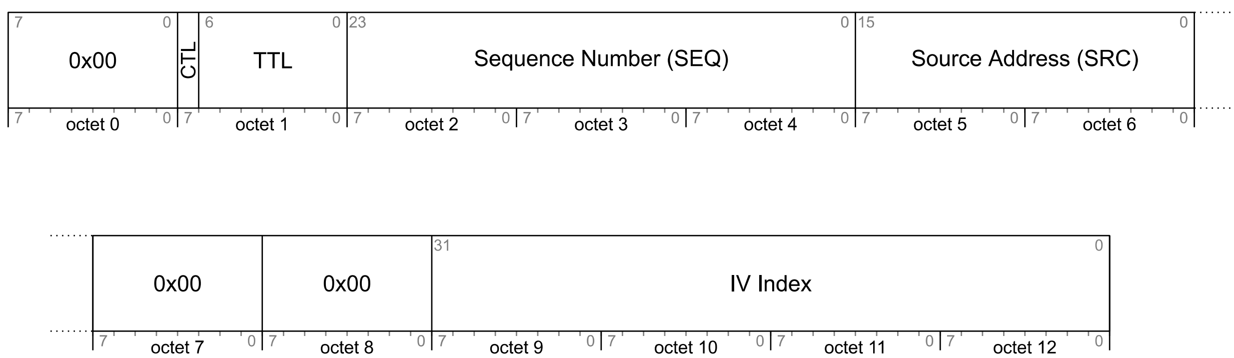


*Figure 3.44: Network nonce format*

The network nonce is used with an encryption key for network data authentication and encryption (see Section 3.8.7.2).

### 3.8.5.2 Application nonce

The application nonce is defined in Table 3.67 and illustrated in Figure 3.45.

| Field | Size (octets) | Notes |
|---|---|---|
| Nonce Type | 1 | 0x01 |
| ASZMIC and Pad | 1 | See Table 3.68 |
| SEQ | 3 | Sequence Number of the Access message (24 lowest bits of SeqAuth in the context of segmented messages) |
| SRC | 2 | Source Address |
| DST | 2 | Destination Address |
| IV Index | 4 | IV Index |

*Table 3.67: Application nonce format*

| Field | Size (bits) | Notes |
|---|---|---|
| ASZMIC | 1 | SZMIC if a Segmented Access message or 0 for all other message formats |
| Pad | 7 | 0b0000000 |

*Table 3.68: ASZMIC and Pad field format*

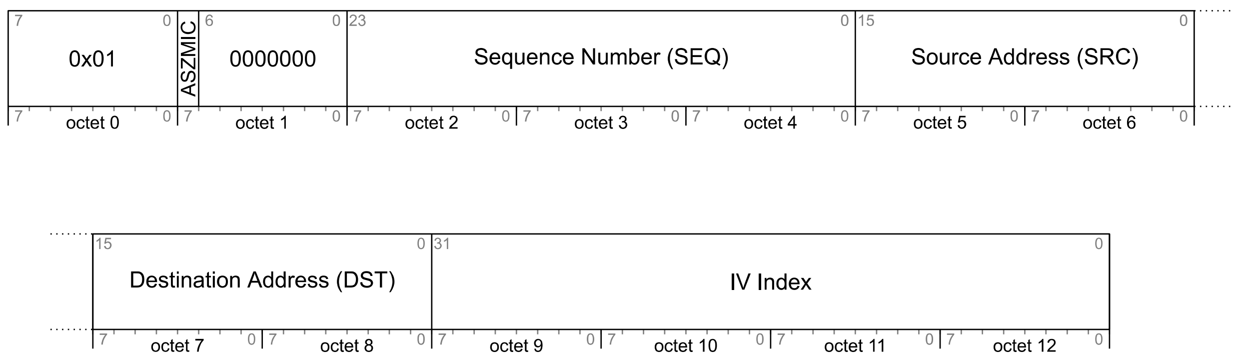


*Figure 3.45: Application nonce format*

The application nonce is used with an application key for application data authentication and encryption (see Section 3.8.6).

### 3.8.5.3 Device nonce

The device nonce is defined in Table 3.69 and illustrated in Figure 3.46.

| Field | Size (octets) | Notes |
|---|---|---|
| Nonce Type | 1 | 0x02 |
| ASZMIC and Pad | 1 | See Table 3.70 |
| SEQ | 3 | Sequence Number of the Access message (24 lowest bits of SeqAuth in the context of segmented messages) |
| SRC | 2 | Source Address |
| DST | 2 | Destination Address |
| IV Index | 4 | IV Index |

*Table 3.69: Device nonce format*

| Field | Size (bits) | Notes |
|---|---|---|
| ASZMIC | 1 | SZMIC if a Segmented Access message or 0 for all other message formats |
| Pad | 7 | 0b0000000 |

*Table 3.70: ASZMIC and Pad field format*

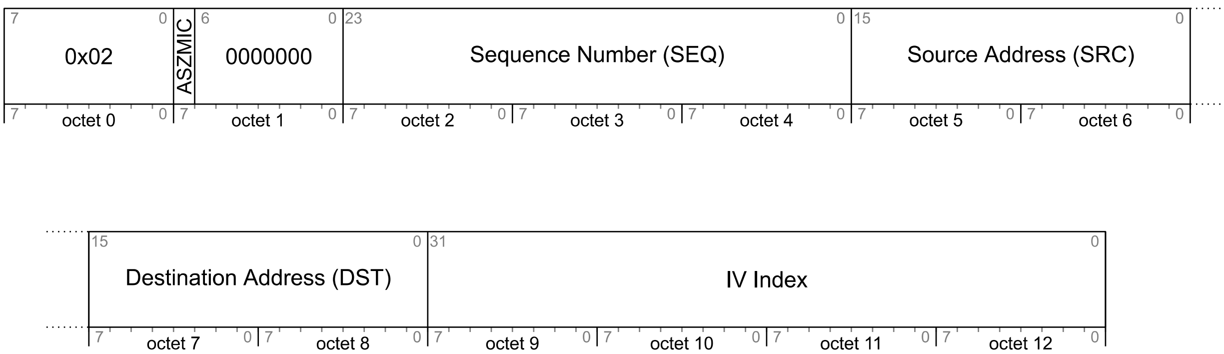


*Figure 3.46: Device nonce format*

The device nonce is used with a device key for application data authentication and encryption specific for a given device (see Section 3.8.6).

### 3.8.5.4 Proxy nonce

The proxy nonce is defined in Table 3.71 and illustrated in Figure 3.47.

| Field | Size (octets) | Notes |
|---|---|---|
| Nonce Type | 1 | 0x03 |
| Pad | 1 | 0x00 |
| SEQ | 3 | Sequence Number |
| SRC | 2 | Source Address |
| Pad | 2 | 0x0000 |
| IV Index | 4 | IV Index |

*Table 3.71: Proxy nonce format*

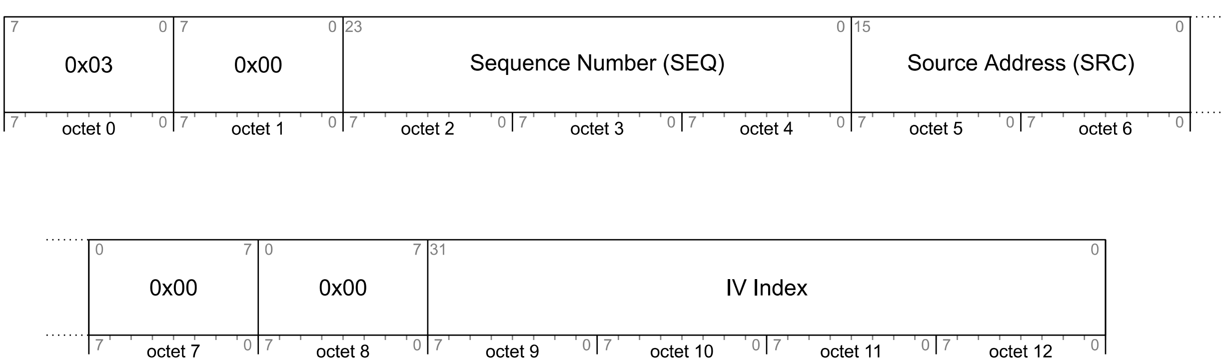


*Figure 3.47: Proxy nonce format*

The proxy nonce is used with an encryption key for proxy configuration message authentication and encryption (see Section 6.5).

### 3.8.6 Keys

The Mesh Profile specification defines two types of keys: application keys (AppKey) and network keys (NetKey). AppKeys are used to secure communications at the upper transport layer and NetKeys are used to secure communications at the network layer. Both types of keys are shared between nodes. There is also a device key (DevKey), which is a special application key that is unique to each node, is known only to the node and a Configuration Client, and is used to secure communications between the node and a Configuration Client.

Application keys are bound to network keys. This means application keys are only used in a context of a network key they are bound to. An application key shall only be bound to a single network key. A device key is implicitly bound to all network keys.

An example of binding application keys to network keys and models is illustrated in Figure 3.48.

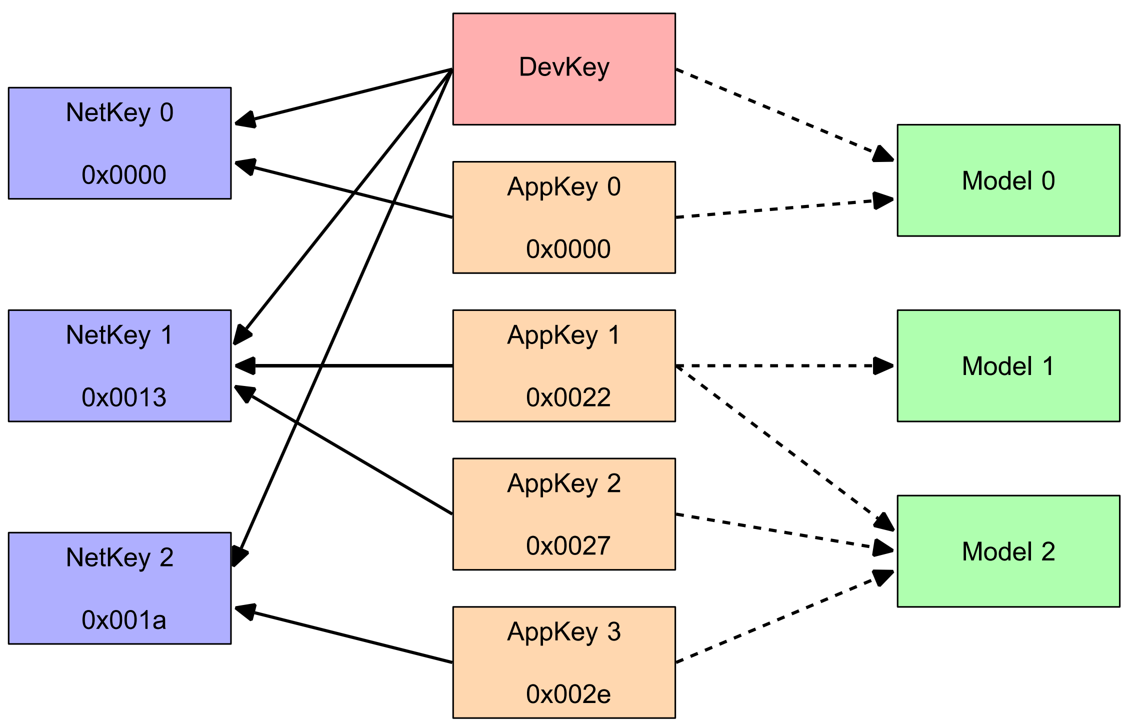


*Figure 3.48: Application key binding example*

#### 3.8.6.1 Device key

The device key (DevKey) is an access layer key known only to the node and a Configuration Client. The device key shall be bound to every network key known to the node. Those bindings cannot be changed. An illustration of the device key derivation is shown in Figure 3.49.

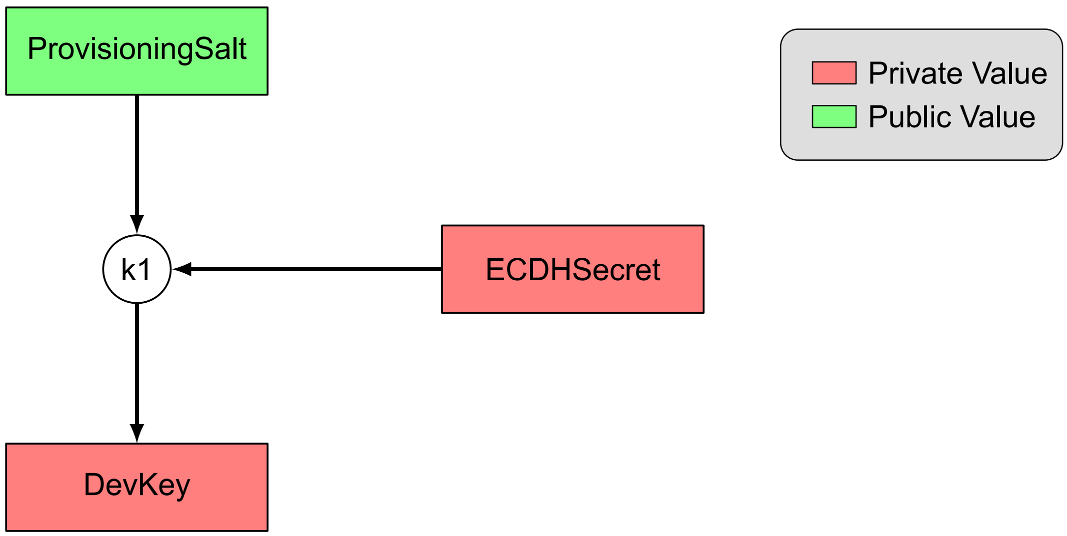


*Figure 3.49: Device key derivation*

The DevKey shall be derived from the ECDHSecret and ProvisioningSalt as described by the formula below:

DevKey = k1(ECDHSecret, ProvisioningSalt, “prdk”)

The ProvisioningSalt is defined in Section 5.4.2.5 and the ECDHSecret is defined in Section 5.4.2.3.

#### 3.8.6.2 Application key

The application key (AppKey) shall be generated using a random number generator compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1].

The application key identifier (AID) is used to identify the application key. An illustration of the AID derivation is shown in Figure 3.50.

AID= k4(AppKey)

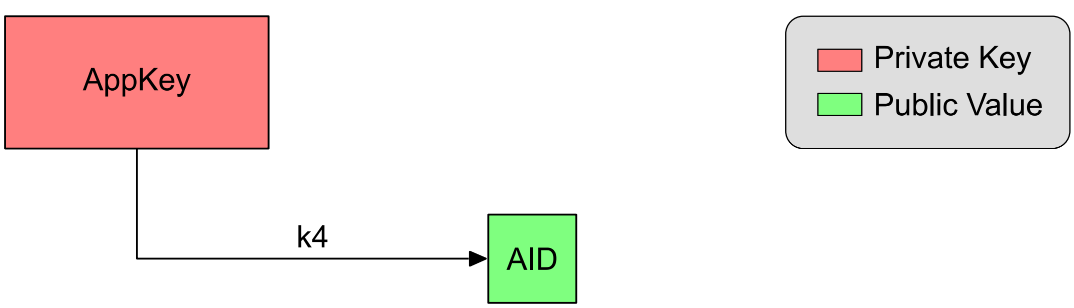


*Figure 3.50: AID derivation*

### 3.8.6.3 Network key

The network key (NetKey) shall be generated using a random number generator compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1]. An illustration of the network key hierarchy is shown in Figure 3.51.

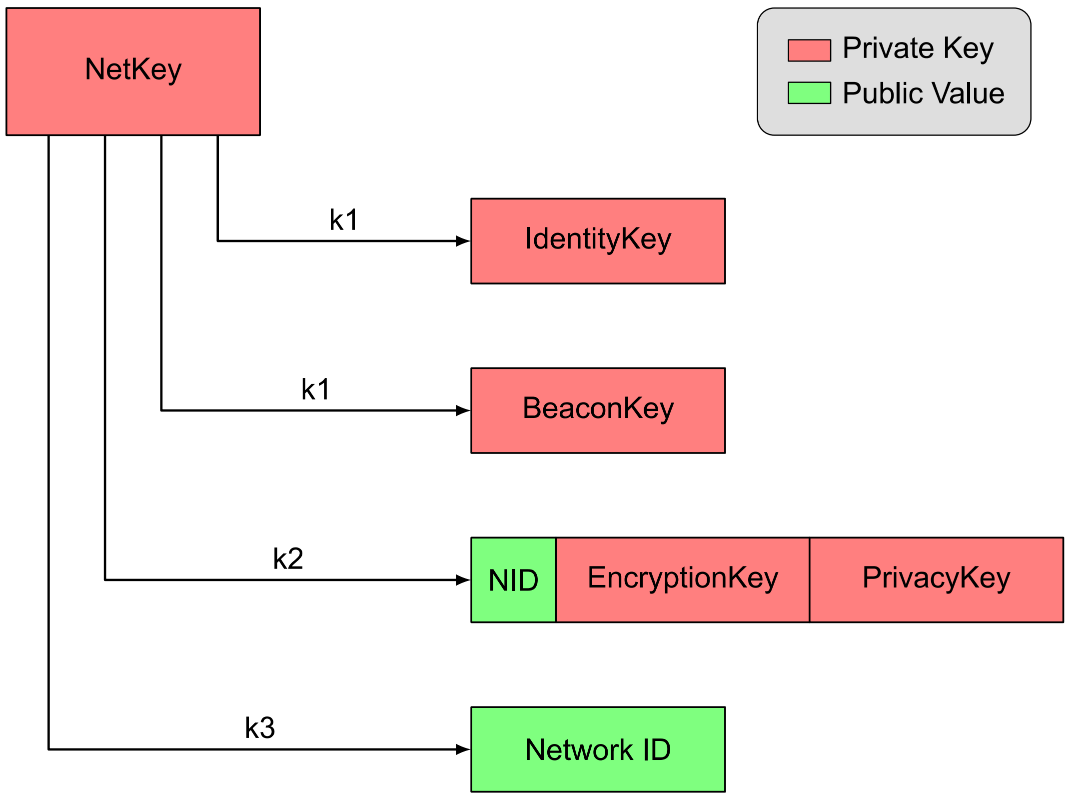


*Figure 3.51: Network key hierarchy*

#### 3.8.6.3.1 NID, Encryption Key, and Privacy Key

Each Network PDU is secured using security material that is composed of the NID, the Encryption Key, and the Privacy Key.

The NID is a 7-bit value that identifies the security material that is used to secure this Network PDU.

Note: There are up to 2121 possible keys for each NID; therefore, the NID value can only provide an indication of the security material that has been used to secure this Network PDU.

The NID, EncryptionKey, and PrivacyKey are derived using the k2 function with security credentials as inputs.

The master security material is derived from the master security credentials using:

NID || EncryptionKey || PrivacyKey = k2(NetKey, 0x00)

The friendship security material is derived from the friendship security credentials using:

NID || EncryptionKey || PrivacyKey = k2(NetKey, 0x01 || LPNAddress || FriendAddress || LPNCounter || FriendCounter)

Where:

The LPNAddress value is the unicast address set as source address in the Friend Request message that set up the friendship.

The FriendAddress value is the unicast address set as source address in the Friend Offer message that set up the friendship.

The LPNCounter value is the value from the LPNCounter field of the Friend Request message that set up the friendship.

The FriendCounter is the value from the FriendCounter field of the Friend Offer message that set up the friendship.

For Network PDUs that are sent between a Low Power node and Friend node that have a friendship relationship, the friendship security material is used.

The directed security material is derived from the directed security credentials using:

NID || EncryptionKey || PrivacyKey = k2(NetKey, 0x02)

For Network PDUs that are forwarded according to the Directed Forwarding feature, the directed security material is used.

For all other Network PDUs, the master security material is used.

#### *3.8.6.3.2 Network ID*

The Network ID is derived from the network key such that each network key generates one Network ID. This identifier becomes public information.

Network ID = k3 (NetKey)

#### *3.8.6.3.3 IdentityKey*

The IdentityKey is derived from the network key such that each network key generates one IdentityKey.

salt = s1(“nkik”)

P = “id128” || 0x01

IdentityKey = k1 (NetKey, salt, P)

#### 3.8.6.3.4 BeaconKey

The BeaconKey is derived from the network key such that each network key generates one BeaconKey.

salt = s1(“nkbk”)

P = “id128” || 0x01

BeaconKey = k1 (NetKey, salt, P)

### 3.8.6.4 Global key indexes

Network and application keys are organized within the mesh network into two lists, maintained by a Configuration Client: a list of network keys and a list of application keys. Each list is a shared mesh network resource and can accommodate up to 4096 keys. Keys are referenced using global key indexes: the NetKey Index and the AppKey Index. The key indexes are 12-bit values ranging from 0x000 to 0xFFF inclusive. A network key at index 0x000 is called the primary NetKey.

## 3.8.7 Message security

Messages are secured using AES-CCM at two different layers. Messages are encrypted and authenticated at the network layer and at the upper transport layer. Each message is also obfuscated to hide possible identifying information from the packets. This is illustrated in Figure 3.52.

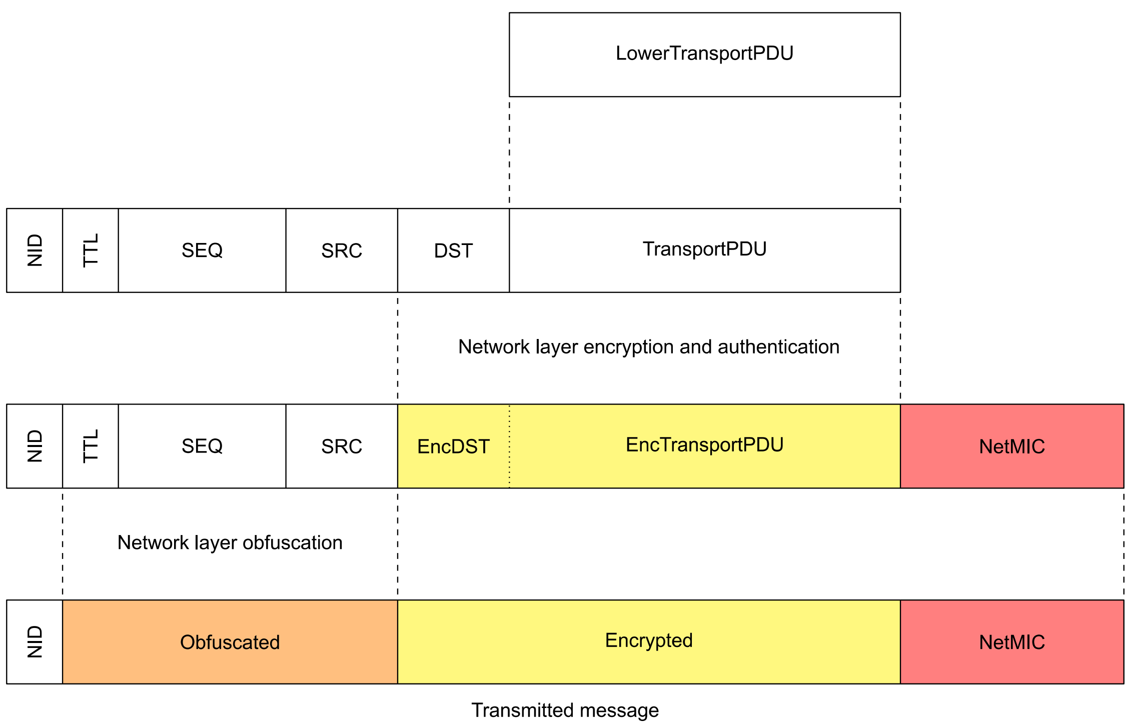


*Figure 3.52: Example of network layer encryption, authentication, and obfuscation*

Every message has a minimum of 64 bits of authentication information associated with it. This authentication information may be split between the network layer and upper transport layer.

Some messages, known as control messages, are not authenticated at the upper transport layer and therefore have a 64-bit NetMIC. Access messages are authenticated at the upper transport layer and therefore have a 32-bit NetMIC. Access messages that are sent in a single unsegmented message have a 32-bit TransMIC. Access messages that are segmented over multiple Network PDUs can have either a 32-bit or 64-bit TransMIC. This allows a higher layer to determine the level of authentication required to securely deliver the access message and therefore apply the appropriate size for the TransMIC.

### 3.8.7.1 Upper transport layer authentication and encryption

Authentication and encryption of the access payload is performed by the upper transport layer.

The access payload is encrypted and authenticated using AES-CCM. This is identical to the way that Bluetooth low energy encryption and authentication works. An illustration of the upper transport layer encryption is shown in Figure 3.53.

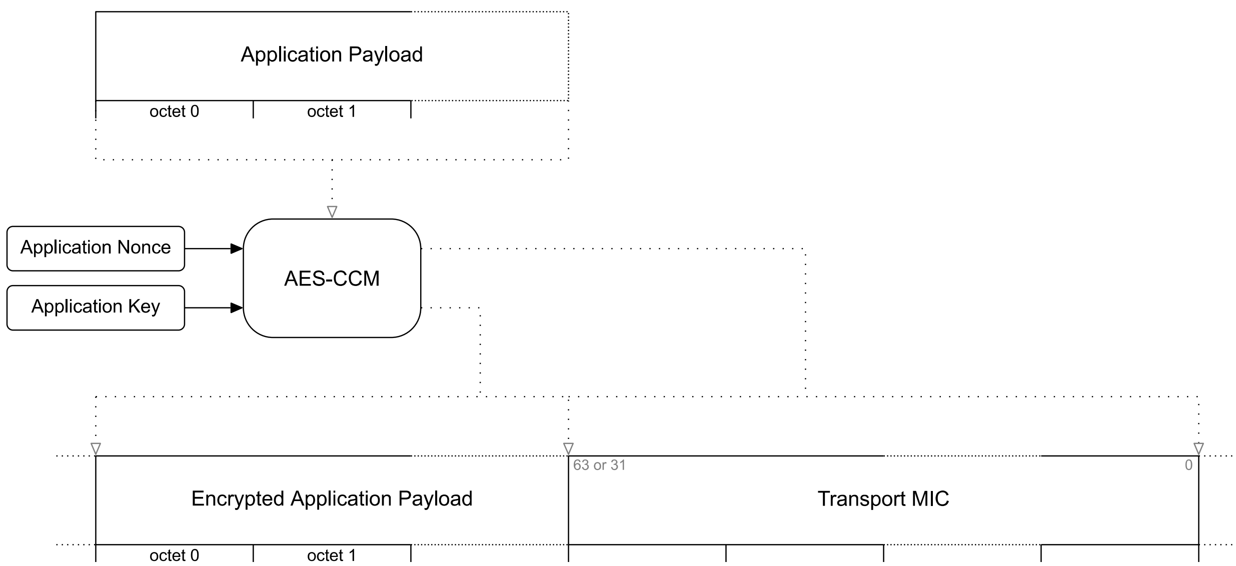


*Figure 3.53: Upper Transport layer encryption*

If the access payload is secured using the application key, then the access payload is encrypted using the application nonce and the application key.

If the access payload is secured using the device key, then the access payload is encrypted using the device nonce and the device key.

The nonce uses the sequence number and the source address, ensuring that two different nodes cannot use the same nonce. The IV Index is used to provide significantly more nonce values than the sequence number can provide for a given node. Management of the IV Index is described in Section 3.10.5.

Note: The authentication and encryption of the access payload is not dependent on the TTL value, meaning that as the access payload is relayed through a mesh network, the access payload does not need to be re-encrypted at each hop.

When using an application key and the destination address is a virtual address:

EncAccessPayload, TransMIC = AES-CCM (AppKey, Application Nonce, AccessPayload, Label UUID)

When using an application key and the destination address is a unicast address or a group address:

EncAccessPayload, TransMIC = AES-CCM (AppKey, Application Nonce, AccessPayload)

When using a device key and the destination address is a unicast address:

EncAccessPayload, TransMIC = AES-CCM (DevKey, Device Nonce, AccessPayload)

The concatenation of the encrypted access payload and the transport MIC is called the Upper Transport PDU:

Upper Transport PDU = EncAccessPayload || TransMIC

#### 3.8.7.2 Network layer authentication and encryption

The destination address and the TransportPDU are encrypted and authenticated using AES-CCM. This is identical to the way that Bluetooth low energy encryption and authentication works.

All Network PDUs are encrypted using an Encryption Key that is derived from a network key (see Section 3.8.6.3.1).

An illustration of the network layer encryption is shown in Figure 3.54.

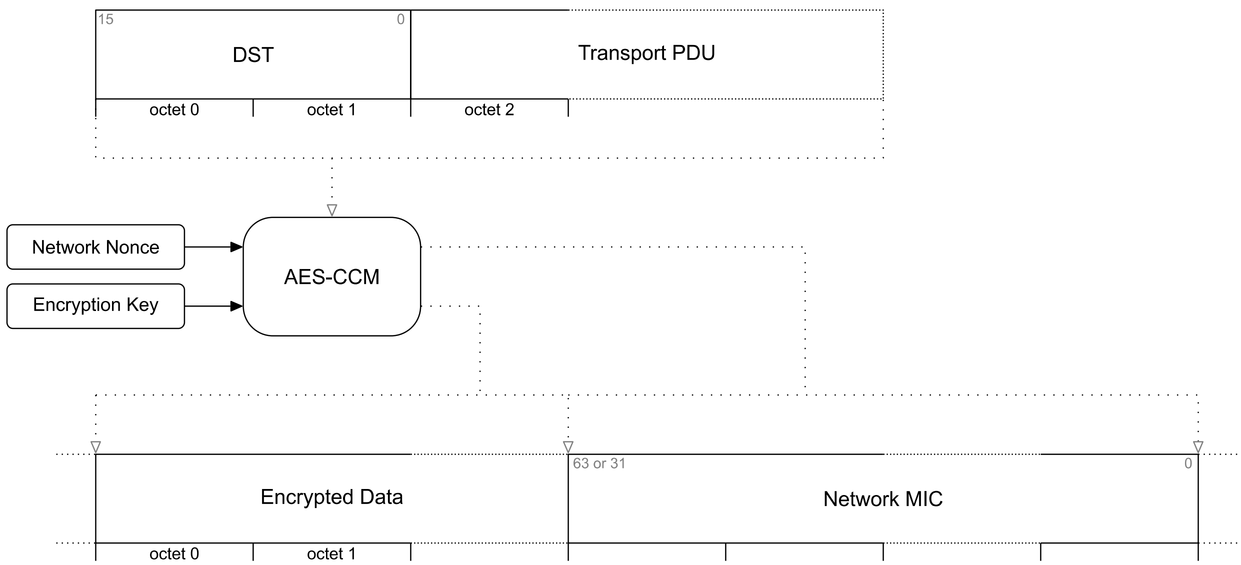


*Figure 3.54: Network layer encryption*

The following defines how this is performed:

EncDST || EncTransportPDU, NetMIC = AES-CCM (EncryptionKey, Network Nonce, DST || TransportPDU)

### 3.8.7.3 Network layer obfuscation

In order to obfuscate the Network Header (CTL, TTL, SEQ, SRC), these values shall be combined with a result of a single encryption function e, designed to prevent a passive eavesdropper from determining the identity of a node by listening to Network PDUs.

The obfuscation occurs after the application and network message integrity check values have been calculated. The obfuscation is calculated using information available from within the Network PDU. This obfuscation is designed only to help prevent a simple passive eavesdropper from tracking nodes. A determined attacker could still discover patterns within this obfuscation that can lead to the revealing of the source address or sequence number of a node. Critically, obfuscation does not enforce that inputs to the encryption function are unique.

Obfuscation does not protect the Privacy Key from compromise, and given the above design considerations for protection against only passive eavesdroppers, it is considered that the Privacy Key could be compromised with sufficient time. The design of obfuscation includes the IV Index, such that when the IV Index changes, any obfuscation attacks would have to start again.

To obfuscate the Network PDU, the first six octets of the Network PDU that have already been encrypted are combined with the IV Index and a Privacy Key.

These first six octets of the Network PDU that have been encrypted includes the DST, and can include up to four octets of the EncTransportPDU and/or NetMIC fields. These octets are known as the PrivacyRandom value.

The Privacy Key is derived using a key derivation function from the network key (see Section 3.8.6.3.1) to protect the network key even if the Privacy Key is compromised.

The IV Index is concatenated with the PrivacyRandom value and used along with the Privacy Key as inputs to the encryption function e. The output of this is known as the PECB value.

The first six octets of the PECB value is then exclusive-ORed with the TTL octet, the sequence number, and the source address fields, and become the ObfuscatedData. The Network PDU is transmitted as the concatenation of the NID/IVI octet, the ObfuscatedData, the EncDST, the EncTransportPDU, and the NetMIC.

An illustration of the network layer obfuscation is shown in Figure 3.55.

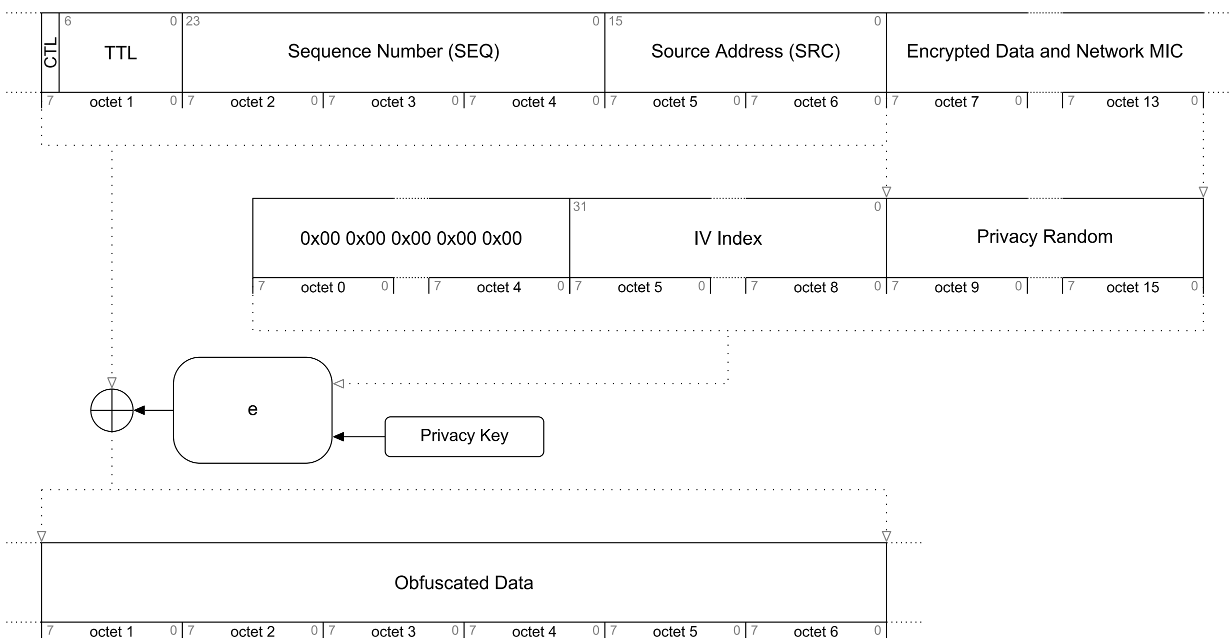


*Figure 3.55: Network layer obfuscation*

Privacy Random = (EncDST || EncTransportPDU || NetMIC)[0–6]

PECB = e (PrivacyKey, 0x0000000000 || IV Index || Privacy Random)

ObfuscatedData = (CTL || TTL || SEQ || SRC) $\oplus$ PECB[0–5]

When reversing this, the following operations are performed:

Privacy Random = (EncDST || EncTransportPDU || NetMIC)[0–6]

PECB = e (PrivacyKey, 0x0000000000 || IV Index || Privacy Random)

(CTL || TTL || SEQ || SRC) = ObfuscatedData ⊕ PECB[0–5]

### 3.8.8 Message replay protection

A message sent by a legitimate originating element can be passively received by an attacker and then replayed later without modification. This is called a replay attack.

Since the originating element has encrypted and authenticated the message using the correct keys, the receiver cannot determine whether it is under a replay attack solely by performing the message integrity checks (i.e., on the Network MIC and, if applicable, on the Transport MIC).

To increase protection against replay attacks, each element increases the sequence number for each new message that it sends. If a valid message has been received from an originating element with a specific sequence number, any future messages from the same originating element that contain numerically lower or equal sequence numbers than the last valid sequence number are very likely replayed messages and shall be discarded. Therefore, messages are delivered to the access layer in sequence number order.

If a lower IV Index from the same originating element has been received, the message shall be discarded.

A node shall implement replay protection for all Access and Control messages that are received from other elements, as well as for Proxy Configuration messages, if applicable.

If a node does not have enough resources to perform replay protection for a given source address, then the node shall discard the message immediately upon reception.

An implementation may perform the replay protection at any layer and in any order with respect to the message authentication steps (the network layer decryption and the transport layer decryption), in order to optimize the message processing flow, the number of cryptographic operations or the memory usage.

However, the implementation shall follow the fundamental requirement that it shall either be able to determine if a certain message is being replayed, or it shall discard the message immediately upon reception.

Figure 3.56 illustrates an example of a replay protection list implementation that handles a multi-segment message transaction which is under a replay attack. The sequence number of the last segment that has been received for this message is stored for that peer node in the replay protection list.

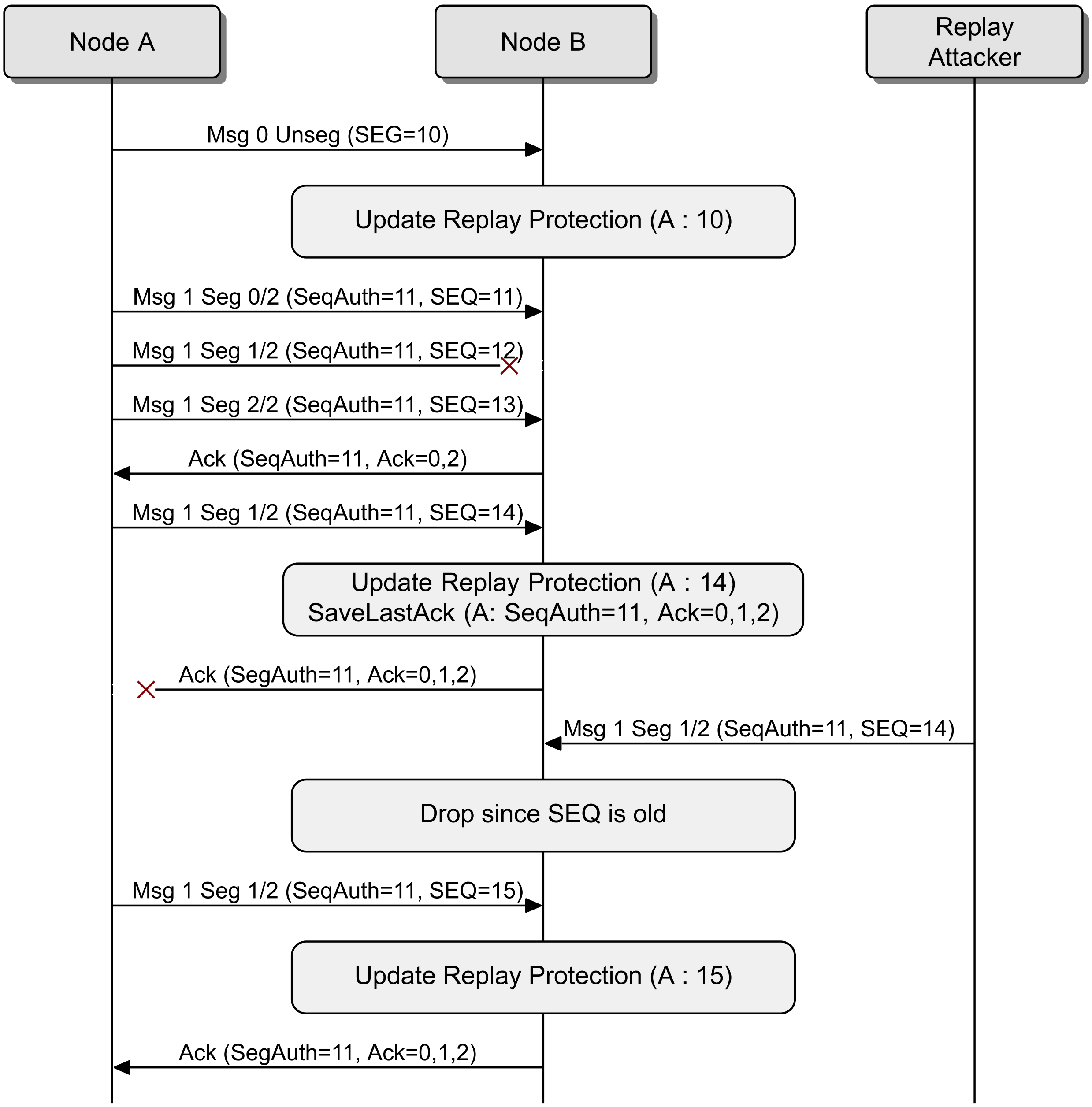


*Figure 3.56: Example of updating replay protection list for segmented messages*

## 3.9 Mesh beacons

Mesh beacons are packets advertised periodically by nodes and unprovisioned devices.

Mesh beacons are contained in a «Mesh Beacon» AD Type. The first octet of the Mesh Beacon advertising data payload (Beacon Type field) determines the type of beacon. Mesh beacons are forwarded to other bearers using the Proxy protocol (see Section 6).

The format of the Mesh Beacon AD Type is shown in Figure 3.57.

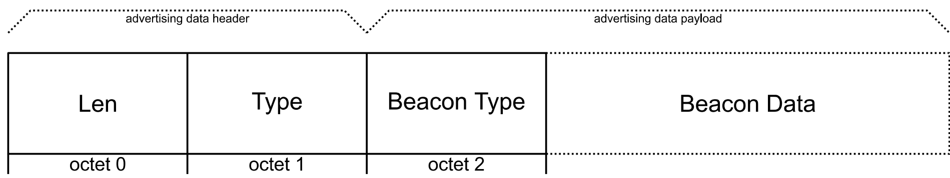


*Figure 3.57: Mesh Beacon AD Type format*

The Beacon Type values are defined in Table 3.72.

| Value | Definition |
|---|---|
| 0x00 | Unprovisioned Device beacon |
| 0x01 | Secure Network beacon |
| 0x02–0xFF | Reserved for Future Use |

*Table 3.72: Beacon Type values*

Mesh beacons shall be advertised with non-connectable and non-scannable undirected advertising events.

### 3.9.1 Endianness

All multiple-octet numeric values in mesh beacons shall be sent in “big endian”, as described in Section 3.1.1.1.

### 3.9.2 Unprovisioned Device beacon

The Unprovisioned Device beacon is used by devices that are unprovisioned to allow them to be discovered by a Provisioner.

The format of this beacon is illustrated in Figure 3.58 and defined in Table 3.73.

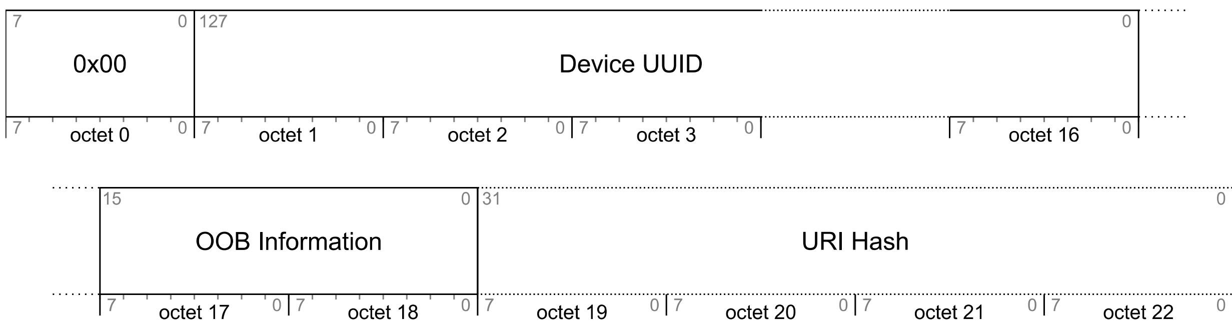


*Figure 3.58: Unprovisioned device beacon format*

| Field | Size (octets) | Notes |
|---|---|---|
| Beacon Type | 1 | Unprovisioned Device beacon type (0x00) |
| Device UUID | 16 | Device UUID uniquely identifying this device (see Section 3.10.3) |
| OOB Information | 2 | See Table 3.74 |
| URI Hash | 4 | Hash of the associated URI advertised with the URI AD Type (optional field) |

*Table 3.73: Unprovisioned Device beacon format*

The OOB Information field shown in Table 3.74 is used to help drive the provisioning process by indicating the availability of OOB data, such as a public key of the device.

| Bit | Description |
|---|---|
| 0 | Other |
| 1 | Electronic / URI |
| 2 | 2D machine-readable code |
| 3 | Bar code |
| 4 | Near Field Communication (NFC) |
| 5 | Number |
| 6 | String |
| 7 | Reserved for Future Use |
| 8 | Reserved for Future Use |
| 9 | Reserved for Future Use |
| 10 | Reserved for Future Use |
| 11 | On box |
| 12 | Inside box |

| Bit | Description |
|---|---|
| 13 | On piece of paper |
| 14 | Inside manual |
| 15 | On device |

*Table 3.74: OOB Information field*

Along with the Unprovisioned Device beacon, the device may also advertise a separate non-connectable advertising packet with a Uniform Resource Identifier (URI) data type (as defined in [7]) that points to out-of-band (OOB) information such as a public key. To allow the association of the advertised URI with the Unprovisioned Device beacon, the beacon may contain an optional 4-octet URI Hash field.

The value of the URI Hash field is calculated using the following formula:

URI Hash = s1(URI Data)[0-3]

The URI Data is a buffer containing the Uniform Resource Identifier (URI) data type, as defined in [7].

### 3.9.3 Secure Network beacon

The Secure Network beacon is used by nodes to identify the subnet and its security state.

The format of this beacon is illustrated in Figure 3.59 and defined in Table 3.75.

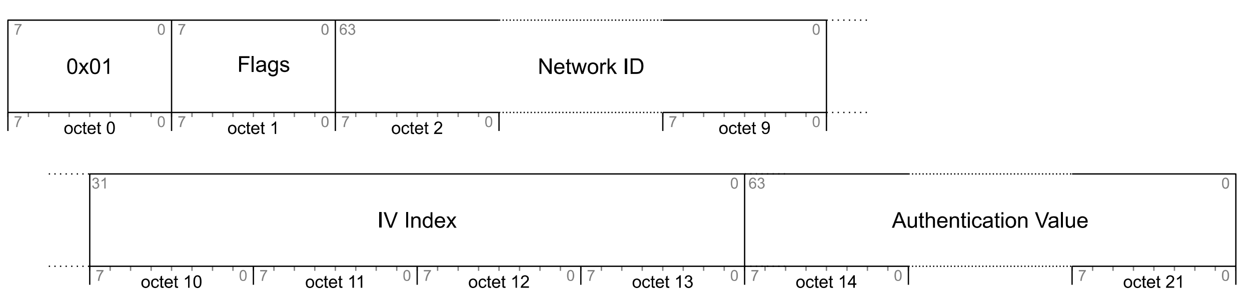


*Figure 3.59: Secure Network beacon*

| Field | Size (octets) | Notes |
|---|---|---|
| Beacon Type | 1 | Secure Network beacon (0x01) |
| Flags | 1 | Contains the Key Refresh Flag and IV Update Flag |
| Network ID | 8 | Contains the value of the Network ID |
| IV Index | 4 | Contains the current IV Index |

| | | |
|---|---|---|
| Authentication Value | 8 | Authenticates security network beacon |

*Table 3.75: Secure Network beacon format*

The Flags field is defined in Table 3.76 as:

| Bits | Definition |
|---|---|
| 0 | Key Refresh Flag<br>0: False<br>1: True |
| 1 | IV Update Flag<br>0: Normal operation<br>1: IV Update active |
| 2–7 | Reserved for Future Use |

*Table 3.76: Flags field definition*

The Network ID field contains the Network ID of this network.

The IV Index field contains the current IV Index of this mesh network.

The Authentication Value field is computed as defined below:

$$\text{Authentication Value} = \text{AES-CMAC}_{\text{BeaconKey}}\ (\text{Flags} \mid\mid \text{Network ID} \mid\mid \text{IV Index})\ [0–7]$$

The generation of the Secure Network beacon is illustrated in Figure 3.60.

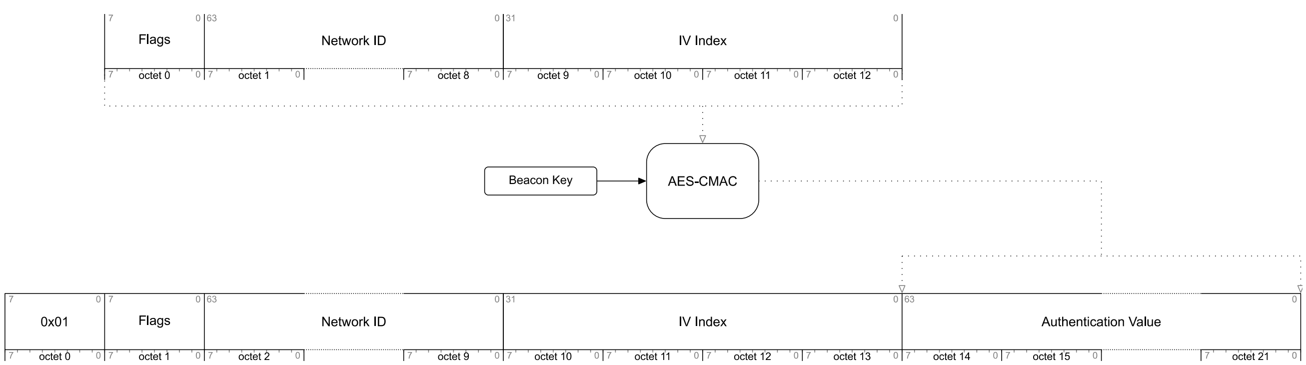


*Figure 3.60: Secure Network beacon generation*

### 3.9.3.1 Secure Network beacon behavior

When a Secure Network beacon is received on a known subnet, the node shall monitor for IV Index updates (see Section 3.10.5) and Key Refresh procedures (see Section 3.10.4).

A Secure Network beacon may be sent for each subnet that a node is a member of to identify the subnet and inform about IV Index updates (see Section 3.10.5) and Key Refresh procedures (see Section 3.10.4).

Relay and Friend nodes should send beacons and other nodes may send beacons. The time between sending two consecutive beacons is called the Beacon Interval. An implementation may define the Beacon Interval together with a back-off procedure to prevent other nodes from overloading the network with too many beacons. The expected behavior is that each node receives one beacon for a given subnet approximately every 10 seconds.

To determine the Beacon Interval, the node should continuously observe beacons and keep a rolling count of the number of beacons for a single subnet over a given observation period. The Beacon Interval should be determined using the formula below:

Beacon Interval = Observation Period * (Observed Number of Beacons + 1) / Expected Number of Beacons

The Observation Period in seconds should typically be double the typical Beacon Interval. Each of the subnets has a separate Secure Network beacon, and therefore, the Expected Number of Beacons, Observed Number of Beacons, and Observation Period may be different for each subnet.

The Observed Number of Beacons is the number of beacons observed for this subnet over the Observation Period.

The Expected Number of Beacons is the Observation Period divided by 10 seconds.

## 3.10 Mesh network management

### 3.10.1 Mesh Network Creation procedure

To create a mesh network, a Provisioner is required. A Provisioner shall generate a network key, provide an IV Index, and allocate a unicast address.

The network key shall be generated using a random number generator, which shall be compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1].

The IV Index shall be set to 0x00000000.

The unicast address shall be set to a unicast address that is allocated by the Provisioner.

The mesh network is created using the above information. The Provisioner's primary element shall be assigned the unicast address. The Provisioner's other elements shall be allocated the sequential addresses after the unicast address.

The Provisioner can then find unprovisioned devices by scanning for Unprovisioned Device beacons using active or passive scanning. The Provisioner can then provision these devices to become nodes within the mesh network. Once these nodes have been provisioned, the Configuration Client can then configure the nodes by providing them application keys and setting publish and subscribe addresses so that the nodes can communicate with each other.

Note: The Provisioner's device key is only used when one Provisioner is communicating directly with another Provisioner and this device key has been communicated OOB. Device keys of Provisioners should be coordinated across multiple Provisioners.

### 3.10.2 Temporary guest access

It is possible to provide a node with temporary guest access to a mesh network. This is done by creating a separate guest subnet by providing a separate network key to the guest and to the nodes the guest will have access to.

Separate application keys are also provided to the guest to restrict the models that the guest has access to at the access layer.

The guest never obtains application keys or network keys used by nodes and models that are excluded from guest access. Only nodes that belong to the guest subnet will communicate with the guest node; within these nodes, only models bound to the guest application keys can be used by the guest. This allows guest access to be very finely controlled down to specific nodes and functionalities.

Guests cannot initiate IV Index updates on the primary subnet. This protects the IV Index, which is a network shared resource, from a potentially malicious behavior.

Guest access is configured by a Configuration Client using the Configuration Server model that is secured by device keys. Multiple guests may be provided with guest access, each within their own guest subnet and model domain.

Guest access is revoked by refreshing application and network keys through the Key Refresh procedure (see Section 3.10.4).

### 3.10.3 Device UUID

To decrease the complexity of deploying nodes, a unique Bluetooth BD_ADDR is not required for mesh operations. Instead, each node shall be assigned a 128-bit UUID known as the Device UUID. Device manufacturers shall follow the standard UUID format as defined in [8] and generation procedure to ensure the uniqueness of each Device UUID.

### 3.10.4 Key Refresh procedure

This procedure is used when the security of one or more network keys and/or one or more of the application keys has been compromised or could be compromised.

For example, when a node is removed from the network, all remaining nodes would have their keys changed such that the removed node would not have knowledge of the new security credentials being used if that node was compromised after being disposed. This is known as the "trash-can attack."

The procedure allows the blacklisting of some nodes (i.e., not sharing the new key(s) with some nodes that are considered compromised or a security risk) because the distribution of the new key(s) is based on device keys established during provisioning between a Configuration Client and each node.

The procedure consists of changing the network keys, the application keys, and all the derived credentials, with a minimal disruption to the operation of the network.

Each key index within a node holds either one or two keys. If two keys are being held, then the most recently added key is referred to as the new key and the other key is referred to as the old key.

The Key Refresh procedure manages the process of changing from one key to another key for a NetKey and its associated AppKeys. AppKeys that have not been given a new key value shall not be changed when their associated NetKey is updated.

The Key Refresh procedure is independent of the IV Update procedure. Both procedures can be performed at the same time, interleaved, or at different times. The behavior of the IV Update procedure has no impact on the Key Refresh procedure, and the Key Refresh procedure has no impact on the IV Update procedure.

The Key Refresh procedure uses three phases to move a network from the current state, using only old keys, to the new state, using only new keys, as illustrated in Figure 3.61:

- The first phase involves distributing new keys to each node. The nodes will continue to transmit using the old keys but can receive using the old keys and new keys.
- The second phase involves transmitting a Secure Network beacon that signals to the network that all nodes have the new keys. The nodes will then transmit using the new keys but can receive using the old keys and the new keys.
- The third phase involves transmitting another Secure Network beacon that signals to the network that all nodes should revoke the old keys. The nodes will transmit and receive using only the new keys.

It is possible to update each NetKey independently of all other NetKeys. A Key Refresh procedure for one NetKey can be in a different phase to another Key Refresh procedure for other NetKeys.

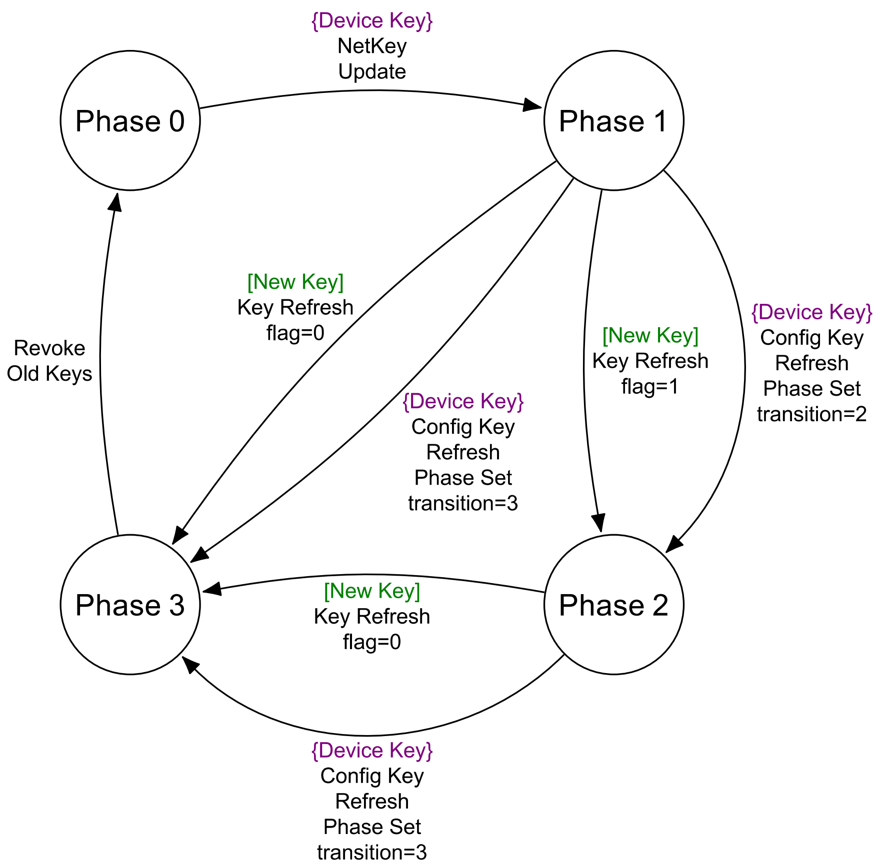


*Figure 3.61: Key Refresh diagram*

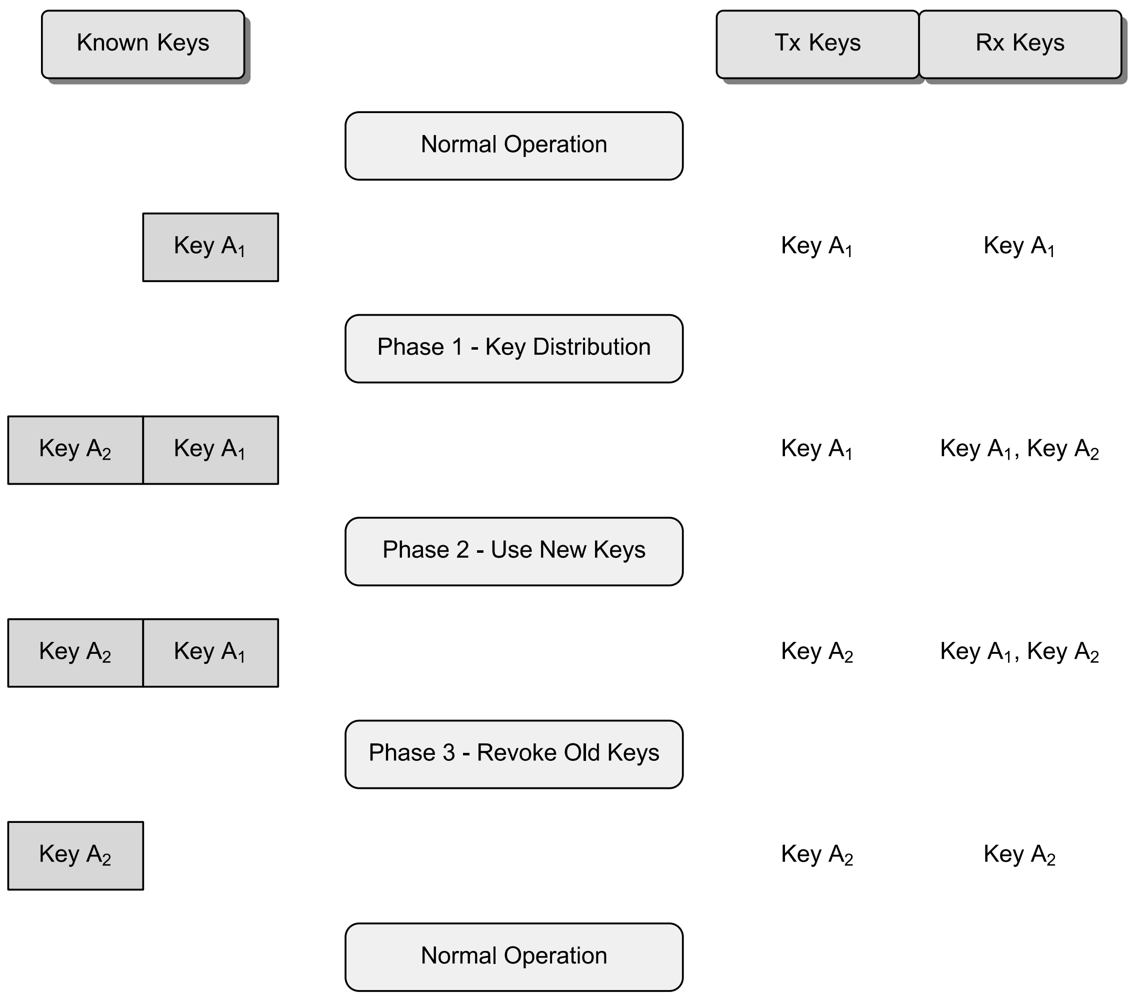


*Figure 3.62: Key Refresh procedure overview*

As illustrated in Figure 3.62, nodes in normal operation only know a single key, Key $A_1$. This key is used for transmitting and receiving packets. When Phase 1 (Key Distribution) is performed, each node will receive a new key that is stored in the same key index. The nodes will continue to transmit using the old key, Key $A_1$, but will additionally receive using the new key, Key $A_2$. Once all nodes have been informed of the new key, Phase 2 (Use New Keys) can start. This sends a signal around the network that the new key should now be used. The nodes will therefore start to transmit using the new key, Key $A_2$, but will also receive from the old and new keys. Finally, Phase 3 (Revoke Old Keys) will revoke the old keys meaning that nodes will only transmit and receive using a single key, Key $A_2$. After the old keys have been revoked, the nodes are back to normal operation.

To increase robustness of the Key Refresh procedure, the node shall successfully process a Config NetKey Update message on a valid NetKeyIndex when the NetKey value is different and the Key Refresh procedure has not been started, or when the NetKey value is the same in Phase 1. The Config NetKey Update message shall generate an error when the node is in Phase 2 or Phase 3. The Config AppKey

Update message shall generate an error when node is in normal operation, Phase 2, or Phase 3 or in Phase 1 when the Config AppKey Update message on a valid AppKeyIndex when the AppKey value is different.

#### 3.10.4.1 Phase 1 – distribution of the new keys

The procedure is triggered by a Configuration Client. A Configuration Client shall determine the set of nodes that will receive the new NetKey and the new AppKeys bound to it. Any node not receiving the new keys is blacklisted (i.e., they will effectively be removed from the network in Phase 3).

The Configuration Client shall send the new keys to each node that is not blacklisted. New keys are distributed using the Config NetKey Update message and the Config AppKey Update message, see Sections 4.3.2.32 and 4.3.2.38.

Upon receiving the new keys, the node shall store them. During this phase, the node shall transmit using the old keys and receive using both the old keys and the new keys.

The Configuration Client should be aware that Low Power nodes may have a very high latency, and therefore new keys may take additional time to be delivered to those nodes. On receiving Segment Acknowledgments with the OBO field set to 1 to key update messages sent to a Low Power node, a Configuration Client may perform a PollTimeout List procedure to the Low Power node's Friend node (identifying the Friend node using the value of SRC field of the Segment Acknowledgment) in order to obtain the current value of the PollTimeout timer, and schedule retries of NetKey or AppKey updates based on this value.

Upon receiving a Secure Network beacon with the Key Refresh flag set to 0 using the new NetKey in Phase 1, the node shall immediately transition to Phase 3, which effectively skips Phase 2.

When a Configuration Client determines that all nodes that are not blacklisted have received the new keys, Phase 1 is complete and it shall transition to Phase 2.

> Note: The Mesh Proxy Service advertising depends on the NetKey value and must be updated upon transition from Phase 1 when applicable.

#### 3.10.4.2 Phase 2 – switching to the new keys

The Configuration Client shall either start sending a Secure Network beacon with the Key Refresh flag set to 1, secured using the new NetKey, see Section 3.9.3, or initiate the Key Refresh Phase transition by sending a Config Key Refresh Phase Set message with the Transition parameter set to 0x02 to one or more nodes.

A Relay node or Friend node, when it is in Phase 2 for a given NetKey, shall send Secure Network beacons for the new NetKey with the Key Refresh flag set to 1 and it shall stop sending Secure Network beacons for the old NetKey.

Upon receiving a Secure Network beacon or a Friend Update message with the Key Refresh flag set to 1, or a Config Key Refresh Phase Set message with the Phase parameter set to 0x02, the node shall set the Key Refresh Phase for this NetKey to Phase 2. When in Phase 2, the node shall only transmit messages and Secure Network beacons using the new keys, shall receive messages using the old keys and the new keys, and shall only receive Secure Network beacons secured using the new NetKey.

The Configuration Client should be aware that Low Power nodes may have a very high latency, and therefore Low Power nodes may take additional time to receive the Key Refresh flag information from a Friend node.

When a Configuration Client determines that all nodes that are not blacklisted are in Phase 2, Phase 2 is complete and it shall transition to Phase 3.

#### 3.10.4.3 Phase 3 – revoking old keys

The Configuration Client shall either start sending a Secure Network beacon with the Key Refresh flag set to 0, secured using the new NetKey (see Section 3.9.3), or initiate Key Refresh Phase transition by sending a Config Key Refresh Phase Set message with the Transition parameter set to 0x03 to on one or more nodes. The Configuration Client shall revoke the old keys.

> Note: When a device has been recently provisioned and does not have the old keys, it will not know the old keys and therefore will not be able to revoke the old keys.

A Relay node or Friend node, when it is in Phase 3 for a given NetKey, shall send Secure Network beacons for the new NetKey with the Key Refresh flag set to 0.

Upon receiving a Secure Network beacon or a Friend Update message with the Key Refresh flag set to 0 or a Config Key Refresh Phase Set message with the Transition parameter set to 0x03, the node shall revoke the old keys and shall send Secure Network beacons for the new NetKey with the Key Refresh flag set to 0. The node will only transmit and receive using the new keys. It shall ignore Secure Network beacons and Friend Update messages secured using the new NetKey with the Key Refresh flag set to 1. After old keys are revoked, the Key Refresh state will be 0.

The Configuration Client should be aware that Low Power nodes may have a very high latency, and therefore Low Power nodes may take additional time to receive the Key Refresh flag information from a Friend node.

### 3.10.5 IV Update procedure

The IV Index provides entropy for the nonce used for the authenticated encryption (AES-CCM) in both the application and network layers. Therefore, it must be changed often enough to avoid repeated use of sequence numbers in the nonce. The IV Update procedure is initiated by any node that is a member of a primary subnet. This may be done when the node believes it is at risk of exhausting its sequence numbers, or it determines another node is close to exhausting its sequence numbers. The node changes its IV Index and sends an indication to other nodes in the mesh that the IV Index is being updated. This is then followed by a change back to normal operation by the same or some other node in the mesh.

At least one node within a connected subnet with a key index different from 0x000 must also be on the primary subnet.

Note: Nodes that rarely send messages will rarely initiate the IV Update procedure.

The IV Update procedure defines two states of operation:

- Normal Operation – IV Update Flag = 0
- IV Update in Progress – IV Update Flag = 1

During the Normal Operation state, the IV Update Flag in the Secure Network beacon and in the Friend Update message shall be set to 0. When this state is active, a node shall transmit using the current IV Index and shall process messages from the current IV Index and also the current IV Index - 1.

For example, when IV Update Flag is set to 0, and the current IV Index is equal to 0x00101847, then the node shall transmit using the IV Index 0x00101847 and accept messages received using the IV Index 0x00101847 when the IVI field in the network layer is set to 1, and 0x00101846 when the IVI field in the network layer is set to 0.

If a node in Normal Operation receives a Secure Network beacon with an IV index greater than the last known IV Index + 1, it may initiate an IV Index Recovery procedure, see Section 3.10.6.

If a node in Normal Operation receives a Secure Network beacon with an IV index equal to the last known IV index+1 and the IV Update Flag set to 0, the node may update its IV without going to the IV Update in Progress state, or it may initiate an IV Index Recovery procedure (Section 3.10.6), or it may ignore the Secure Network beacon. The node makes the choice depending on the time since last IV update and the likelihood that the node has missed the Secure Network beacons with the IV update Flag set to 1.

If a node in Normal Operation receives a Secure Network beacon with an IV index less than the last known IV Index or greater than the last known IV Index + 42, the Secure Network beacon shall be ignored.

Note: This above requirement allows a node to be away from the network for 48 weeks. A node that is away from a network for longer than 48 weeks must be reprovisioned.

If this node is a member of a primary subnet and receives a Secure Network beacon on a secondary subnet with an IV Index greater than the last known IV Index of the primary subnet, the Secure Network beacon shall be ignored.

A node shall not start an IV Update procedure more often when once every 192 hours.

After 96 hours of operating in Normal Operation, a node may initiate the IV Update procedure by transitioning to the IV Update in Progress state. When a node transitions from the Normal Operation state to the IV Update in Progress state, the IV Index on the node shall be incremented by one.

The transition from Normal Operation state to IV Update in Progress state must occur at least 96 hours before the sequence numbers are exhausted, as defined in Section 3.8.3.

A node that is in Normal Operation state that receives and accepts a Secure Network beacon with the IV Update Flag set to 1 (indicating the IV Update in Progress state) should transition to the IV Update in Progress state as soon as possible.

During the IV Update in Progress state, the IV Update Flag in the Secure Network beacon and in the Friend Update message shall be set to 1. When this state is active, a node shall transmit using the current IV Index - 1 and shall process messages from the current IV Index - 1 and also the current IV Index.

For example, if the IV Index was 0x00101847 before transitioning from the Normal Operation state to the IV Update in Progress state, after transitioning, the IV Update Flag will be 1, the current IV Index will be 0x00101848, and the node shall transmit using the IV Index 0x00101847 and accept messages received using the IV Index 0x00101847 when the IVI field in the network layer is set to 1 and 0x00101848 when the IVI field in the network layer is set to 0. This allows all nodes that are in the Normal Operation state using the old IV Index to send messages to this node, and this node sends messages to those nodes that have not yet transitioned.

After at least 96 hours and before 144 hours of operating in IV Update in Progress state, the node shall transition back to the IV Normal Operation state and not change the IV Index. At the point of transition, the node shall reset the sequence number to 0x000000.

For example, when transitioning back to the Normal Operation state, the IV Update Flag will be 0, the current IV Index will be 0x00101848, the node shall transmit using the IV Index 0x00101848 and accept messages received using the IV Index 0x00101847 when the IVI field in the network layer is set to 1 and 0x00101848 when the IVI field in the network layer is set to 0. This allows the node to send messages to all nodes in the network whether they are also in the Normal Operation state or in the IV Update in Progress state. It also allows the node to receive messages from all nodes that are in the Normal Operation state or the IV Update in Progress state. A summary of the IV Update procedure is provided in Table 3.77 below.

| IV Index | IV Update Flag | IV Update Procedure State | IV Index Accepted | IV Index used when transmitting |
|---|---|---|---|---|
| n | 0 | Normal | n-1, n | n |
| m (m=n+1) | 1 | In Progress | m-1, m | m-1 |
| m | 0 | Normal | m-1, m | m |

*Table 3.77: IV Update procedure summary*

A node that is in the IV Update in Progress state that receives and accepts a Secure Network beacon with the IV Update Flag set to 0 (indicating the Normal Operation state) should transition into the Normal Operation state as soon as possible.

A node shall defer state change from IV Update in Progress to Normal Operation, as defined by this procedure, when the node has transmitted a Segmented Access message or a Segmented Control message without receiving the corresponding Segment Acknowledgment messages. The deferred change of the state shall be executed when the appropriate Segment Acknowledgment message is received or timeout for the delivery of this message is reached.

Note: This requirement is necessary because upon completing the IV Update procedure the sequence number is reset to 0x000000 and the SeqAuth value would not be valid.

When a node is added to a network, the node is given an IV Index. If the node is added to a network when the network is in Normal operation, then it shall operate in Normal operation for at least 96 hours. If a node is added to a network while the network is in the IV Update in Progress state, then the node shall be given the new IV Index value and operate in Normal operation for at least 96 hours.

#### 3.10.5.1 IV Update test mode

To enable efficient testing of the IV Update procedure, a node shall support IV Update test mode used testing in connection with the Bluetooth Qualification Process. The activation of the test mode shall be carried out locally (via a HW or SW interface). The IV Update test mode only removes the 96-hour limit; all other behavior of the device shall be unchanged.

Two signals are defined in the IV Update test mode:

- Transit to IV Update in Progress signal
- Transit to Normal signal

When the Transit to IV Update in Progress signal is received, the node shall transition to the IV Update in Progress state, ignoring the 96 hour limit.

When the Transit to Normal signal is received, the node shall transition to the Normal state, ignoring the 96 hour limit.

### 3.10.6 IV Index Recovery procedure

A node shall support the IV index recovery procedure because a node that is away from the network for a long time may miss IV Update procedures, in which case it can no longer communicate with the other nodes. In order to recover the IV Index, the node must listen for a Secure Network beacon, which contains the Network ID and the current IV Index. Upon receiving and successfully authenticating a Secure Network beacon for a primary subnet whose IV Index is 1 or more higher than the current known IV Index, the node shall set its current IV Index and its current IV Update procedure state from the values in this Secure Network beacon.

Note: After the IV Index Recovery procedure has updated the IV Index, the 96 hour time limits for changing the IV Update procedure state, as defined in the IV Update procedure, do not apply.

Given that nodes collectively transmit a Secure Network beacon once every 10 seconds, a low duty cycle node will have to listen for an average of 5 seconds to recover the current IV Index before transmitting and receiving mesh messages. If a Low Power node has insufficient power to listen for 5 seconds, then it must stay up to date with the current IV Index by polling its Friend node at least once every 96 hours.

The node shall not execute more than one IV Index Recovery within a period of 192 hours.

### 3.10.7 Node Removal procedure

In some cases, it may be necessary to remove a node from a network (e.g., for security reasons or due to the hardware and/or software failure of the node).

Node removal is a consequence of blacklisting the node by excluding it from the Key Refresh procedure (see Section 3.10.4).

After a node is removed from a network, its unicast addresses may be reused by a Provisioner. A Provisioner shall only reuse these addresses after the current IV Index (at the time of removal) has been updated (see Section 3.10.5) in order to enable the SEQ numbers to be reused.

## 3.11 Message processing flow

The flow of messages through the layers defined by this specification, along with key decision points, is illustrated by Figure 3.64 (lower layers) and Figure 3.63 (upper layers).

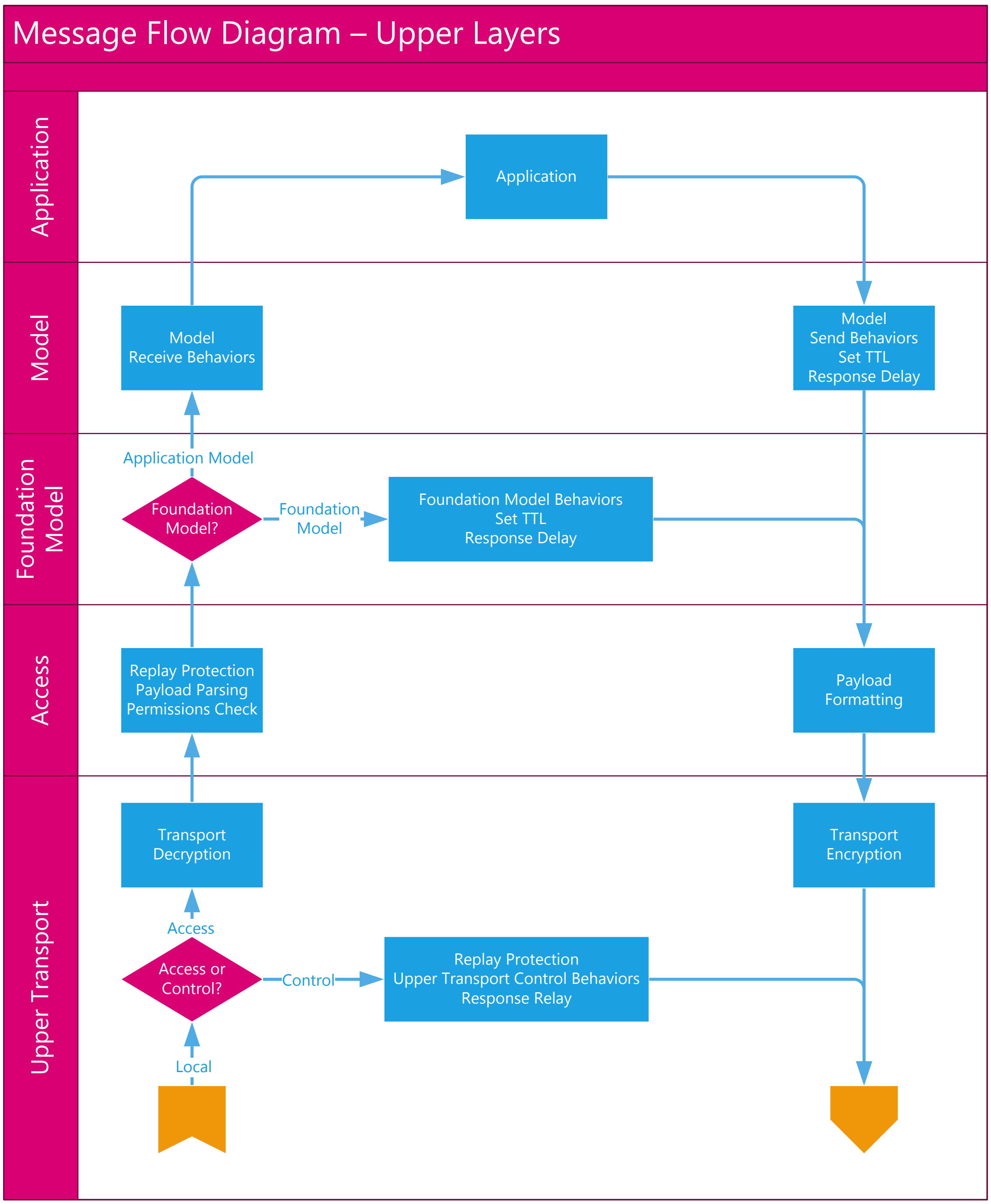

*Figure 3.63: Message flow diagram – upper layers*

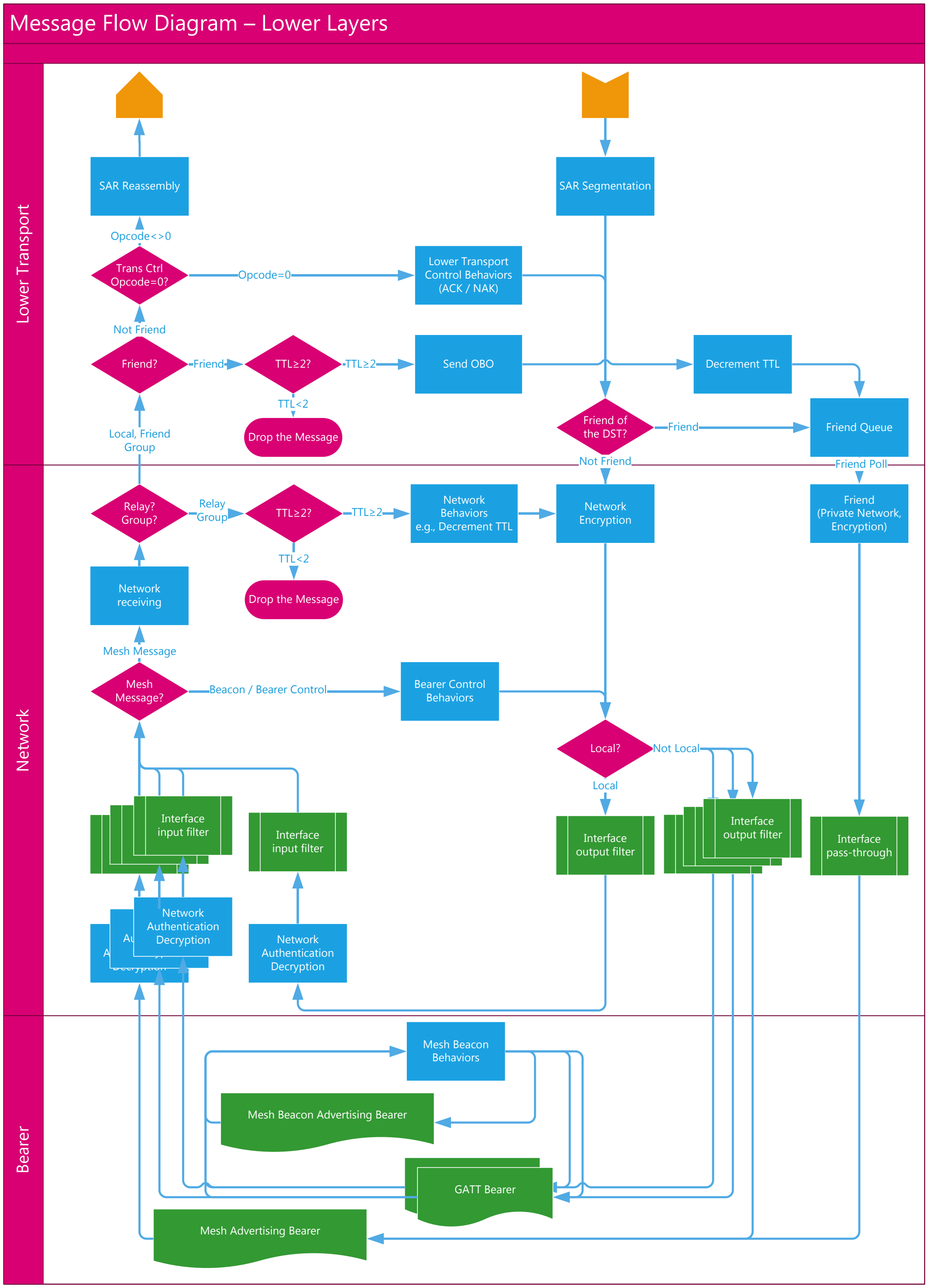
Message Flow Diagram – Lower Layers
Lower Transport
Network
Bearer
SAR Reassembly
SAR Segmentation
Opcode<>0
Trans Ctrl Opcode=0?
Opcode=0
Lower Transport Control Behaviors (ACK / NAK)
Not Friend
Friend?
Friend
TTL≥2?
TTL≥2
Send OBO
Decrement TTL
TTL<2
Drop the Message
Friend of the DST?
Friend
Friend Queue
Local, Friend Group
Not Friend
Friend Poll
Relay? Group?
Relay Group
TTL≥2?
TTL≥2
Network Behaviors e.g., Decrement TTL
Network Encryption
Friend (Private Network, Encryption)
TTL<2
Network receiving
Drop the Message
Mesh Message
Mesh Message?
Beacon / Bearer Control
Bearer Control Behaviors
Local?
Not Local
Local
Interface input filter
Interface input filter
Interface output filter
Interface output filter
Interface pass-through
Network Authentication Decryption
Network Authentication Decryption
Mesh Beacon Behaviors
Mesh Beacon Advertising Bearer
GATT Bearer
Mesh Advertising Bearer

*Figure 3.64: Message flow diagram – lower layers*

The flow of messages through the access layer defined by this specification, along with key decision points, is illustrated by Figure 3.65.

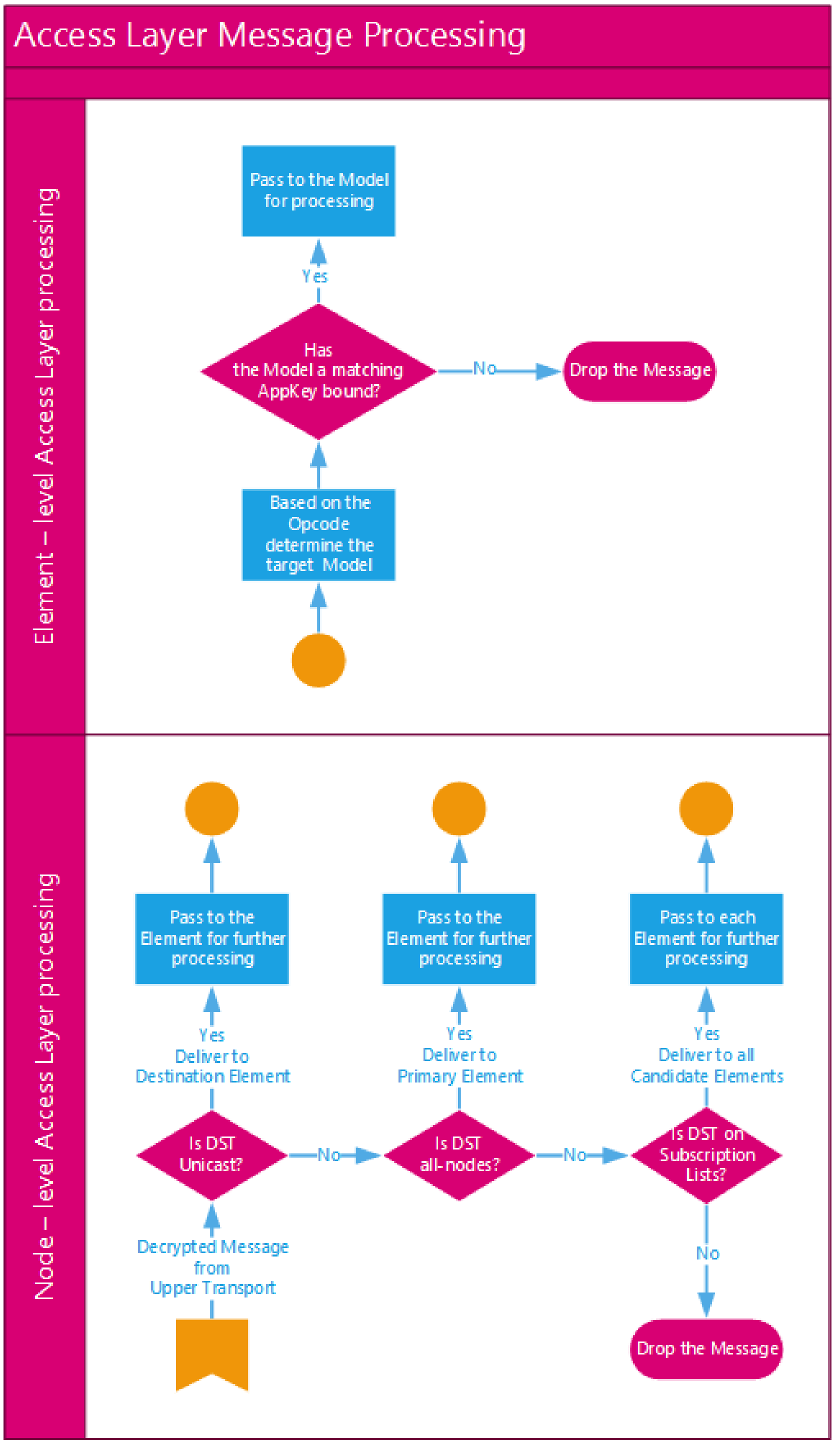


*Figure 3.65: Message flow diagram – access layer*

# 4 Foundation models

The Foundation Models define the access layer states, messages, and models required to configure and manage a mesh network.

## 4.1 Conventions

### 4.1.1 Endianness

All multiple-octet numeric values in this layer shall be “little endian”, as described in Section 3.1.1.2.

### 4.1.2 Log field transformation

In order to compress two-octet values into one-octet fields, the following logarithmic transformation is used: any two-octet value is mapped onto a one-octet field value representing the largest integer n, where $2^{(n-1)}$ is less than or equal to the two-octet value.

This transformation is represented in Table 4.1.

| Log Field Value | 2-octet Value |
|---|---|
| 0x01 | 0x0001 |
| 0x02 | 0x0002 through 0x0003 |
| 0x03 | 0x0004 through 0x0007 |
| 0x04 | 0x0008 through 0x000F |
| 0x05 | 0x0010 through 0x001F |
| 0x06 | 0x0020 through 0x003F |
| 0x07 | 0x0040 through 0x007F |
| 0x08 | 0x0080 through 0x00FF |
| 0x09 | 0x0100 through 0x01FF |
| 0x0A | 0x0200 through 0x03FF |
| 0x0B | 0x0400 through 0x07FF |
| 0x0C | 0x0800 through 0x0FFF |
| 0x0D | 0x1000 through 0x1FFF |
| 0x0E | 0x2000 through 0x3FFF |
| 0x0F | 0x4000 through 0x7FFF |

| Log Field Value | 2-octet Value |
|---|---|
| 0x10 | 0x8000 through 0xFFFF |

*Table 4.1: Log field values*

## 4.2 State definitions

The state of a node is defined using one or more state definitions. This section defines states used throughout this specification.

State definitions that are not required as part of this specification are defined in the Mesh Model specification [11] and follow the same format and architecture as mesh state definitions.

### 4.2.1 Composition Data

The Composition Data state contains information about a node, the elements it includes, and the supported models. The Composition Data is composed of a number of pages of information. Composition Data Page 0 is mandatory. All other pages are optional. All Composition Data Pages not defined in this specification are reserved for future use.

#### 4.2.1.1 Composition Data Page 0

The format of the Composition Data Page 0 is defined in Table 4.2.

| Field | Size (octets) | Notes |
|---|---|---|
| CID | 2 | Contains a 16-bit company identifier assigned by the Bluetooth SIG (the list is available at [6]) |
| PID | 2 | Contains a 16-bit vendor-assigned product identifier |
| VID | 2 | Contains a 16-bit vendor-assigned product version identifier |
| CRPL | 2 | Contains a 16-bit value representing the minimum number of replay protection list entries in a device (see Section 3.8.8) |
| Features | 2 | Contains a bit field indicating the device features, as defined in Table 4.3 |
| Elements | variable | Contains a sequence of element descriptions |

*Table 4.2: Composition Data Page 0 fields*

The Features field contains a bit field indicating the node capabilities as defined in Section 3.2. The format of the Features field is defined in Table 4.3.

| Bit | Feature | Notes |
|---|---|---|

| Bit | Feature | Notes |
|---|---|---|
| 0 | Relay | Relay feature support: 0 = False, 1 = True |
| 1 | Proxy | Proxy feature support: 0 = False, 1 = True |
| 2 | Friend | Friend feature support: 0 = False, 1 = True |
| 3 | Low Power | Low Power feature support: 0 = False, 1 = True |
| 4–15 | RFU | Reserved for Future Use |

*Table 4.3: Features field format*

The Elements field contains a sequence of one or more element descriptions. The format of each element description is defined in Table 4.4.

| Field | Size (octets) | Notes |
|---|---|---|
| Loc | 2 | Contains a location descriptor |
| NumS | 1 | Contains a count of SIG Model IDs in this element |
| NumV | 1 | Contains a count of Vendor Model IDs in this element |
| SIG Models | variable | Contains a sequence of NumS SIG Model IDs |
| Vendor Models | variable | Contains a sequence of NumV Vendor Model IDs |

*Table 4.4: Element description format*

The Loc field contains a location description as defined in the GATT Bluetooth Namespace Descriptors section of the Bluetooth SIG Assigned Numbers [4]. Values not defined in the GATT Units table are Reserved for Future Use.

The SIG Models field contains a sequence of NumS SIG Model IDs. For each extended model included in this sequence, all models it extends shall also be included.

The Vendor Models field contains a sequence of NumV Vendor Model IDs.

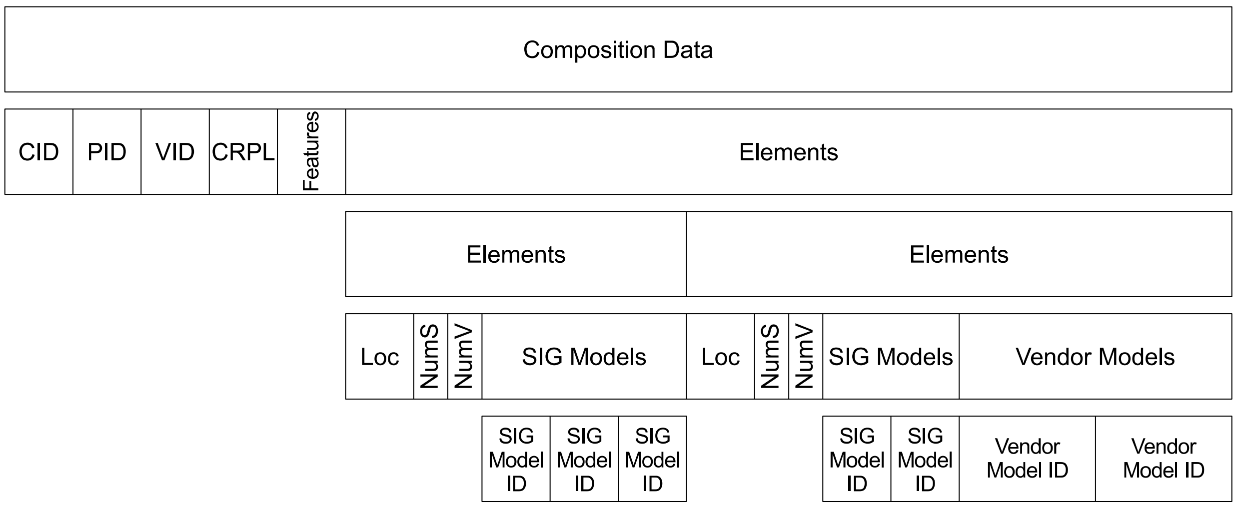


*Figure 4.1: Composition Data Page 0 format*

The example in Figure 4.1 shows a Composition Data Page 0 with two elements. Each element includes the location, the number of SIG Model IDs, and the number of Vendor Model IDs. In this example, the first element has three SIG Model IDs and no Vendor Model IDs, and the second element has two SIG Model IDs and two Vendor Model IDs.

### 4.2.2 Model Publication

The Model Publication state is a composite state that controls parameters of messages that are published by a model. The state includes a Publish Address, a Publish Period, a Publish AppKey Index, a Publish Friendship Credential Flag, a Publish TTL, a Publish Retransmission Count, and a Publish Retransmit Interval Steps. There is a single instance of this state for each model within a node. It is highly recommended that SIG Models defined by higher layer specifications and Vendor Models use instances of the Model Publication state to control the publishing of messages.

#### 4.2.2.1 Publish Address

The Publish Address state determines the destination address in messages sent by a model. The publish address shall be the unassigned address, a unicast address, a Label UUID, or a group address.

If the publish address of the model is the unassigned address, the model is inactive: it does not send any unsolicited messages out and can only send a response message to an incoming acknowledged message.

#### 4.2.2.2 Publish Period

The Publish Period state determines the interval at which status messages are published by a model. This is a 1-octet value and consists of two fields: a 2-bit field representing the step resolution and a 6-bit field representing the number of steps. The format of this state is defined in Table 4.5.

| **Field** | **Size (bits)** | **Description** |
|---|---|---|
| Number of Steps | 6 | The number of steps |
| Step Resolution | 2 | The resolution of the Number of Steps field |

*Table 4.5: Publish Period format*

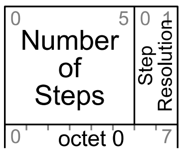


*Figure 4.2: Publish Period format*

The Step Resolution field enumerates the resolution of the Number of Steps field and the values are defined in Table 4.6.

| **Value** | **Description** |
|---|---|
| 0b00 | The Step Resolution is 100 milliseconds |
| 0b01 | The Step Resolution is 1 second |
| 0b10 | The Step Resolution is 10 seconds |
| 0b11 | The Step Resolution is 10 minutes |

*Table 4.6: Step Resolution values*

The Number of Steps field is a value representing the number of steps and the values are defined in Table 4.7.

| **Value** | **Description** |
|---|---|
| 0x00 | Publish Period is disabled |
| 0x01–0x3F | The number of steps |

*Table 4.7: Number of Steps values*

The Publish Period is calculated using the formula:

*Publish Period = Step Resolution * Number of Steps*

For example, if the Step Resolution is 0b10 and the Number of Steps is 0x31, then the Publish Period would be 490 seconds.

#### 4.2.2.3 Publish AppKey Index

The Publish AppKey Index state is the global AppKey Index of the application key used in messages sent by a model. The Publish AppKey Index shall be in the model to AppKey List as defined in Section 4.2.6.

#### 4.2.2.4 Publish Friendship Credentials Flag

The Publish Friendship Credential Flag is a 1-bit state controlling the credentials used to publish messages from a model. The Publish Friendship Credentials Flag values are described in Table 4.8.

| Value | Description |
|---|---|
| 0 | Master security material is used for Publishing |
| 1 | Friendship security material is used for Publishing |

*Table 4.8: Publish Friendship Credential Flag values*

When Publish Friendship Credential Flag is set to 1 and the friendship security material is not available, the master security material shall be used.

#### 4.2.2.5 Publish TTL

The Publish TTL state determines the TTL value for outgoing messages published by the model and the values are defined in Table 4.9. Setting Publish TTL to 0xFF will make the messages use the Default TTL as defined in Section 4.2.7.

| Value | Description |
|---|---|
| 0x00–0x7F | The Publish TTL value, represented as a 1-octet integer |
| 0x80–0xFE | Prohibited |
| 0xFF | Use Default TTL |

*Table 4.9: Publish TTL values*

#### 4.2.2.6 Publish Retransmit Count

The Publish Retransmit Count state is a 3-bit value controlling the number of times that a message published by a model will be retransmitted after the original message is received. For example, a value of 0b000 represents no retransmissions, and a value of 0b111 represents 7 retransmissions.

#### 4.2.2.7 Publish Retransmit Interval Steps

The Publish Retransmit Interval Steps state is a 5-bit value controlling the interval between retransmissions of a message that is published by a model. The state represents the number of 50-millisecond steps that shall transpire before a message that was published by a model is retransmitted.

The retransmission interval is calculated using the following formula:

retransmission interval = (Publish Retransmit Interval Steps + 1) * 50

For example, a value of 0b10000 represents an interval of 850 milliseconds.

### 4.2.3 Subscription List

The Subscription List state is a list of group addresses or Label UUIDs. There is a single instance of a Subscription List for each model's element within a node. It is highly recommended that SIG Models defined by higher layer specifications and Vendor Models use instances of the Model Publication state to control the publishing of messages.

Instances of models that have their states bound (i.e., when one model extends another as described in Section 2.3.6) shall share a single instance of a Subscription List per element.

The Subscription List is used by a model when receiving access messages as defined in Section 3.7.4.2.

### 4.2.4 NetKey List

The NetKey List state is an indexed list of NetKeys.

Each entry in the NetKey List holds up to two key values: the old key value and the new key value. The use of the old key and the new key values is described in the Key Refresh procedure (see Section 3.10.4).

The NetKey List shall contain a minimum of one NetKey.

### 4.2.5 AppKey List

The AppKey List state is an indexed list of AppKeys.

Each entry in the AppKey List holds an AppKey Index and up to two key values: the old key value and the new key value. The use of the old key and the new key values is described in the Key Refresh procedure (see Section 3.10.4).

### 4.2.6 Model to AppKey List

The Model to AppKey List state is a list of relationships between models and AppKeys. A model may be associated with one or more AppKeys.

### 4.2.7 Default TTL

The Default TTL state determines the TTL value used when sending messages. The Default TTL is applied by the access layer unless the application specifies a TTL. The Default TTL values are defined in Table 4.10.

| Value | Description |
|---|---|
| 0x00, 0x02–0x7F | The Default TTL state |
| 0x01, 0x80–0xFF | Prohibited |

*Table 4.10: Default TTL values*

### 4.2.8 Relay

The Relay state indicates support for the Relay feature. If the Relay feature is supported, then this also indicates and controls whether the Relay feature is enabled or disabled. The values are defined in Table 4.11.

| Value | Description |
|---|---|
| 0x00 | The node support Relay feature that is disabled |
| 0x01 | The node supports Relay feature that is enabled |
| 0x02 | Relay feature is not supported |
| 0x03–0xFF | Prohibited |

*Table 4.11: Relay values*

If Relay feature is not supported, the Relay state value shall be 0x02 and not be changed.

If Relay feature is supported, the Relay state value 0x02 shall not be used.

### 4.2.9 Attention Timer

The Attention Timer state determines if the Attention Timer state is on or off. This is generally intended to allow an element to attract human attention and, among others, is used during provisioning (see Section 5.4.2).

A device may not support the Attention Timer. On a device that does not support the Attention Timer, the Attention Timer state shall always be set to zero.

If the Attention Timer is non-zero for an element, Attention Timer state is on. If the Attention Timer is zero for an element, the Attention Timer state is off.

When the Attention Timer state is on, the value determines how long the element shall remain attracting human’s attention. The element does that by behaving in a human-recognizable way (e.g., a lamp flashes, a motor makes noise, an LED blinks). The exact behavior is implementation specific and depends on the type of device. Normal behavior of the element is still active, although the method of identification may override the physical state of the device.

The Attention Timer is a momentary state, active for a time indicated by its value, in seconds. The value is decremented every second by 1 until it reaches zero. The values for this state are defined in Table 4.12.

| Value | Description |
|---|---|
| 0x00 | Off |
| 0x01–0xFF | On, remaining time in seconds |

*Table 4.12: Attention Timer values*

### 4.2.10 Secure Network Beacon

The Secure Network Beacon state determines if a node is periodically broadcasting Secure Network beacon messages (see Section 3.9.3). The values for this state are defined in Table 4.13.

| Value | Description |
|---|---|
| 0x00 | The node is not broadcasting a Secure Network beacon |
| 0x01 | The node is broadcasting a Secure Network beacon |
| 0x02–0xFF | Prohibited |

*Table 4.13: Secure Network Beacon values*

### 4.2.11 GATT Proxy

The GATT Proxy state indicates if the Mesh Proxy Service (see Section 7.2) is supported, and if supported, it indicates and controls the status of the Mesh Proxy Service. The values for this state are defined in Table 4.14.

| Value | Description |
|---|---|
| 0x00 | The Mesh Proxy Service is running, Proxy feature is disabled |
| 0x01 | The Mesh Proxy Service is running, Proxy feature is enabled |
| 0x02 | The Mesh Proxy Service is not supported, Proxy feature is not supported |
| 0x03–0xFF | Prohibited |

*Table 4.14: GATT Proxy values*

If the Mesh Proxy Service is not supported, the GATT Proxy state value shall be 0x02 and shall not be changed.

If the Mesh Proxy Service is supported, the GATT Proxy state value 0x02 shall not be used.

Upon transition from GATT Proxy state 0x01 to GATT Proxy state 0x00 the GATT Bearer Server shall disconnect all GATT Bearer Clients.

Note: The Configuration Client should turn off the Proxy state as the last step in the configuration process.

#### 4.2.11.1 Binding with GATT Proxy state

When the GATT Proxy state is set to 0x00, the Node Identity state for all subnets shall be set to 0x00 and shall not be changed.

### 4.2.12 Node Identity

The Node Identity state determines if a node that supports the Mesh Proxy Service (see Section 7.2) is advertising on a subnet using Node Identity (see Section 7.2.2.2.3) messages. The values for this state are defined in Table 4.15.

| Value | Description |
|---|---|
| 0x00 | Node Identity for a subnet is stopped |
| 0x01 | Node Identity for a subnet is running |
| 0x02 | Node Identity is not supported |
| 0x03–0xFF | Prohibited |

*Table 4.15: Node Identity values*

If the Mesh Proxy Service is not supported, the Node Identity state value shall be 0x02 and not be changed.

If the Mesh Proxy Service is supported, the Node Identity state value 0x02 shall not be used.

### 4.2.13 Friend

The Friend state indicates support for the Friend feature. If Friend feature is supported, then this also indicates and controls whether Friend feature is enabled or disabled. The values for this state are defined in Table 4.16.

| Value | Description |
|---|---|
| 0x00 | The node supports Friend feature that is disabled |
| 0x01 | The node supports Friend feature that is enabled |
| 0x02 | The Friend feature is not supported |
| 0x03–0xFF | Prohibited |

*Table 4.16: Friend values*

If the Friend feature is not supported, the Friend state value shall be 0x02 and not be changed.

If the Friend feature is supported, the Friend state value 0x02 shall not be used.

If the Friend feature is supported and the Friend state changes to value 0x00 and if a node is a friend for one or more Low Power nodes, the node shall terminate all friend relationships and clear the associated Friend Queue.

### 4.2.14 Key Refresh Phase

The Key Refresh Phase state indicates and controls the Key Refresh procedure (see Section 3.10.4) for each NetKey in the NetKey List. The values for this state are defined in Table 4.17.

| Value | Description |
|---|---|
| 0x00 | Normal operation; Key Refresh procedure is not active |
| 0x01 | First phase of Key Refresh procedure |
| 0x02 | Second phase of Key Refresh procedure |
| 0x03–0xFF | Prohibited |

*Table 4.17: Key Refresh Phase state values*

Table 4.18 defines all possible transitions of the Key Refresh Phase state that can be controlled using this state. All other transitions are handled internally (e.g., the transition from 0x00 to 0x01 when a device receives a key update) by Key Refresh procedure.

| Old State | Transition | New State | Description |
|---|---|---|---|
| 0x00 | 0x03 | 0x00 | Transition 3 from Key Refresh Phase 0x00 does not cause any state change. |
| 0x01 | 0x02 | 0x02 | Transition 2 from Key Refresh Phase 0x01 moves to Key Refresh Phase 0x02 |
| 0x01 | 0x03 | 0x00 | Transition 3 from Key Refresh Phase 0x01 invokes Key Refresh Phase 3 and then moves to Key Refresh Phase 0x00. |
| 0x02 | 0x02 | 0x02 | Transition 2 from Key Refresh Phase 0x02 does not cause any state change. |
| 0x02 | 0x03 | 0x00 | Transition 3 from Key Refresh Phase 0x02 invokes Key Refresh Phase 3 and then moves to Key Refresh Phase 0x00. |

*Table 4.18: Controllable Key Refresh transition values*

### 4.2.15 Health Fault

The Health Fault state is a composite state that represent a warning or an error condition of an element.

The Health Fault state is identified by Company ID and may be present in the node for more than one Company ID.

#### 4.2.15.1 Current Fault

The Current Fault state is a 1-octet value of the most recently performed self-test and an array containing a sequence of 1-octet values, each representing the current warning or error condition of an element. The format of this state is defined in Table 4.19.

| Field | Size | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a most recently performed self-test |
| FaultArray | N | Array of current faults |

*Table 4.19: Current Fault format*

Values for the Test ID are defined in Table 4.20.

| Value | Description |
|---|---|
| 0x00 | Standard test |
| 0x01–0xFF | Vendor specific test |

*Table 4.20: Test ID values*

The FaultArray values are defined in Table 4.21.

Values 0x01–0x7F are Bluetooth assigned numbers, with values representing specific warning and error conditions. Values 0x80–0xFF are vendor specific and may be used to represent warnings and errors defined by device manufacturers. The Current Fault FaultArray is empty when no warning or error condition is present. The FaultArray reflects a real time state. This means when a fault condition arises, a corresponding record is present in the FaultArray and when a fault condition is not present, the corresponding record is removed from the FaultArray automatically.

A warning indicates the state of an element that is operating within the design limits but close to them.

An error indicates that the state of an element is outside the design limits and may not perform its functions.

| Value | Description |
|---|---|
| 0x00 | No Fault |
| 0x01 | Battery Low Warning |
| 0x02 | Battery Low Error |
| 0x03 | Supply Voltage Too Low Warning |
| 0x04 | Supply Voltage Too Low Error |
| 0x05 | Supply Voltage Too High Warning |
| 0x06 | Supply Voltage Too High Error |

| Value | Description |
|---|---|
| 0x07 | Power Supply Interrupted Warning |
| 0x08 | Power Supply Interrupted Error |
| 0x09 | No Load Warning |
| 0x0A | No Load Error |
| 0x0B | Overload Warning |
| 0x0C | Overload Error |
| 0x0D | Overheat Warning |
| 0x0E | Overheat Error |
| 0x0F | Condensation Warning |
| 0x10 | Condensation Error |
| 0x11 | Vibration Warning |
| 0x12 | Vibration Error |
| 0x13 | Configuration Warning |
| 0x14 | Configuration Error |
| 0x15 | Element Not Calibrated Warning |
| 0x16 | Element Not Calibrated Error |
| 0x17 | Memory Warning |
| 0x18 | Memory Error |
| 0x19 | Self-Test Warning |
| 0x1A | Self-Test Error |
| 0x1B | Input Too Low Warning |
| 0x1C | Input Too Low Error |
| 0x1D | Input Too High Warning |
| 0x1E | Input Too High Error |
| 0x1F | Input No Change Warning |
| 0x20 | Input No Change Error |
| 0x21 | Actuator Blocked Warning |
| 0x22 | Actuator Blocked Error |
| 0x23 | Housing Opened Warning |
| 0x24 | Housing Opened Error |
| 0x25 | Tamper Warning |
| 0x26 | Tamper Error |

| Value | Description |
|---|---|
| 0x27 | Device Moved Warning |
| 0x28 | Device Moved Error |
| 0x29 | Device Dropped Warning |
| 0x2A | Device Dropped Error |
| 0x2B | Overflow Warning |
| 0x2C | Overflow Error |
| 0x2D | Empty Warning |
| 0x2E | Empty Error |
| 0x2F | Internal Bus Warning |
| 0x30 | Internal Bus Error |
| 0x31 | Mechanism Jammed Warning |
| 0x32 | Mechanism Jammed Error |
| 0x33–0x7F | Reserved for Future Use |
| 0x80–0xFF | Vendor Specific Warning / Error |

*Table 4.21: Fault values*

#### 4.2.15.2 Registered Fault

The Registered Fault state is a 1-octet value of the most recently performed self-test and a shadow array of the Current Fault FaultArray. The format of this state is defined in Table 4.22.

| Field | Size | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a most recently performed self-test |
| FaultArray | N | Array of registered faults |

*Table 4.22: Registered Fault format*

Values for the Test ID are defined in Table 4.23.

| Value | Description |
|---|---|
| 0x00 | Standard test |
| 0x01–0xFF | Vendor specific test |

*Table 4.23: Test ID values*

Whenever a fault condition has been present in the Current Fault state (see Section 4.2.15.1), the corresponding record is added to the Registered Fault FaultArray. The FaultArray is cleared with a dedicated Health Fault Clear message (see Section 4.3.3.3).

### 4.2.16 Health Fast Period Divisor

The Health Fast Period Divisor state is a 1-octet value that controls the increased cadence of publishing Health Current Status messages.

The value range for the Health Fast Period Divisor state is 0 through 15, all other values are prohibited. This is used to divide the Health Publish Period by $2^n$ where the n is the value of the Health Fast Period Divisor state.

### 4.2.17 Heartbeat Publication

The Heartbeat Publication state is a composite state that controls sending of periodical Heartbeat transport control messages.

#### 4.2.17.1 Heartbeat Publication Destination

The Heartbeat Publication Destination state determines the destination address for Heartbeat messages. The Heartbeat Publication Destination shall be the unassigned address, a unicast address, or a group address, all other values are Prohibited.

> Note: If the Heartbeat Publication Destination is set to the unassigned address, the Heartbeat messages are not being sent.

#### 4.2.17.2 Heartbeat Publication Count Log

The Heartbeat Publication Count state is a 16-bit value that controls the number of periodical Heartbeat transport control messages to be sent. When set to 0xFFFF, it is not decremented after sending each Heartbeat message. When set to 0x0000, Heartbeat messages are not sent. When set to a value greater than or equal to 0x0001 or less than or equal to 0xFFFE, it is decremented after sending each Heartbeat message. The values for this state are defined in Table 4.24.

The Heartbeat Publication Count Log is a representation of the Heartbeat Publication Count value. The Heartbeat Publication Count Log and Heartbeat Publication Count with the value 0x00 and 0x0000 are equivalent. The Heartbeat Publication Count Log value of 0xFF is equivalent to the Heartbeat Publication count value of 0xFFFF. The Heartbeat Publication Count Log value between 0x01 and 0x11 shall represent that smallest integer n where $2^{(n-1)}$ is greater than or equal to the Heartbeat Publication Count value. For example, if the Heartbeat Publication Count value is 0x0579, then the Heartbeat Publication Count Log value would be 0x0C.

| Value | Description |
|---|---|
| 0x00 | Heartbeat messages are not being sent periodically |
| 0x01–0x11 | Number of Heartbeat messages, $2^{(n-1)}$, that remain to be sent |
| 0x12-0xFE | Prohibited |
| 0xFF | Heartbeat messages are being sent indefinitely |

*Table 4.24: Heartbeat Publication Count Log values*

#### 4.2.17.3 Heartbeat Publication Period Log

The Heartbeat Publication Period Log state is an 8-bit value that controls the cadence of periodical Heartbeat transport control messages. The value is represented as $2^{(n-1)}$ seconds. For example, the value 0x04 would have a publication period of 8 seconds, and the value 0x07 would have a publication period of 64 seconds. The values for this state are defined in Table 4.25.

| Value | Description |
|---|---|
| 0x00 | Heartbeat messages are not being sent periodically |
| 0x01–0x11 | Period in $2^{(n-1)}$ seconds for sending periodical Heartbeat messages |
| 0x12–0xFF | Prohibited |

*Table 4.25: Heartbeat Publication Period Log values*

#### 4.2.17.4 Heartbeat Publication TTL

The Heartbeat Publication TTL state determines the TTL value used when sending Heartbeat messages. The values for this state are defined in Table 4.26.

| Value | Description |
|---|---|
| 0x00–0x7F | The Heartbeat Publication TTL state |
| 0x80–0xFF | Prohibited |

*Table 4.26: Heartbeat Publication TTL values*

#### 4.2.17.5 Heartbeat Publication Features

The Heartbeat Publication Features state determines the features that trigger sending Heartbeat messages when changed. The values for this state are defined in Table 4.27.

| Bit | Feature | Notes |
|---|---|---|
| 0 | Relay | Relay feature change triggers a Heartbeat message: 0 = False, 1 = True |
| 1 | Proxy | Proxy feature change triggers a Heartbeat message: 0 = False, 1 = True |
| 2 | Friend | Friend feature change triggers a Heartbeat message: 0 = False, 1 = True |
| 3 | Low Power | Low Power feature change triggers a Heartbeat message: 0 = False, 1 = True |
| 4 | Directed Forwarding | Directed Forwarding feature change triggers a Heartbeat message: 0 = False, 1 = True |
| 5–15 | RFU | Reserved for Future Use |

*Table 4.27: Heartbeat Publication Feature values*

#### 4.2.17.6 Heartbeat Publication NetKey Index

The Heartbeat Publication NetKey Index state determines the global NetKey Index of the NetKey used to send Heartbeat messages.

### 4.2.18 Heartbeat Subscription

The Heartbeat Subscription state is a composite state that controls receiving of periodical Heartbeat transport control messages.

#### 4.2.18.1 Heartbeat Subscription Source

The Heartbeat Subscription Source state determines the source address for Heartbeat messages a node shall process. The Heartbeat Subscription Source shall be the unassigned address or a unicast address, all other values are Prohibited.

If the Heartbeat Subscription Source is set to the unassigned address, the Heartbeat messages are not being processed.

#### 4.2.18.2 Heartbeat Subscription Destination

The Heartbeat Subscription Destination state determines the destination address for Heartbeat messages. This can be used by nodes to configure a proxy filter to allow them to receive Heartbeat messages, for example, nodes connected using a GATT bearer or in a friendship. The Heartbeat Subscription Destination shall be the unassigned address, the primary unicast address of the node, or a group address, all other values are Prohibited.

If the Heartbeat Subscription Destination is set to the unassigned address, the Heartbeat messages are not being processed.

#### 4.2.18.3 Heartbeat Subscription Count Log

The Heartbeat Subscription Count state is a 16-bit counter that controls the number of periodical Heartbeat transport control messages received since receiving the most recent Config Heartbeat Subscription Set message. The counter stops counting at 0xFFFF. The values for this state are defined in Table 4.28.

The Heartbeat Subscription Count Log is a representation of the Heartbeat Subscription Count value. The Heartbeat Subscription Count Log and Heartbeat Subscription Count with the value 0x00 and 0x0000 are equivalent. The Heartbeat Subscription Count Log value of 0xFF is equivalent to the Heartbeat Subscription count value of 0xFFFF. The Heartbeat Subscription Count Log value between 0x01 and 0x11 shall represent the Heartbeat Subscription Count value, using the transformation defined in Table 4.1.

| Value | Description |
|---|---|
| 0x0000–0xFFFE | Number of Heartbeat messages received |
| 0xFFFF | More than 0xFFFE messages have been received |

*Table 4.28: Heartbeat Subscription Count values*

#### 4.2.18.4 Heartbeat Subscription Period Log

The Heartbeat Subscription Period state is a 16-bit value that controls the period for processing periodical Heartbeat transport control messages. When set to 0x0000, Heartbeat messages are not being processed. When set to a value greater than or equal to 0x0001, Heartbeat messages are being processed. The values for this state are defined in Table 4.29.

The Heartbeat Subscription Period Log is a representation of the Heartbeat Subscription Period value. The Heartbeat Subscription Period Log and Heartbeat Subscription Period with the value 0x00 and 0x0000 are equivalent. The Heartbeat Subscription Period Log value between 0x01 and 0x11 shall represent the Heartbeat Subscription Period value, using the transformation defined in Table 4.1.

| Value | Description |
|---|---|
| 0x00 | Heartbeat messages are not being processed |
| 0x01–0x11 | Remaining period in $2^{(n-1)}$ seconds for processing periodical Heartbeat messages |
| 0x12-0xFF | Prohibited |

*Table 4.29: Heartbeat Subscription Period values*

#### 4.2.18.5 Heartbeat Subscription Min Hops

The Heartbeat Subscription Min Hops state determines the minimum hops value registered when receiving Heartbeat messages since receiving the most recent Config Heartbeat Subscription Set message. The values for this state are defined in Table 4.30.

| Value | Description |
|---|---|
| 0x00 | No Heartbeat messages have been received |
| 0x01–0x7F | The Heartbeat Subscription Min Hops state |
| 0x80–0xFF | Prohibited |

*Table 4.30: Heartbeat Subscription Min TTL values*

#### 4.2.18.6 Heartbeat Subscription Max Hops

The Heartbeat Subscription Max Hops state determines the maximum hops value registered when receiving Heartbeat messages since receiving the most recent Config Heartbeat Subscription Set message. The values for this state are defined in Table 4.31.

| Value | Description |
|---|---|
| 0x00 | No Heartbeat messages have been received. |
| 0x01–0x7F | The Heartbeat Subscription Max Hops state |
| 0x80–0xFF | Prohibited |

*Table 4.31: Heartbeat Subscription Max TTL values*

### 4.2.19 Network Transmit

The Network Transmit state is a composite state that controls the number and timing of the transmissions of Network PDU originating from a node.

The state includes a Network Transmit Count field and a Network Transmit Interval Steps field.

There is a single instance of this state for the node.

#### 4.2.19.1 Network Transmit Count

The Network Transmit Count field is a 3-bit value that controls the number of message transmissions of the Network PDU originating from the node. The number of transmissions is the Transmit Count + 1.

For example a value of 0b000 represents a single transmission and a value of 0b111 represents 8 transmissions.

#### 4.2.19.2 Network Transmit Interval Steps

The Network Transmit Interval Steps field is a 5-bit value representing the number of 10 millisecond steps that controls the interval between message transmissions of Network PDUs originating from the node.

The transmission interval is calculated using the formula:

transmission interval = (Network Retransmit Interval Steps + 1) * 10

Each transmission should be perturbed by a random value between 0 to 10 milliseconds between each transmission.

For example, a value of 0b10000 represents a transmission interval between 170 and 180 milliseconds between each transmission.

### 4.2.20 Relay Retransmit

The Relay Retransmit state is a composite state that controls parameters of retransmission of the Network PDU relayed by the node.

The state includes a Relay Retransmit Count and a Relay Retransmit Interval Steps states.

There is a single instance of this state for the node.

#### 4.2.20.1 Relay Retransmit Count

The Relay Retransmit Count field is a 3-bit value that controls the number of message retransmissions of the Network PDU relayed by the node. The Relay Retransmit Count + 1 is the number of times that packet is transmitted for each packet that is relayed.

For example, a value of 0b000 represents a single transmission with no retransmissions, and a value of 0b111 represents a single transmission and 7 retransmissions for a total of 8 transmissions.

#### 4.2.20.2 Relay Retransmit Interval Steps

The Relay Retransmit Interval Steps field is a 5-bit value representing the number of 10 millisecond steps that controls the interval between message retransmissions of the Network PDU relayed by the node.

The retransmission interval is calculated using the formula:

retransmission interval = (Relay Retransmit Interval Steps + 1) * 10

### 4.2.21 PollTimeout List

The PollTimeout List state is a list of current values of PollTimeout timer of the Low Power nodes within a Friend node.

Each entry in the PollTimeout List holds the current value of the PollTimeout timer. The list is indexed by Low Power node primary element address.

| Value | Description |
|---|---|
| 0x000000 | The node is no longer a Friend node of the Low Power node identified by the LPNAddress |
| 0x000001 through 0x000009 | Prohibited |
| 0x00000A through 0x34BBFF | The PollTimeout timer value in units of 100 milliseconds |
| 0x34BC00 through 0xFFFFFF | Prohibited |

*Table 4.32: PollTimeout Timer values*

If the Friend feature is not supported or the Friend feature is supported and disabled, the current value of the PollTimeout List state for any Low Power node shall be set to 0x000000.

If the Friend feature is supported and enabled and the Friend node has not established friendship with the Low Power node identified by a primary element address, the current value of the PollTimeout List state for that Low Power node shall be set to 0x000000.

### 4.2.22 Directed Forwarding

The Directed Forwarding state controls whether the Directed Forwarding feature is enabled or disabled for a given subnet. The values are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The Directed Forwarding feature is disabled for a subnet |
| 0x01 | The Directed Forwarding feature is enabled for a subnet |

| | |
|---|---|
| 0x02–0xFF | Prohibited |

*Table 4.33: Directed Forwarding values*

The default value of the Directed Forwarding state is 0x00.

## 4.2.23 Path Metric

The Path Metric state is a composite state that controls the type of path metric to be used to evaluate the quality of a non-fixed path in a subnet and the duration of non-fixed paths established in the subnet. It includes the Path Metric Type state and the Path Lifetime state.

### 4.2.23.1 Path Metric Type

The Path Metric Type state is a 1-octet value that represents the type of path metric to be used to evaluate the quality of a non-fixed path in a subnet and to rank the path accordingly when the node selects the best path toward a given Path Destination.The values of the Path Metric Type state are defined in **錯誤! 找不到參照來源。**.

| Value | Description |
|---|---|
| 0x00 | The node uses the hop-count metric with random path selection |
| 0x01 | The node uses the hop-count metric with average RSSI-based path selection |
| 0x02–0xFF | Reserved for Future Use |

*Table 4.34: Path Metric Type values*

The default value of the Path Metric Type state is 0x00.

If the Path Metric Type state is set to 0x00, then:

- The Path Metric is 1 octet in size, counts the number of hops in a path, and can range from 0 to the maximum TTL value (that is, 0x7F).
- The Node Metric of each node in the path is equal to 0, and the Link Metric of each link in the path is equal to 1, as introduced in Section **錯誤! 找不到參照來源。**.
- The Path Metric calculated at N+1 hops from the Path Originator shall be equal to the Path Metric calculated at N hops incremented by 1.
- When selecting the best path, one of the paths with the lowest hop-count is randomly chosen.

If the Path Metric Type state is set to 0x01, then:

- The Path Metric is 2 octets in size. The least significant octet stores the count of the number of hops in a path, which can range from 0 to the maximum TTL value (that is, 0x7F). The most significant octet stores the average RSSI along the path, and is a signed integer from -127 to 127.

- The Node Metric of each node in the path is equal to 0, and the Link Metric of each link in the path is 2 octets in size, according to the Path Metric introduced in Section 錯誤! 找不到參照來源。, with the least significant octet equal to 1, and the most significant octet set to the RSSI measured on the receiver side of the link.

  Note: To improve the accuracy of the link quality estimation, multiple RSSI measurements should be taken at the receiver for the same link. The RSSI field in the Neighbor Table may be used for this purpose (see Section 3.6.8.5).

- The least significant octet of the Path Metric calculated at N+1 hops from the path originator shall be equal to the least significant octet of the Path Metric calculated at N hops incremented by 1. The most significant octet of the Path Metric calculated at N+1 hops ($averageRSSI_{N+1}$) depends on the most significant octet of the Path Metric calculated at N hops ($averageRSSI_{N}$) and on the RSSI measured at N+1 hops ($RSSI_{N+1}$), according to the following equation (EQ1). The result is rounded down to the nearest integer.

$$averageRSSI_{N+1} = \frac{averageRSSI_N \times N + RSSI_{N+1}}{N+1} \quad (EQ1)$$

  When selecting the best path from among many paths, the path with the lowest hop-count is chosen. In case of hop-count equality, the path with the highest average RSSI is chosen from among the paths with the lowest hop count.

### 4.2.23.2 Path Lifetime

The Path Lifetime state is a 2-octet value that determines how long a non-fixed path in a subnet is valid. While a non-fixed path is valid, the path information is stored in a non-fixed path entry in the Forwarding Table state of the subnet and can be used to forward messages over the path. When this time elapses, the path entry is deleted from the Forwarding Table state.

The format of the Path Lifetime state is defined in 錯誤! 找不到參照來源。錯誤! 找不到參照來源。.

| Field | Size (bits) | Notes |
|---|---|---|
| Time Unit | 2 | Binary code associated with the time unit that is used to express the value of the Path Lifetime state:<br>• 0b00 indicates 1 second.<br>• 0b01 indicates 1 minute.<br>• 0b10 indicates 1 hour.<br>• 0b11 indicates one day. |
| Time Value | 14 | The value of the Path Lifetime state expressed as a multiple of the selected time unit. |

*Table 4.35: Path Lifetime format*

The default value of the Path Lifetime state is 0xC007 (i.e., 7 days), or 0b11 for Time Unit and 0x0007 for Time Value.

0x0000 is a Prohibited value for the Path Lifetime state.

### 4.2.24 Path Request Transmit

The Path Request Transmit state is a composite state that controls the maximum number of concurrent Directed Forwarding Initialization operations being performed on a node in a given subnet and the minimum interval between two concurrent operations directed to the same Path Destination. It includes the Concurrent Initialization state and the Path Request Interval state.

#### 4.2.24.1 Concurrent Initialization

The Concurrent Initialization state is a 1-octet value that represents the maximum number of concurrent Directed Forwarding Initialization operations in a node in a given subnet.

The default value of the Concurrent Initialization state is 0x10.

#### 4.2.24.2 Path Request Interval

The Path Request Interval state is a 1-octet value that represents the minimum interval between two concurrent Directed Forwarding Initialization operations directed to the same Path Destination.

The values of the Path Request Interval state are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The Path Request Interval state is set to 30 seconds. |
| 0x01 | The Path Request Interval state is set to 5 minutes. |
| 0x02 | The Path Request Interval state is set to 1 hour. |
| 0x03–0xFF | Reserved for Future Use |

*Table 4.36: Path Request Interval values*

The default value of the Path Request Interval state is 0x00.

### 4.2.25 Neighbor Information

The Neighbor Information state is a composite state that controls the Neighbor Information Management operation. It includes the Neighborhood state and the Neighbor Information Options state.

#### 4.2.25.1 Neighborhood

The Neighborhood state indicates the maximum hop-wise distance between the node and another node within its neighborhood in a given subnet. Paths toward nodes in the neighborhood are established with the reception of NINFO messages. The values are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The node is not configured to establish path to neighbors in a subnet through receipt of NINFO messages. |
| 0x01 | Neighboring nodes are all nodes reachable in a single hop in a subnet. |
| 0x02 | Neighboring nodes are all nodes reachable in two hops in a subnet. |
| 0x03–0xFF | Prohibited |

*Table 4.37: Neighborhood values*

The default value is 0x01.

#### 4.2.25.2 Neighbor Information Options

The Neighbor Information Options state determines which options for transmission and processing of NINFO messages are used in a subnet, as defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | Periodic NINFO transmission and processing are not enabled in a subnet |
| 0x01 | Periodic NINFO transmission is enabled in a subnet, but received NINFO messages are not used to establish and maintain paths in the subnet |
| 0x02 | Periodic NINFO transmission is disabled in a subnet, but received NINFO messages are used to establish and maintain paths in the subnet |
| 0x03 | Periodic NINFO transmission is enabled in a subnet, and received NINFO messages are used to establish and maintain paths in the subnet |
| 0x04–0xFF | Prohibited |

*Table 4.38: Neighbor Information Options values*

The default value is 0x00.

If the Neighborhood state (see Section 錯誤! 找不到參照來源。) is set to 0x00, the values 0x02 and 0x03 are Prohibited.

### 4.2.26 Forwarding Table

The Forwarding Table state is a list of entries associated with existing paths toward all Path Destinations within a given subnet. Each node contains a Forwarding Table for each subnet to which the node belongs.

Each entry in a Forwarding Table shall have the format defined in 錯誤! 找不到參照來源。.

| Field | Size (octets) | Notes |
|---|---|---|
| Fixed | 1 | Flag indicating whether or not the path is fixed. |
| BackwardPathValidated | 1 | Flag indicating whether or not the backward path is validated. |
| ProactivePathUpdated | 1 | Flag indicating whether or not the entry has been updated by a received NINFO message. |
| PO | 2 | Primary unicast address of the Path Originator, if the Path Originator is present; otherwise, set to the unassigned address and ignored. |
| DPO | 2 | Primary unicast address of the Delegate Path Originator, if a Delegate Path Originator is present; otherwise, set to the unassigned address and ignored. |
| FN | 1 | Last Forwarding Number known to have been generated by either the Path Originator or the Delegate Path Originator, if either of them is present, or by either the Path Destination or the Delegate Path Destination; if the entry is associated with a fixed path, it is set to 0 and ignored. |
| PD | 2 | Primary unicast address of the Path Destination. |
| DPD | 2 | Primary unicast address of the Delegate Path Destination, if a Delegate Path Destination present; otherwise, set to the unassigned address and ignored. |
| DHC | 1 | Hop count from either the Path Destination or the Delegate Path Destination, if the Forwarding Table entry is associated with a non-fixed path; otherwise, set to 0 and ignored. The field value is greater than 0 only if the Path Destination is in the neighborhood (see Section 錯誤! 找不到參照來源。). |
| OAR | 1 | Range of secondary element addresses known to be assigned to the Path Originator or to the Low Power node associated with the Delegate Path Originator, if either of them is present; otherwise, set to 0xFF and ignored. |
| DAR | 1 | Range of secondary element addresses known to be assigned to the Path Destination or to the Low Power node associated with the Delegate Path Destination. |
| NTO | 2 | Primary unicast address of the next-hop node toward the Path Originator, if present; otherwise, set to the unassigned address and ignored. |
| NTD | 2 | Primary unicast address of the next-hop node toward the |

| Field | Size (octets) | Notes |
|---|---|---|
| | | Path Destination, if present; otherwise, set to the unassigned address and ignored. |
| BTO | 1 | The index of the bearer to be used for forwarding messages directed to the Path Originator. |
| BTD | 1 | The index of the bearer to be used for forwarding messages directed to the Path Destination. |
| Path Expiration Time | 2 | Length of time during which the entry is considered valid, if the path is not a fixed path; otherwise, set to 0 and ignored. |
| Subscription List | Variable (2 * N) | List of group addresses and virtual addresses (N in total) that the Path Destination or the Low Power node associated with the Delegate Path Destination is subscribed to. This is a subset of the node's Subscription List state containing only group addresses and virtual addresses discovered by the Directed Forwarding feature. |

*Table 4.39: Forwarding Table entry format*

The values of the Fixed field are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The path is non-fixed. |
| 0x01 | The path is fixed. |
| 0x02–0xFF | Prohibited |

*Table 4.40: Field values for the Fixed field (Forwarding Table)*

The values of the BackwardPathValidated field are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The backward path is not validated. |
| 0x01 | The backward path is validated. |
| 0x02–0xFF | Prohibited |

*Table 4.41: BackwardPathValidated field values (Forwarding Table)*

The values of the ProactivePathUpdated field are defined in 錯誤! 找不到參照來源。.

| Value | Description |
|---|---|
| 0x00 | The entry is not updated after processing a NINFO. |
| 0x01 | The entry is updated after processing a NINFO. |
| 0x02–0xFF | Prohibited |

*Table 4.42: ProactivePathUpdated field values (Forwarding Table)*

The format of the Path Expiration Time field is the same as the format of the Path Lifetime state defined in **錯誤! 找不到參照來源。**. The Forwarding Table state is empty by default.

## 4.3 Message definitions

This section defines messages used throughout this specification. Each message has an opcode and zero or more parameters, as defined in Section 3.7.3. Messages are also defined as either unacknowledged or acknowledged. Acknowledged messages have defined responses that shall be used to confirm execution.

Message definitions that are not required as part of this specification are defined in the Mesh Model specification [11] and follow the same format and architecture as mesh message definitions.

### 4.3.1 Supplemental parameter requirements

This section contains supplemental requirements for the handling of some parameters. Parameter values that do not conform to these requirements shall be considered Reserved for Future Use.

#### 4.3.1.1 Key indexes

Both NetKeys and AppKeys are 16-octets long and thus do not fit in a single-segment message. The generic segmentation and reassembly mechanism is used to transport new keys to nodes.

A Configuration Client maintains two indexed global lists of NetKeys and AppKeys – the NetKey List and AppKey List. Each key shall have a unique global index in the appropriate list and this index is used to identify that key in the module.

Global key indexes are 12 bits long. Some messages include one, two or multiple key indexes. To enable efficient packing, two key indexes are packed into three octets. Where an odd number of key indexes need to be packed, all but the last key index are packed into sequences of three octets (see Figure 4.3), and the last key index is packed into two octets (see Figure 4.4). Where an even number of key indexes need to be packed, they are all packed into sequences of three octets.

To pack two key indexes into three octets, 8 LSbs of first key index value are packed into the first octet, placing the remaining 4 MSbs into 4 LSbs of the second octet. The first 4 LSbs of the second 12-bit key index are packed into the 4 MSbs of the second octet with the remaining 8 MSbs into the third octet.

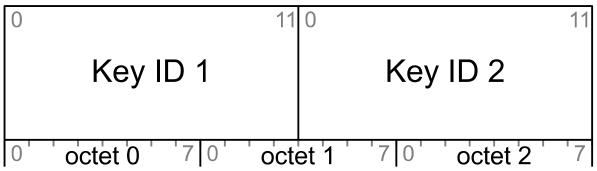


*Figure 4.3: Packing of two 12-bit key Indexes into three octets*

To pack one key index into two octets, 8 LSbs of first key index value are packed into the first octet, placing the remaining 4 MSbs into 4 LSbs of the second octet, and the 4 MSbs of the second octet shall be set to 0.

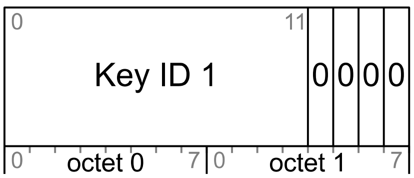


*Figure 4.4: Encoding of one 12-bit key index into two octets*

### 4.3.1.2 Element addresses

Messages may contain fields carrying network addresses, such as an element address, a group address, or a virtual address. Such fields may be used by the access layer and other upper layers, but shall not be used by the network layer, the lower transport layer, nor the upper transport layer.

### 4.3.1.3 Model identifiers

Messages may contain fields carrying Model IDs. Such fields may have a 2-octet size or a 4-octet size to accommodate either a SIG Model ID (16-bit) or a Vendor Model ID (32-bit).

### 4.3.1.4 Variable length parameters

When a variable length field is used, it shall be the last field for a message, and the size of the field is determined based on the message length information provided by lower layers. There shall only be one sequence of consecutive optional fields or a variable length parameter.

### 4.3.1.5 Optional parameters

When optional fields are used, they shall be the last fields for a message and the presence of the fields is determined based on the message length information provided by lower layers. There shall only be one sequence of consecutive optional fields or a variable length parameter.

## 4.3.2 Configuration messages

Configuration messages are used to control states that determine network-related behaviors of the node, manipulate network and application keys, as well as perform other operations that require an elevated level of security. Every configuration message shall be encrypted and authenticated using a DevKey.

Because DevKeys are unique for every node, configuration messages shall be sent only to unicast addresses.

#### 4.3.2.1 Config Beacon Get

The Config Beacon Get is an acknowledged message used to get the current Secure Network Beacon state of a node (see Section 4.2.10).

The response to a Config Beacon Get message is a Config Beacon Status message.

There are no Parameters for this message.

#### 4.3.2.2 Config Beacon Set

The Config Beacon Set is an acknowledged message used to set the Secure Network Beacon state of a node (see Section 4.2.10).

The response to a Config Beacon Set message is a Config Beacon Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Beacon | 1 | New Secure Network Beacon state |

*Table 4.43: Config Beacon Set message parameters*

The Beacon field shall provide the new Secure Network Beacon state of the node (see Section 4.2.10).

#### 4.3.2.3 Config Beacon Status

The Config Beacon Status is an unacknowledged message used to report the current Secure Network Beacon state of a node (see Section 4.2.10).

| Field | Size (octets) | Notes |
|---|---|---|
| Beacon | 1 | Secure Network Beacon state |

*Table 4.44: Config Beacon Status message parameters*

The Beacon field shall provide the current Secure Network Beacon state of the node (see Section 4.2.10).

#### 4.3.2.4 Config Composition Data Get

The Config Composition Data Get is an acknowledged message used to read one page of the Composition Data (see Section 4.2.1).

The response to a Config Composition Data Get message is a Config Composition Data Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Page | 1 | Page number of the Composition Data |

*Table 4.45: Config Composition Data Get message parameters*

The Page field shall identify the Composition Data Page number that is being read.

#### 4.3.2.5 Config Composition Data Status

The Config Composition Data Status is an unacknowledged message used to report a single page of the Composition Data (see Section 4.2.1).

This message uses a single octet opcode to maximize the size of a payload.

| Field | Size (octets) | Notes |
|---|---|---|
| Page | 1 | Page number of the Composition Data |
| Data | variable | Composition Data for the identified page |

*Table 4.46: Config Composition Data Status message parameters*

The Page field shall identify the Composition Data Page number.

The Data field shall contain the identified single page of the Composition Data.

#### 4.3.2.6 Config Default TTL Get

The Config Default TTL Get is an acknowledged message used to get the current Default TTL state of a node.

The response to a Config Default TTL Get message is a Config Default TTL Status message.

There are no Parameters for this message.

#### 4.3.2.7 Config Default TTL Set

The Config Default TTL Set is an acknowledged message used to set the Default TTL state of a node (see Section 4.2.7).

The response to a Config Default TTL Set message is a Config Default TTL Status message.

| Parameter | Size (octets) | Notes |
|---|---|---|
| TTL | 1 | New Default TTL value |

*Table 4.47: Config Default TTL Set message parameters*

The TTL field shall identify a new Default TTL for the node (see Section 4.2.7).

#### 4.3.2.8 Config Default TTL Status

The Config Default TTL Status is an unacknowledged message used to report the current Default TTL state of a node (see Section 4.2.7).

| Parameters | Size (octets) | Notes |
|---|---|---|
| TTL | 1 | Default TTL |

*Table 4.48: Config Default TTL Status message parameters*

The TTL field shall identify the Default TTL for the node, as defined in Default TTL (see Section 4.2.7).

#### 4.3.2.9 Config GATT Proxy Get

The Config GATT Proxy Get is an acknowledged message used to get the current GATT Proxy state of a node (see Section 4.2.11).

The response to a Config GATT Proxy Get message is a Config GATT Proxy Status message.

There are no Parameters for this message.

#### 4.3.2.10 Config GATT Proxy Set

The Config GATT Proxy Set is an acknowledged message used to set the GATT Proxy state of a node (see Section 4.2.11).

The response to a Config GATT Proxy Set message is a Config GATT Proxy Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| GATTProxy | 1 | New GATT Proxy state |

*Table 4.49: Config GATT Proxy Set message parameters*

The GATTProxy field shall provide the new GATT Proxy state of the node (see Section 4.2.11).

#### 4.3.2.11 Config GATT Proxy Status

The Config GATT Proxy Status is an unacknowledged message used to report the current GATT Proxy state of a node (see Section 4.2.11).

| Field | Size (octets) | Notes |
|---|---|---|
| GATTProxy | 1 | GATT Proxy state |

*Table 4.50: Config GATT Proxy Status message parameters*

The GATTProxy field shall provide the current GATT Proxy state of the node (see Section 4.2.11).

#### 4.3.2.12 Config Relay Get

The Config Relay Get is an acknowledged message used to get the current Relay (see Section 4.2.8) and Relay Retransmit (see Section 4.2.20) states of a node.

The response to a Config Relay Get message is a Config Relay Status message.

There are no Parameters for this message.

#### 4.3.2.13 Config Relay Set

The Config Relay Set is an acknowledged message used to set the Relay (see Section 4.2.8) and Relay Retransmit (see Section 4.2.20) states of a node.

The response to a Config Relay Set message is a Config Relay Status message.

| Field | Size (bits) | Notes |
|---|---|---|
| Relay | 8 | Relay |
| RelayRetransmitCount | 3 | Number of retransmissions on advertising bearer for each Network PDU relayed by the node |
| RelayRetransmitIntervalSteps | 5 | Number of 10-millisecond steps between retransmissions |

*Table 4.51: Config Relay Set message parameters*

The Relay field shall identify the new Relay state for the node, as defined in Section 4.2.8.

The RelayRetransmitCount field shall contain a new value for the Relay Retransmit Count state of a node (see Section 4.2.20.1).

The RelayRetransmitIntervalSteps field shall contain a new value for the Relay Retransmit Interval Steps state of a node (see Section 4.2.20.2).

#### 4.3.2.14 Config Relay Status

The Config Relay Status is an unacknowledged message used to report the current Relay (see Section 4.2.8) and Relay Retransmit (see Section 4.2.20) states of a node.

| Field | Size (bits) | Notes |
|---|---|---|
| Relay | 8 | Relay |
| RelayRetransmitCount | 3 | Number of retransmissions on advertising bearer for each Network PDU relayed by the node |
| RelayRetransmitIntervalSteps | 5 | Number of 10-millisecond steps between retransmissions |

*Table 4.52: Config Relay Status message parameters*

The Relay field shall identify the current Relay state for the node, as defined in Section 4.2.8.

The RelayRetransmitCount field shall contain a new value for the Relay Retransmit Count state of a node (see Section 4.2.20.1).

The RelayRetransmitIntervalSteps field shall contain a new value for the Relay Retransmit Interval Steps state of a node (see Section 4.2.20.2).

#### 4.3.2.15 Config Model Publication Get

The Config Model Publication Get is an acknowledged message used to get the publish address and parameters of an outgoing message that originates from a model.

The response to a Config Model Publication Get message is a Config Model Publication Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.53: Config Model Publication Get message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.16 Config Model Publication Set

The Config Model Publication Set is an acknowledged message used to set the Model Publication state (see Section 4.2.2) of an outgoing message that originates from a model.

The response to a Config Model Publication Set message is a Config Model Publication Status message.

The Config Model Publication Set message uses a single octet opcode to maximize the size of a payload.

| Field | Size (bits) | Notes |
|---|---|---|
| ElementAddress | 16 | Address of the element |
| PublishAddress | 16 | Value of the publish address |
| AppKeyIndex | 12 | Index of the application key |
| CredentialFlag | 1 | Value of the Friendship Credential Flag |
| RFU | 3 | Reserved for Future Use |
| PublishTTL | 8 | Default TTL value for the outgoing messages |
| PublishPeriod | 8 | Period for periodic status publishing |
| PublishRetransmitCount | 3 | Number of retransmissions for each published message |
| PublishRetransmitIntervalSteps | 5 | Number of 50-millisecond steps between retransmissions |
| ModelIdentifier | 16 or 32 | SIG Model ID or Vendor Model ID |

*Table 4.54: Config Model Publication Set message parameters*

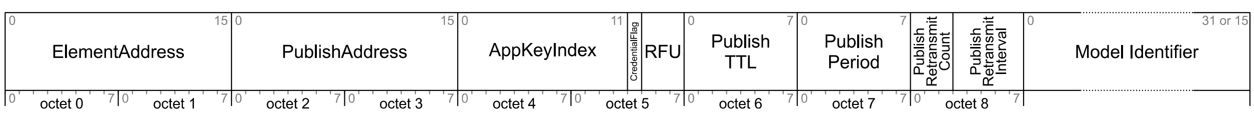


*Figure 4.5: Config Model Publication Set format*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The PublishAddress field shall contain the new Publish Address state (see Section 4.2.2.1) for the model. The value of PublishAddress field shall not be a virtual address.

The AppKeyIndex field shall contain the new Publish AppKey Index state (see Section 4.2.2.3).

The CredentialFlag field shall contain the new Publish Friendship Credentials Flag state (see Section 4.2.2.4).

The PublishTTL field shall contain the new Publish TTL state (see Section 4.2.2.5).

The PublishPeriod field shall contain a new value for the Publish Period state (see Section 4.2.2.2).

The PublishRetransmitCount field shall contain a new value for the Publish Retransmit Count state of an element (see Section 4.2.2.6).

The PublishRetransmitIntervalSteps field shall contain a new value for the Publish Retransmit Interval Steps state of an element (see Section 4.2.2.7).

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.17 Config Model Publication Virtual Address Set

The Config Model Publication Virtual Address Set is an acknowledged message used to set the model Publication state (see Section 4.2.2) of an outgoing message that originates from a model.

The response to a Config Model Publication Virtual Address Set message is a Config Model Publication Status message.

| Field | Size (bits) | Notes |
|---|---|---|
| ElementAddress | 16 | Address of the element |
| PublishAddress | 128 | Value of the Label UUID publish address |
| AppKeyIndex | 12 | Index of the application key |
| CredentialFlag | 1 | Value of the Friendship Credential Flag |
| RFU | 3 | Reserved for Future Use |
| PublishTTL | 8 | Default TTL value for the outgoing messages |
| PublishPeriod | 8 | Period for periodic status publishing |
| PublishRetransmitCount | 3 | Number of retransmissions for each published message |
| PublishRetransmitIntervalSteps | 5 | Number of 50-millisecond steps between retransmissions |
| ModelIdentifier | 16 or 32 | SIG Model ID or Vendor Model ID |

*Table 4.55: Config Model Publication Virtual Address Set message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The PublishAddress field shall contain the virtual address used as new Publish Address state (see Section 4.2.2.1) for the model.

The AppKeyIndex field shall contain the new Publish AppKey Index state (see Section 4.2.2.3).

The CredentialFlag field shall contain the new Publish Friendship Credentials Flag state (see Section 4.2.2.4).

The PublishTTL field shall contain the new Publish TTL state (see Section 4.2.2.5).

The PublishPeriod field shall contain a new value for the Publish Period state (see Section 4.2.2.2).

The PublishRetransmitCount field shall contain a new value for the Publish Retransmit Count state of an element (see Section 4.2.2.6).

The PublishRetransmitIntervalSteps field shall contain a new value for the Publish Retransmit Interval Steps state of an element (see Section 4.2.2.7).

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.18 Config Model Publication Status

The Config Model Publication Status is an unacknowledged message used to report the model Publication state (see Section 4.2.2) of an outgoing message that is published by the model.

| Field | Size (bits) | Notes |
|---|---|---|
| Status | 8 | Status Code for the requesting message |
| ElementAddress | 16 | Address of the element |
| PublishAddress | 16 | Value of the publish address |
| AppKeyIndex | 12 | Index of the application key |
| CredentialFlag | 1 | Value of the Friendship Credential Flag |
| RFU | 3 | Reserved for Future Use |
| PublishTTL | 8 | Default TTL value for the outgoing messages |
| PublishPeriod | 8 | Period for periodic status publishing |
| PublishRetransmitCount | 3 | Number of retransmissions for each published message |
| PublishRetransmitIntervalSteps | 5 | Number of 50-millisecond steps between retransmissions |
| ModelIdentifier | 16 or 32 | SIG Model ID or Vendor Model ID |

*Table 4.56: Config Model Publication Status message parameters*

The Status field shall identify the Status Code for the last operation on Config Model Publication parameters. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field shall contain the unicast address of the element, all other address types are Prohibited.

The PublishAddress field shall contain the current Publish Address for the model. When using a Label UUID, the status message shall provide this value as the virtual address as defined in Section 3.4.2.3.

The AppKeyIndex is a global AppKey Index of the AppKey.

The CredentialFlag field shall contain the current value Publish Friendship Credentials Flag state (see Section 4.2.2.4).

The PublishTTL field shall contain the current value of the Publish TTL state (see Section 4.2.2.5) for outgoing messages published by the model within the element.

The PublishPeriod field shall contain the current value for the Publish Period state (see Section 4.2.2.2) for outgoing messages published by the model within the element.

The PublishRetransmitCount field shall contain a new value for the Publish Retransmit Count state of an element (see Section 4.2.2.6).

The PublishRetransmitIntervalSteps field shall contain a new value for the Publish Retransmit Interval Steps state of an element (see Section 4.2.2.7).

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.19 Config Model Subscription Add

The Config Model Subscription Add is an acknowledged message used to add an address to a Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Add message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Address | 2 | Value of the address |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.57: Config Model Subscription Add message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Address field shall contain the new address to be added to the Subscription List. The value of the Address field shall not be an unassigned address, unicast address, all-nodes address or virtual address.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.20 Config Model Subscription Virtual Address Add

The Config Model Subscription Virtual Address Add is an acknowledged message used to add an address to a Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Virtual Address Add message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Label | 16 | Value of the Label UUID |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.58: Config Model Subscription Virtual Address Add message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Label field shall contain the Label UUID to be added to the Subscription List.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

### 4.3.2.21 Config Model Subscription Delete

The Config Model Subscription Delete is an acknowledged message used to delete a subscription address from the Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Delete message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Address | 2 | Value of the Address |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.59: Config Model Subscription Delete message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Address field shall identify the address to be removed from the Subscription List. The value of the Address field shall not be an unassigned address, unicast address, all-nodes address or virtual address.

The ModelIdentifier field either is a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

### 4.3.2.22 Config Model Subscription Virtual Address Delete

The Config Model Subscription Virtual Address Delete is an acknowledged message used to delete a subscription address from the Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Virtual Address Delete message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Address | 16 | Value of the Label UUID |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.60: Config Model Subscription Virtual Address Delete message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Address field shall contain the Label UUID used to identify the Address to be removed from the Subscription List.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.23 Config Model Subscription Overwrite

The Config Model Subscription Overwrite is an acknowledged message used to discard the Subscription List and add an address to the cleared Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Overwrite message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Address | 2 | Value of the Address |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.61: Config Model Subscription Overwrite message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Address field shall contain the new address to be added to the Subscription List. The value of the Address field shall not be an unassigned address, unicast address, all-nodes address or virtual address.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.24 Config Model Subscription Virtual Address Overwrite

The Config Model Subscription Virtual Address Overwrite is an acknowledged message used to discard the Subscription List and add an address to the cleared Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Virtual Address Overwrite message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| Address | 16 | Value of the Label UUID |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.62: Config Model Subscription Virtual Address Overwrite message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The Address field shall contain the Label UUID used as the new Address to be added to the Subscription List.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.25 Config Model Subscription Delete All

The Config Model Subscription Delete All is an acknowledged message used to discard the Subscription List of a model (see Section 4.2.3).

The response to a Config Model Subscription Delete All message is a Config Model Subscription Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.63: Config Model Subscription Delete All message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.26 Config Model Subscription Status

The Config Model Subscription Status is an unacknowledged message used to report a status of the operation on the Subscription List (see Section 4.2.3).

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message. |
| ElementAddress | 2 | Address of the element |
| Address | 2 | Value of the address |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.64: Config Model Subscription Status message parameters*

The Status field shall identify the Status Code for the last operation on the Subscription List. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The value of the Address field shall contain the address that was used to modify the Subscription List or the unassigned address. When referencing the Label UUID, the virtual address shall be used. The value of the Address field shall not be a unicast address or the all-nodes address.

The ModelIdentifier field is a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.27 Config SIG Model Subscription Get

The Config SIG Model Subscription Get is an acknowledged message used to get the list of subscription addresses of a model within the element. This message is only for SIG Models.

The response to a Config SIG Model Subscription Get message is a Config SIG Model Subscription List message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 | SIG Model ID |

*Table 4.65: Config SIG Model Subscription Get message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a SIG Model ID that shall identify the model within the element.

#### 4.3.2.28 Config SIG Model Subscription List

The Config SIG Model Subscription List is an unacknowledged message used to report all addresses from the Subscription List of the model (see Section 4.2.3). This message is only for SIG Models.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 | SIG Model ID |
| Addresses | variable | A block of all addresses from the Subscription List |

*Table 4.66: Config SIG Model Subscription List message parameters*

The Status field shall identify the Status Code for the last operation on the Subscription List. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a SIG Model ID that shall identify the model within the element.

The Addresses field shall identify all addresses from the Subscription List of an element. When using a Label UUID, the status message shall provide the value of the virtual address as defined in Section 3.4.2.3. The empty Subscription List results in Address field of zero length.

#### 4.3.2.29 Config Vendor Model Subscription Get

The Config Vendor Model Subscription Get is an acknowledged message used to get the list of subscription addresses of a model within the element. This message is only for Vendor Models.

The response to a Config Vendor Model Subscription Get message is a Config Vendor Model Subscription List message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 4 | Vendor Model ID |

*Table 4.67: Config Vendor Model Subscription Get message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.30 Config Vendor Model Subscription List

The Config Vendor Model Subscription List is an unacknowledged message used to report all addresses from the Subscription List of the model (see Section 4.2.3). This message is only for Vendor Models.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 4 | Vendor Model ID |
| Addresses | variable | A block of all addresses from the Subscription List |

*Table 4.68: Config Vendor Model Subscription List message parameters*

The Status field shall identify the Status Code for the last operation on the Subscription List. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a Vendor Model ID that shall identify the model within the element.

The Addresses field shall identify all addresses from the Subscription List of an element. When using a Label UUID, the status message shall provide the value of the virtual address as defined in Section 3.4.2.3. The empty Subscription List results in Address field of zero length.

#### 4.3.2.31 Config NetKey Add

The Config NetKey Add is an acknowledged message used to add a NetKey to a NetKey List (see Section 4.2.4) on a node. The added NetKey is then used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends.

The response to a Config NetKey Add message is a Config NetKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | NetKey Index |
| NetKey | 16 | NetKey |

*Table 4.69: Config NetKey Add message parameters*

The NetKeyIndex field shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The NetKey field shall contain the NetKey.

#### 4.3.2.32 Config NetKey Update

The Config NetKey Update is an acknowledged message used to update a NetKey on a node. The updated NetKey is then used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends, as defined by the Key Refresh procedure (see Section 3.10.4).

The response to a Config NetKey Update message is a Config NetKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | Index of the NetKey |
| NetKey | 16 | NetKey |

*Table 4.70: Config NetKey Update message parameters*

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The NetKey field shall contain the NetKey.

#### 4.3.2.33 Config NetKey Delete

The Config NetKey Delete is an acknowledged message used to delete a NetKey on a NetKey List from a node.

The response to a Config NetKey Delete message is a Config NetKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | Index of the NetKey |

*Table 4.71: Config NetKey Delete message parameters*

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.34 Config NetKey Status

The Config NetKey Status is an unacknowledged message used to report the status of the operation on the NetKey List.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| NetKeyIndex | 2 | Index of the NetKey |

*Table 4.72: Config NetKey Status message parameters*

The Status field shall identify the Status Code for the last operation on the NetKey List. The allowed values for Status codes and their meanings are documented in Section 4.3.6. The Status Code shall be Success if the received request was redundant (add of an identical existing key, update of an identical updated key, or delete of a non-existent key), with no further action taken.

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.35 Config NetKey Get

The Config NetKey Get is an acknowledged message used to report all NetKeys known to the node.

The response to a Config NetKey Get message is a Config NetKey List message.

There are no Parameters for this message.

#### 4.3.2.36 Config NetKey List

The Config NetKey List is an unacknowledged message reporting all NetKeys known to the node.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndexes | variable | A list of NetKey Indexes known to the node |

*Table 4.73: Config NetKey List message parameters*

The NetKeyIndexes field shall contain all NetKey Indexes that are known to the node. The NetKey Indexes shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.37 Config AppKey Add

The Config AppKey Add is an acknowledged message used to add an AppKey to the AppKey List on a node and bind it to the NetKey identified by NetKeyIndex. The added AppKey can be used by the node only as a pair with the specified NetKey. The AppKey is used to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends.

The response to a Config AppKey Add message is a Config AppKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndexAndAppKeyIndex | 3 | Index of the NetKey and index of the AppKey |
| AppKey | 16 | AppKey value |

*Table 4.74: Config AppKey Add message parameters*

The NetKeyIndexAndAppKeyIndex field contains two indexes that shall identify the global NetKey Index of the NetKey and the global AppKey Index of the AppKey. These two indexes shall be encoded as defined in Section 4.3.1.1 using NetKey Index as first key index and AppKey Index as second key index.

The AppKey field shall contain the AppKey value identified by the AppKeyIndex.

#### 4.3.2.38 Config AppKey Update

The Config AppKey Update is an acknowledged message used to update an AppKey value on the AppKey List on a node. The updated AppKey is used by the node to authenticate and decrypt messages it receives, as well as authenticate and encrypt messages it sends, as defined by the Key Refresh procedure (see Section 3.10.4).

The response to an Config AppKey Update message is an Config AppKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndexAndAppKeyIndex | 3 | Index of the NetKey and index of the AppKey |
| AppKey | 16 | New AppKey value |

*Table 4.75: Config AppKey Update message parameters*

The NetKeyIndexAndAppKeyIndex field contains two indexes that shall identify the global NetKey Index of the NetKey and the global AppKey Index of the AppKey. These two indexes shall be encoded as defined in Section 4.3.1.1 using NetKey Index as first key index and AppKey Index as second key index. The AppKeyIndex shall be bound to the NetKeyIndex.

The AppKey field shall contain the new value of the AppKey, identified by the AppKeyIndex.

#### 4.3.2.39 Config AppKey Delete

The Config AppKey Delete is an acknowledged message used to delete an AppKey from the AppKey List on a node.

The response to a Config AppKey Delete message is a Config AppKey Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndexAndAppKeyIndex | 3 | Index of the NetKey and index of the AppKey |

*Table 4.76: Config AppKey Delete message parameters*

The NetKeyIndexAndAppKeyIndex field contains two indexes that shall identify the global NetKey Index of the NetKey and the global AppKey Index of the AppKey. These two indexes shall be encoded as defined in Section 4.3.1.1 using NetKey Index as first key index and AppKey Index as second key index.

#### 4.3.2.40 Config AppKey Status

The Config AppKey Status is an unacknowledged message used to report a status for the requesting message, based on the NetKey Index identifying the NetKey on the NetKey List and on the AppKey Index identifying the AppKey on the AppKey List.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| NetKeyIndexAndAppKeyIndex | 3 | Index of the NetKey and index of the AppKey |

*Table 4.77: Config AppKey Status message parameters*

The Status field shall identify the Status Code for the last operation on the AppKey List. The allowed values for Status codes and their meanings are documented in Section 4.3.6. The Status Code shall be Success if the received request was redundant (add of an identical existing key, update of an identical updated key, or delete of a non-existent key), with no further action taken.

The NetKeyIndexAndAppKeyIndex field contains two indexes that shall identify the global NetKey Index of the NetKey and the global AppKey Index of the AppKey. These two indexes shall be encoded as defined in Section 4.3.1.1 using NetKey Index as first key index and AppKey Index as second key index.

#### 4.3.2.41 Config AppKey Get

The AppKey Get is an acknowledged message used to report all AppKeys bound to the NetKey.

The response to a Config AppKey Get message is a Config AppKey List message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | Index of the NetKey |

*Table 4.78: Config AppKey Get message parameters*

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.42 Config AppKey List

The Config AppKey List is an unacknowledged message reporting all AppKeys that are bound to the NetKey.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| NetKeyIndex | 2 | NetKey Index of the NetKey that the AppKeys are bound to |
| AppKeyIndexes | variable | A list of AppKey indexes that are bound to the NetKey identified by NetKeyIndex |

*Table 4.79: Config AppKey List message parameters*

The Status field shall identify the Status Code for the last operation on the AppKey of the NetKey. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey to which the AppKeys are bound. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The AppKeyIndexes field shall contain all AppKey indexes that are bound to the NetKey. The AppKey indexes shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.43 Config Node Identity Get

The Config Node Identity Get is an acknowledged message used to get the current Node Identity state for a subnet (see Section 4.2.12).

The response to a Config Node Identity Get message is a Config Node Identity Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | Index of the NetKey |

*Table 4.80: Config Node Identity Get message parameters*

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.44 Config Node Identity Set

The Config Node Identity Set is an acknowledged message used to set the current Node Identity state for a subnet (see Section 4.2.12).

The response to a Config Node Identity Set message is a Config Node Identity Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | Index of the NetKey |
| Identity | 1 | New Node Identity state |

*Table 4.81: Config Node Identity Set message parameters*

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey of the Node Identity state. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The Identity field shall provide the new Node Identity state of the NetKey (see Section 4.2.12).

#### 4.3.2.45 Config Node Identity Status

The Config Node Identity Status is an unacknowledged message used to report the current Node Identity state for a subnet (see Section 4.2.12).

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| NetKeyIndex | 2 | Index of the NetKey |
| Identity | 1 | Node Identity state |

*Table 4.82: Config Node Identity Status message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The NetKeyIndex field is an index that shall identify the global NetKey Index of the NetKey of the Node Identity state. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The Identity field shall provide the current Node Identity state for a subnet (see Section 4.2.12).

#### 4.3.2.46 Config Model App Bind

The Config Model App Bind is an acknowledged message used to bind an AppKey to a model.

The response to a Config Model App Bind message is a Config Model App Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| AppKeyIndex | 2 | Index of the AppKey |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.83: Config Model App Bind message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The AppKeyIndex field is an index that shall identify the global AppKey Index of the AppKey. The AppKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The ModelIdentifier field is either a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.47 Config Model App Unbind

The Config Model App Unbind is an acknowledged message used to remove the binding between an AppKey and a model.

The response to a Config Model App Unbind message is a Config Model App Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| AppKeyIndex | 2 | Index of the AppKey |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.84: Config Model App Unbind message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The AppKeyIndex field is an index that shall identify the global AppKey Index of the AppKey. The AppKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The ModelIdentifier field is a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.48 Config Model App Status

The Config Model App Status is an unacknowledged message used to report a status for the requesting message, based on the element address, the AppKeyIndex identifying the AppKey on the AppKey List, and the ModelIdentifier.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| ElementAddress | 2 | Address of the element |
| AppKeyIndex | 2 | Index of the AppKey |
| ModelIdentifier | 2 or 4 | SIG Model ID or Vendor Model ID |

*Table 4.85: Config Model App Status message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6. The Status Code shall be Success if the received request was redundant (bind request of existing binding, or unbind of a non-existing binding), with no further action taken.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The AppKeyIndex field is an index that shall identify the global AppKey Index of the AppKey. The AppKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The ModelIdentifier field is a SIG Model ID or a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.49 Config SIG Model App Get

The Config SIG Model App Get is an acknowledged message used to request report of all AppKeys bound to the SIG Model.

The response to a Config SIG Model App Get message is a Config SIG Model App List message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 | SIG Model ID |

*Table 4.86: Config SIG Model App Get message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a SIG Model ID that shall identify the model within the element.

#### 4.3.2.50 Config SIG Model App List

The Config SIG Model App List is an unacknowledged message used to report all AppKeys bound to the SIG Model.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 2 | SIG Model ID |
| AppKeyIndexes | Variable | All AppKey indexes bound to the Model |

*Table 4.87: Config SIG Model App List message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a SIG Model ID that shall identify the SIG model within the element.

The AppKeyIndexes field shall contain all AppKey indexes that are bound to an instance of a model. The AppKey indexes shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.51 Config Vendor Model App Get

The Config Vendor Model App Get is an acknowledged message used to request report of all AppKeys bound to the model. This message is only for Vendor Models.

The response to a Config Vendor Model App Get message is a Config Vendor Model App List message.

| Field | Size (octets) | Notes |
|---|---|---|
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 4 | Vendor Model ID |

*Table 4.88: Config Vendor Model App Get message parameters*

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a Vendor Model ID that shall identify the model within the element.

#### 4.3.2.52 Config Vendor Model App List

The Config Vendor Model App List is an unacknowledged message used to report indexes of all AppKeys bound to the model. This message is only for Vendor Models.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| ElementAddress | 2 | Address of the element |
| ModelIdentifier | 4 | Vendor Model ID |
| AppKeyIndexes | variable | Indexes of all AppKeys bound to the model |

*Table 4.89: Config Vendor Model App List message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The ElementAddress field is the unicast address of the element, all other address types are Prohibited.

The ModelIdentifier field is a Vendor Model ID that shall identify the model within the element.

The AppKeyIndexes field shall contain indexes of all AppKeys that are bound to an instance of a model. The AppKey indexes shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.53 Config Node Reset

The Config Node Reset is an acknowledged message used to reset a node (other than a Provisioner) and remove it from the network.

The response to a Config Node Reset message is a Config Node Reset Status message.

There are no Parameters for this message.

#### 4.3.2.54 Config Node Reset Status

The Config Node Reset Status is an unacknowledged message used to acknowledge that an element has received a Config Node Reset message.

There are no Parameters for this message.

#### 4.3.2.55 Config Friend Get

The Config Friend Get is an acknowledged message used to get the current Friend state of a node (see Section 4.2.13).

The response to a Config Friend Get message is a Config Friend Status message.

There are no Parameters for this message.

#### 4.3.2.56 Config Friend Set

The Config Friend Set is an acknowledged message used to set the Friend state of a node (see Section 4.2.13).

The response to a Config Friend Set message is a Config Friend Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Friend | 1 | New Friend state |

*Table 4.90: Config Friend Set message parameters*

The Friend field shall provide the new Friend state of the node (see Section 4.2.13).

#### 4.3.2.57 Config Friend Status

The Config Friend Status is an unacknowledged message used to report the current Friend state of a node (see Section 4.2.13).

| Field | Size (octets) | Notes |
|---|---|---|
| Friend | 1 | Friend state |

*Table 4.91: Config Friend Status message parameters*

The Friend field shall provide the current Friend state of the node (see Section 4.2.13).

#### 4.3.2.58 Config Key Refresh Phase Get

The Config Key Refresh Phase Get is an acknowledged message used to get the current Key Refresh Phase state of the identified network key.

The response to a Config Key Refresh Phase Get message is a Config Key Refresh Phase Status message.

| Parameter | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | NetKey Index |

*Table 4.92: Config Key Refresh Phase Get message parameters*

The NetKeyIndex field shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.59 Config Key Refresh Phase Set

The Config Key Refresh Phase Set is an acknowledged message used to set the Key Refresh Phase state of the identified network key (see Section 4.2.14).

The response to a Config Key Refresh Phase Set message is a Config Key Refresh Phase Status message.

| Parameter | Size (octets) | Notes |
|---|---|---|
| NetKeyIndex | 2 | NetKey Index |
| Transition | 1 | New Key Refresh Phase Transition |

*Table 4.93: Config Key Refresh Phase Set message parameters*

The NetKeyIndex field shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The Transition field shall identify the Key Refresh Phase Transitions (see Section 4.2.14, Table 4.18) allowed for each given starting state. All other transition values are Prohibited.

#### 4.3.2.60 Config Key Refresh Phase Status

The Config Key Refresh Phase Status is an unacknowledged message used to report the current Key Refresh Phase state of the identified network key (see Section 4.2.14).

| Parameters | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| NetKeyIndex | 2 | NetKey Index |
| Phase | 1 | Key Refresh Phase State |

*Table 4.94: Config Key Refresh Phase Status message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6. The Status Code shall be Success if the received request was redundant (the requested phase transition has already occurred), with no further action taken.

The NetKeyIndex field shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

The Phase field shall identify the Key Refresh Phase state for the node, as defined in Key Refresh Phase (see Section 4.2.14).

#### 4.3.2.61 Config Heartbeat Publication Get

The Config Heartbeat Publication Get is an acknowledged message used to get the current Heartbeat Publication state of an element (see Section 4.2.17).

The response to a Config Heartbeat Publication Get message is a Config Heartbeat Publication Status message.

The message has no parameters.

#### 4.3.2.62 Config Heartbeat Publication Set

The Config Heartbeat Publication Set is an acknowledged message used to set the current Heartbeat Publication state of an element (see Section 4.2.17).

The response to a Config Heartbeat Publication Set message is a Config Heartbeat Publication Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Destination | 2 | Destination address for Heartbeat messages |
| CountLog | 1 | Number of Heartbeat messages to be sent |

| Parameters | Size (octets) | Notes |
|---|---|---|
| PeriodLog | 1 | Period for sending Heartbeat messages |
| TTL | 1 | TTL to be used when sending Heartbeat messages |
| Features | 2 | Bit field indicating features that trigger Heartbeat messages when changed |
| NetKeyIndex | 2 | NetKey Index |

*Table 4.95: Config Heartbeat Publication Set message parameters*

The Destination field shall identify the Heartbeat Publication Destination state (see Section 4.2.17.1).

The CountLog field shall identify the Heartbeat Publication Count Log state (see Section 4.2.17.2).

The PeriodLog field shall identify the Heartbeat Publication Period Log state (see Section 4.2.17.3).

The TTL field shall identify the Heartbeat Publication TTL state (see Section 4.2.17.4).

The Features field shall identify the Heartbeat Publication Features state (see Section 4.2.17.5).

The NetKeyIndex field shall identify the global NetKey Index of the NetKey. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.63 Config Heartbeat Publication Status

The Config Heartbeat Publication Status is an unacknowledged message used to report the Heartbeat Publication state of a node (see Section 4.2.17).

| Parameters | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| Destination | 2 | Destination address for Heartbeat messages |
| CountLog | 1 | Number of Heartbeat messages remaining to be sent |
| PeriodLog | 1 | Period for sending Heartbeat messages |
| TTL | 1 | TTL to be used when sending Heartbeat messages |
| Features | 2 | Bit field indicating features that trigger Heartbeat messages when changed |
| NetKeyIndex | 2 | NetKey Index |

*Table 4.96: Config Heartbeat Publication Status message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The Destination field shall identify the Heartbeat Publication Destination state (see Section 4.2.17.1).

The CountLog field shall identify the Heartbeat Publication Count Log state (see Section 4.2.17.2).

The PeriodLog field shall identify the Heartbeat Publication Period Log state (see Section 4.2.17.3).

The TTL field shall identify the Heartbeat Publication TTL state (see Section 4.2.17.4).

The Features field shall identify the Heartbeat Publication Features state (see Section 4.2.17.5).

The NetKeyIndex field shall identify the global NetKey Index of the NetKey used to publish heartbeats. The NetKeyIndex field shall be encoded as defined in Section 4.3.1.1.

#### 4.3.2.64 Config Heartbeat Subscription Get

The Config Heartbeat Subscription Get is an acknowledged message used to get the current Heartbeat Subscription state of an element (see Section 4.2.18).

The response to a Config Heartbeat Subscription Get message is a Config Heartbeat Subscription Status message.

The message has no parameters.

#### 4.3.2.65 Config Heartbeat Subscription Set

The Config Heartbeat Subscription Set is an acknowledged message used to set the current Heartbeat Subscription state of an element (see Section 4.2.18).

The response to a Config Heartbeat Subscription Set message is a Config Heartbeat Subscription Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Source | 2 | Source address for Heartbeat messages |
| Destination | 2 | Destination address for Heartbeat messages |
| PeriodLog | 1 | Period for receiving Heartbeat messages |

*Table 4.97: Config Heartbeat Subscription Set message parameters*

The Source field shall identify the Heartbeat Subscription Source state (see Section 4.2.18.1).

The Destination field shall identify the Heartbeat Subscription Destination state (see Section 4.2.18.2)

The PeriodLog field shall identify the Heartbeat Subscription Period state (see Section 4.2.18.4).

#### 4.3.2.66 Config Heartbeat Subscription Status

The Config Heartbeat Subscription Status is an unacknowledged message used to report the Heartbeat Subscription state of a node (see Section 4.2.18).

| Parameters | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| Source | 2 | Source address for Heartbeat messages |
| Destination | 2 | Destination address for Heartbeat messages |
| PeriodLog | 1 | Remaining Period for processing Heartbeat messages |
| CountLog | 1 | Number of Heartbeat messages received |
| MinHops | 1 | Minimum hops when receiving Heartbeat messages |
| MaxHops | 1 | Maximum hops when receiving Heartbeat messages |

*Table 4.98: Config Heartbeat Subscription Status message parameters*

The Status field shall identify the Status Code for the requesting message. The allowed values for Status codes and their meanings are documented in Section 4.3.6.

The Source field shall identify the Heartbeat Subscription Source state (see Section 4.2.18.1).

The Destination field shall identify the Heartbeat Subscription Destination state (see Section 4.2.18.2).

The PeriodLog field shall identify the Heartbeat Subscription Period Log state (see Section 4.2.18.4).

The CountLog field shall identify the Heartbeat Subscription Count Log state (see Section 4.2.18.3).

The MinHops field shall identify the Heartbeat Subscription Min Hops state (see Section 4.2.18.5).

The MaxHops field shall identify the Heartbeat Subscription Max Hops state (see Section 4.2.18.6).

#### 4.3.2.67 Config Low Power Node PollTimeout Get

The Config Low Power Node PollTimeout Get is an acknowledged message used to get the current value of PollTimeout timer of the Low Power node within a Friend node (see Section 3.6.6.1). The message is sent to a Friend node that has claimed to be handling messages by sending ACKs On Behalf Of (OBO) the indicated Low Power node. This message should only be sent to a node that has the Friend feature supported and enabled.

The response to a Config Low Power Node PollTimeout Get message is a Config Low Power Node PollTimeout Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| LPNAddress | 2 | The unicast address of the Low Power node |

*Table 4.99: Config Low Power Node PollTimeout Get message parameters*

The LPNAddress field shall contain the primary unicast address of the Low Power node within a Friend node.

#### 4.3.2.68 Config Low Power Node PollTimeout Status

The Config Low Power Node PollTimeout Status is an unacknowledged message used to report the current value of the PollTimeout timer of the Low Power node within a Friend node.

| Field | Size (octets) | Notes |
|---|---|---|
| LPNAddress | 2 | The unicast address of the Low Power node |
| PollTimeout | 3 | The current value of the PollTimeout timer of the Low Power node |

*Table 4.100: Config Low Power Node PollTimeout Status message parameters*

The LPNAddress field shall contain the primary unicast address of the Low Power node.

The PollTimeout field shall contain the current value of the PollTimeout timer of the Low Power node within a Friend node, or 0x000000 if the node is not a Friend node for the Low Power node identified by LPNAddress.

#### 4.3.2.69 Config Network Transmit Get

The Config Network Transmit Get is an acknowledged message used to get the current Network Transmit state of a node (see Section 4.2.19).

The response to a Config Network Transmit Get message is a Config Network Transmit Status message.

There are no Parameters for this message.

#### 4.3.2.70 Config Network Transmit Set

The Config Network Transmit Set is an acknowledged message used to set the Network Transmit state of a node (see Section 4.2.19).

The response to a Config Network Transmit Set message is a Config Network Transmit Status message.

| Field | Size (bits) | Notes |
|---|---|---|
| NetworkTransmitCount | 3 | Number of transmissions for each Network PDU originating from the node |
| NetworkTransmitIntervalSteps | 5 | Number of 10-millisecond steps between transmissions |

*Table 4.101: Config Network Transmit Set message parameters*

The NetworkTransmitCount field shall contain a new value for the Network Transmit Count state of a node (see Section 4.2.19.1).

The NetworkTransmitIntervalSteps field shall contain a new value for the Network Transmit Interval Steps state of a node (see Section 4.2.19.2).

#### 4.3.2.71 Config Network Transmit Status

The Config Network Transmit Status is an unacknowledged message used to report the current Network Transmit state of a node (see Section 4.2.19).

| Field | Size (bits) | Notes |
|---|---|---|
| NetworkTransmitCount | 3 | Number of transmissions for each Network PDU originating from the node |
| NetworkTransmitIntervalSteps | 5 | Number of 10-millisecond steps between transmissions |

*Table 4.102: Config Network Transmit Set message parameters*

The NetworkTransmitCount field shall contain a new value for the Network Transmit Count state of a node (see Section 4.2.19.1).

The NetworkTransmitIntervalSteps field shall contain a new value for the Network Transmit Interval Steps state of a node (see Section 4.2.19.2).

### 4.3.3 Health messages

Health messages are used to monitor states that determine the physical condition of a node.

Health messages shall be encrypted and authenticated using an AppKey.

#### 4.3.3.1 Health Current Status

The Health Current Status is an unacknowledged message used to report the Current Health state of an element (see Section 4.2.15.1). The message may contain several Fault fields, depending on the number of concurrently present fault conditions. If no Fault fields are present, it means no fault condition exists on an element.

The message uses a single-octet opcode to maximize the size of the FaultArray.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a most recently performed test |
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |
| FaultArray | N | The FaultArray field contains a sequence of 1-octet fault values |

*Table 4.103: Health Current Status message parameters*

The Test ID field identifies a most recently performed test by the element.

The Company ID field is a Bluetooth assigned Company Identifier [6]. It shall be used to resolve vendor specific fault codes as specified in Table 4.21.

The FaultArray field contains a sequence of fault values, as specified in Table 4.21.

#### 4.3.3.2 Health Fault Get

The Health Fault Get is an acknowledged message used to get the current Registered Fault state identified by Company ID of an element (see Section 4.2.15.2).

The response to a Health Fault Get message is a Health Fault Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |

*Table 4.104: Health Fault Get message parameters*

The Company ID field is a Bluetooth assigned Company identifier [6]. It shall be used to resolve specific fault codes as specified in Table 4.21.

#### 4.3.3.3 Health Fault Clear Unacknowledged

The Health Fault Clear Unacknowledged is an unacknowledged message used to clear the current Registered Fault state identified by Company ID of an element (see Section 4.2.15.2).

| Parameters | Size (octets) | Notes |
|---|---|---|
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |

*Table 4.105: Health Fault Clear Unacknowledged message parameters*

The Company ID field is a Bluetooth assigned Company identifier [6]. It shall be used to resolve specific fault codes as specified in Table 4.21.

#### 4.3.3.4 Health Fault Clear

The Health Fault Clear is an acknowledged message used to clear the current Registered Fault state identified by Company ID of an element (see Section 4.2.15.2).

The response to a Health Fault Clear message is a Health Fault Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |

*Table 4.106: Health Fault Clear message parameters*

The Company ID field is a Bluetooth assigned Company identifier [6]. It shall be used to resolve specific fault codes as specified in Table 4.21

#### 4.3.3.5 Health Fault Test

The Health Fault Test is an acknowledged message used to invoke a self-test procedure of an element. The procedure is implementation specific and may result in changing the Health Fault state of an element (see Section 4.2.15).

The response to a Health Fault Test message is a Health Fault Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a specific test to be performed |
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |

*Table 4.107: Health Fault Test message parameters*

The Test ID field identifies a test the element should perform.

The Company ID field is a Bluetooth assigned Company Identifier [6]. It shall be used to resolve vendor specific fault codes as specified in Table 4.21.

#### 4.3.3.6 Health Fault Test Unacknowledged

The Health Fault Test Unacknowledged is an unacknowledged message used to invoke a self-test procedure of an element. The procedure is implementation specific and may result in changing the Health Fault state of an element (see Section 4.2.15).

| Parameters | Size (octets) | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a specific test to be performed |
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |

*Table 4.108: Health Fault Test Unacknowledged message parameters*

The Test ID field identifies a test the element should perform.

The Company ID field is a Bluetooth assigned Company Identifier [6]. It shall be used to resolve vendor specific fault codes as specified in Table 4.21.

#### 4.3.3.7 Health Fault Status

The Health Fault Status is an unacknowledged message used to report the current Registered Fault state of an element (see Section 4.2.15.2). The message may contain several Fault fields, depending on the number of concurrently present fault conditions. If no Fault fields are present, it means no registered fault condition exists on an element.

The message uses a single-octet opcode to maximize the size of the FaultArray.

| Parameters | Size (octets) | Notes |
|---|---|---|
| Test ID | 1 | Identifier of a most recently performed test |
| Company ID | 2 | 16-bit Bluetooth assigned Company Identifier |
| FaultArray | N | The FaultArray field contains a sequence of 1-octet fault values |

*Table 4.109: Health Fault Status message parameters*

The Test ID field identifies a most recently performed test by the element.

The Company ID field is a Bluetooth assigned Company Identifier (https://www.bluetooth.com/specifications/assigned-numbers/company-Identifiers). It shall be used to resolve vendor specific fault codes as specified in Table 4.22.

The FaultArray field contains a sequence of fault values, as specified in Table 4.22.

#### 4.3.3.8 Health Period Get

The Health Period Get is an acknowledged message used to get the current Health Period state of an element (see Section 4.2.16).

The response to a Health Period Get message is a Health Period Status message.

There are no parameters for this message.

#### 4.3.3.9 Health Period Set Unacknowledged

The Health Period Set Unacknowledged is an unacknowledged message used to set the current Health Period state of an element (see Section 4.2.16).

| Parameters | Size (octets) | Notes |
|---|---|---|
| FastPeriodDivisor | 1 | Divider for the Publish Period. Modified Publish Period is used for sending Current Health Status messages when there are active faults to communicate |

*Table 4.110: Health Period Set Unacknowledged message parameters*

The FastPeriodDivisor field shall identify the Health Fast Period Divisor state for the element (see Section 4.2.16).

#### 4.3.3.10 Health Period Set

The Health Period Set is an acknowledged message used to set the current Health Period state of an element (see Section 4.2.16).

The response to a Health Period Set message is a Health Period Status message.

| Parameters | Size (octets) | Notes |
|---|---|---|
| FastPeriodDivisor | 1 | Divider for the Publish Period. Modified Publish Period is used for sending Current Health Status messages when there are active faults to communicate |

*Table 4.111: Health Period Set message parameters*

The FastPeriodDivisor field shall identify the Health Fast Period Divisor state for the element (see Section 4.2.16).

#### 4.3.3.11 Health Period Status

The Health Period Status is an unacknowledged message used to report the Health Period state of an element (see Section 4.2.16).

| Parameters | Size (octets) | Notes |
|---|---|---|
| FastPeriodDivisor | 1 | Divider for the Publish Period. Modified Publish Period is used for sending Current Health Status messages when there are active faults to communicate |

*Table 4.112: Health Period Status message parameters*

The FastPeriodDivisor field shall identify the Health Fast Period Divisor state for the element (see Section 4.2.16).

#### 4.3.3.12 Health Attention Get

The Health Attention Get is an acknowledged message used to get the current Attention Timer state of an element (see Section 4.2.9).

The response to a Health Attention Get message is an Attention Status message.

There are no Parameters for this message.

#### 4.3.3.13 Health Attention Set

The Health Attention Set is an acknowledged message used to set the Attention Timer state of an element (see Section 4.2.9).

The response to a Health Attention Set message is a Health Attention Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Attention | 1 | Value of the Attention Timer state |

*Table 4.113: Health Attention Set message parameters*

The Attention field shall identify the new Attention Timer state of an element (see Section 4.2.9).

#### 4.3.3.14 Health Attention Set Unacknowledged

The Health Attention Set Unacknowledged is an unacknowledged message used to set the Attention Timer state of an element (see Section 4.2.9).

| Field | Size (octets) | Notes |
|---|---|---|
| Attention | 1 | Value of the Attention Timer state |

*Table 4.114: Health Attention Set Unacknowledged message parameters*

The Attention field shall identify the new Attention Timer state of an element (see Section 4.2.9).

#### 4.3.3.15 Health Attention Status

The Health Attention Status is an unacknowledged message used to report the current Attention Timer state of an element (see Section 4.2.9).

| Field | Size (octets) | Notes |
|---|---|---|
| Attention | 1 | Value of the Attention Timer state |

*Table 4.115: Attention Status message parameters*

The Attention field shall identify the current Attention Timer state of the node (see Section 4.2.9).

### 4.3.4 Directed Forwarding messages

Directed Forwarding messages are used to control and monitor states that determine the behavior of Directed Forwarding nodes on each subnet.

#### 4.3.4.1 DF Directed Forwarding Get

The DF Directed Forwarding Get message is an acknowledged message used to get the current Directed Forwarding state of a node in a subnet (see Section 4.2.22).

The response to a DF Directed Forwarding Get message is a DF Directed Forwarding Status message.

There are no parameters for the DF Directed Forwarding Get message.

#### 4.3.4.2 DF Directed Forwarding Set

The DF Directed Forwarding Set message is an acknowledged message used to set the Directed Forwarding state of a node in a subnet (see Section 4.2.22).

The response to a DF Directed Forwarding Set message is a DF Directed Forwarding Status message.

**錯誤! 找不到參照來源。** defines the structure of the DF Directed Forwarding Set message.

| Field | Size (octets) | Notes |
|---|---|---|
| Directed Forwarding | 1 | New Directed Forwarding state in a subnet |

*Table 4.116: DF Directed Forwarding Set message parameters*

The Directed Forwarding field shall identify the new Directed Forwarding state for the node in the subnet, as defined in Section 4.2.22.

### 4.3.4.3 DF Directed Forwarding Status

DF Directed Forwarding Status message is an unacknowledged message used to report the current Directed Forwarding state of a node in a subnet (see Section 4.2.22).

錯誤! 找不到參照來源。 defines the structure of the DF Directed Forwarding Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Directed Forwarding | 1 | Current Directed Forwarding state in a subnet |

*Table 4.117: DF Directed Forwarding Status message parameters*

The Directed Forwarding field shall identify the current Directed Forwarding state for the node in a subnet, as defined in Section 4.2.22.

### 4.3.4.4 DF Path Metric Get

The DF Path Metric Get message is an acknowledged message used to get the current Path Metric state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Path Metric Get message is a DF Path Metric Status message.

There are no parameters for the DF Path Metric Get message.

### 4.3.4.5 DF Path Metric Set

The DF Path Metric Set message is an acknowledged message used to set the Path Metric state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Path Metric Set message is a DF Path Metric Status message.

錯誤! 找不到參照來源。 defines the structure of the DF Path Metric Set message.

| Field | Size (octets) | Notes |
|---|---|---|
| Path Metric Type | 1 | New Path Metric Type in a subnet |
| Path Lifetime | 2 | New Path Lifetime in a subnet |

*Table 4.118: DF Path Metric Set message parameters*

The Path Metric Type field shall identify the new Path Metric Type value of the Path Metric State for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

The Path Lifetime field shall identify the new Path Lifetime value of the Path Metric state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

#### 4.3.4.6 DF Path Metric Status

The DF Path Metric Status message is an unacknowledged message used to report the current Path Metric state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

錯誤! 找不到參照來源。 defines the structure of the DF Path Metric Status message

| Field | Size (octets) | Notes |
|---|---|---|
| Path Metric Type | 1 | Current Path Metric Type in a subnet |
| Path Lifetime | 2 | Current Path Lifetime in a subnet |

*Table 4.119: DF Path Metric Status message parameters*

The Path Metric Type field shall identify the current Path Metric Type value of the Path Metric state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。..

The Path Lifetime field shall identifythe current Path Lifetime value of the Path Metric state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

#### 4.3.4.7 DF Path Request Transmit Get

The DF Path Request Transmit Get message is an acknowledged message used to get the current Path Request Transmit state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Path Request Transmit Get message is a DF Path Request Transmit Status message.

There are no parameters for the DF Path Request Transmit Get message.

#### 4.3.4.8 DF Path Request Transmit Set

The DF Path Request Transmit Set message is an acknowledged message used to set the Path Request Transmit state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Path Request Transmit Set message is a DF Path Request Transmit Status message.

Table 4.120 defines the structure of the DF Path Request Transmit Set message.

| Field | Size (octets) | Notes |
|---|---|---|
| Concurrent Initialization | 1 | New Concurrent Initialization state in a subnet |
| Path Request Interval | 1 | New Path Request Interval state in a subnet |

*Table 4.120: DF Path Request Transmit Set message parameters*

The Concurrent Initialization field shall identify the new Concurrent Initialization state of the Path Request Transmit composite state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

The Path Request Interval field shall identify the new Path Request Interval state of the Path Request Transmit composite state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

### 4.3.4.9 DF Path Request Transmit Status

The DF Path Request Transmit Status message is an unacknowledged message used to report the current Path Request Transmit state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

Table 4.121 defines the structure of the DF Path Request Transmit Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Concurrent Initialization | 1 | Current Concurrent Initialization state in a subnet |
| Path Request Interval | 1 | Current Path Request Interval state in a subnet |

*Table 4.121: DF Path Request Transmit Status message parameters*

The Concurrent Initialization field shall identify the current Concurrent Initialization state of the Path Request Transmit composite state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

The Path Request Interval field shall identify the current Path Request Interval state of the Path Request Transmit composite state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

### 4.3.4.10 DF Neighbor Information Get

The DF Neighbor Information Get message is an acknowledged message used to get the current Neighbor Information state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Neighbor Information Get message is a DF Neighbor Information Status message.

There are no parameters for the DF Neighbor Information Get message.

#### 4.3.4.11 DF Neighbor Information Set

The DF Neighbor Information Set message is an acknowledged message used to set the Neighbor Information state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Neighbor Information Set message is a DF Neighbor Information Status message.

錯誤! 找不到參照來源。 defines the structure of the DF Neighbor Information Set message.

| Field | Size (octets) | Notes |
|---|---|---|
| Neighborhood | 1 | New Neighborhood state in a subnet |
| Neighbor Information Options | 1 | New Neighbor Information Options state in a subnet |

*Table 4.122: DF Neighbor Information Set message parameters*

The Neighborhood field shall identify the new Neighborhood state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

The Neighbor Information Options field shall identify the new Neighbor Information Options state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

#### 4.3.4.12 DF Neighbor Information Status

The DF Neighbor Information Status message is an unacknowledged message used to report the current Neighbor Information state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

錯誤! 找不到參照來源。 defines the structure of the DF Neighbor Information Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Neighborhood | 1 | Current Neighborhood state in a subnet |
| Neighbor Information Options | 1 | Current Neighbor Information Options state in a subnet |

*Table 4.123: DF Neighbor Information Status message parameters*

The Neighborhood field shall identify the current Neighborhood state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

The Neighbor Information Options field shall identify the current Neighbor Information Options state for the node in a subnet, as defined in Section 錯誤! 找不到參照來源。.

### 4.3.4.13 DF Forwarding Table Add

The DF Forwarding Table Add message is an acknowledged message used to add or update a fixed path entry to the Forwarding Table state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Forwarding Table Add message is a DF Forwarding Table Status message.

錯誤! 找不到參照來源。 defines the structure of the DF Forwarding Table Add message.

| Field | Size (octets) | Notes |
|---|---|---|
| PO | 2 | Primary unicast address of the Path Originator |
| OAR | 1 | Secondary element address range assigned to the Path Originator |
| PD | 2 | Primary unicast address of the Path Destination |
| DAR | 1 | Secondary element address range assigned to the Path Destination |
| BTO | 1 | Index of the bearer to be used for forwarding messages directed to the Path Originator |
| BTD | 1 | Index of the bearer to be used for forwarding messages directed to the Path Destination |
| Subscription List | Variable (2 * N) | List of group addresses and/or virtual addresses (N in total) to which the Path Destination is subscribed |

*Table 4.124: DF Forwarding Table Add message parameters*

If the Forwarding Table state of the node in a subnet does not contain a path entry with PO and PD matching the corresponding fields in the message, a new path entry shall be added for the node with the Fixed field set to 1 and containing the field values defined in 錯誤! 找不到參照來源。. If the Forwarding Table state of the node in a subnet contains a path entry with PO and PD matching the corresponding fields in the message, the field values in the path entry shall be set to the values defined in 錯誤! 找不到參照來源。.

In the PO field and the PD field, group addresses, virtual addresses, and the unassigned address are prohibited values.

In the OAR field and the DAR field, 0xFF is a prohibited value.

Values not defined in Section 3.3 and the unassigned bearer index are prohibited for the BTO field and the BTD field.In the Subscription List field, unicast addresses, fixed group addresses, and the unassigned address are prohibited address values.

#### 4.3.4.14 DF Forwarding Table Delete

The DF Forwarding Table Delete message is an acknowledged message used to delete an entry from the Forwarding Table state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

The response to a DF Forwarding Table Delete message is a DF Forwarding Table Status message.

錯誤! 找不到參照來源。 defines the structure of the DF Forwarding Table Delete message.

| Field | Size (octets) | Notes |
|---|---|---|
| PO | 2 | Primary unicast address of the Path Originator |
| DPO | 2 | Primary unicast address of the Delegate Path Originator or the unassigned address |
| PD | 2 | Primary unicast address of the Path Destination |
| DPD | 2 | Primary unicast address of the Delegate Path Destination or the unassigned address |

*Table 4.125: DF Forwarding Table Delete message parameters*

The fields reported in 錯誤! 找不到參照來源。 contain the values of the fields with the same names of the entry to be removed from the Forwarding Table state of a subnet.

In the PO field and the PD field, group addresses, virtual addresses, and the unassigned address are prohibited values.

In the DPO field and the DPD field, group addresses and virtual addresses are prohibited values.

#### 4.3.4.15 DF Forwarding Table Get

The DF Forwarding Table Get message is an acknowledged message used to read a list of entries in the Forwarding Table state of a node in a subnet (see Section 錯誤! 找不到參照來源。). The Forwarding Table state is read as an indexed array of path entries, each referenced by an index from 0 to the size of the array (that is, number of path entries) minus 1.

The response to a DF Forwarding Table Get message is a DF Forwarding Table List message.

錯誤! 找不到參照來源。 defines the structure of the DF Forwarding Table Get message.

| Field | Size (octets) | Notes |
|---|---|---|
| FirstIndex | 2 | Index of the first requested entry in the DF Forwarding Table List |
| IndexRange | 2 | Range of indexes of the requested entries in the DF Forwarding Table List |

*Table 4.126: DF Forwarding Table Get message parameters*

The FirstIndex field shall identify the first path entry of the Forwarding Table state that is being read.

The IndexRange field shall be the number of path entries in the Forwarding Table state that is to be read minus 1. The last entry of the Forwarding Table state that is to be read is identified by the index given by the sum of values of FirstIndex and IndexRange.

#### 4.3.4.16 DF Forwarding Table List

The DF Forwarding Table List message is an unacknowledged message used to report a list of entries in the Forwarding Table state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

Table 4.127 defines the structure of the DF Forwarding Table List message.

| Field | Size (octets) | Notes |
|---|---|---|
| FirstIndex | 2 | Index of the first path entry in PathEntryList |
| PathEntryList | variable | List of reported path entries |

*Table 4.127: DF Forwarding Table List message format*

The FirstIndex field shall be set to the index of the first path entry in the PathEntryList field and shall indicate the position of the entry in the Forwarding Table state of a subnet. The indexes of the path entries in the PathEntryList field (defined in Table 4.128) are consecutive; therefore, the index of the last path entry in PathEntryList is equal to the sum of the FirstIndex value and the number of path entries in PathEntryList minus 1.

The PathEntryList field shall contain a list of path entries consecutively extracted from the Forwarding Table state of a subnet, starting from the position indicated by the FirstIndex field value. 錯誤! 找不到參照來源。 defines the format of each path entry included in the message's PathEntryList.

| Field | Size (octets) | Notes |
|---|---|---|
| OptionalFieldMap | 2 | Bitmap of the optional fields of the path entry:<br>Bit 0: DPD field present (1) or not (0). Set to 1 if a Delegate Path Destination is present for this path.<br>Bit 1: PO field present (1) or not (0). Set to 1 if a Path Originator is present for this path.<br>Bit 2: DPO field present (1) or not (0). Set to 1 if a Delegate Path Originator is present for this path.<br>Bit 3: DAR field present (1) or not (0). Set to 1 if the DAR field is set to a value greater than 0x00.<br>Bit 4: OAR field present (1) or not (0). Set to 1 if the OAR field is set to a value greater than 0x00 and lower than 0xFF.<br>Bit 5: NTD field present (1) or not (0). Set to 1 if a Next Toward Destination address exists for this path. |

| Field | Size (octets) | Notes |
|---|---|---|
| | | Bit 6: NTO field present (1) or not (0). Set to 1 if a Next Toward Originator address exists for this path.<br>Bit 7: DHC field present (1) or not (0). Set to 1 if the DHC field is set to a value greater than 0.<br>Bit 8: BTD field present (1) or not (0). Set to 1 if a BTD field exists for this path.<br>Bit 9: BTO field present (1) or not (0). Set to 1 if a BTO field exists for this path.Bit 10: Both the FN field and Path Remaining Time field present (1) or not (0). Set to 1 if this is a non-fixed path.<br>Bit 11: Both the Subscription List Size field and Subscription List field present (1) or not (0). Set to 1 if the Subscription List Size field is set to a value greater than 0.<br>Bits 12–15: Reserved for Future Use |
| Flags | 1 | Bitmap of the flags of the path entry:<br>Bit 0: Least significant bit of the Fixed field<br>Bit 1: Least significant bit of the BackwardPathValidated field<br>Bit 2: Least significant bit of the ProactivePathUpdated field<br>Bits 3–15: Reserved for Future Use |
| PD | 2 | Primary unicast address of the Path Destination |
| DPD (optional) | 2 | Primary unicast address of the Delegate Path Destination |
| PO (optional) | 2 | Primary unicast address of the Path Originator |
| DPO (optional) | 2 | Primary unicast address of the Delegate Path Originator |
| DAR (optional) | 1 | Secondary element address range assigned to the Path Destination |
| OAR (optional) | 1 | Secondary element address range assigned to the Path Originator |
| NTD (optional) | 2 | Primary unicast address of the next-hop node toward the Path Destination |
| NTO (optional) | 2 | Primary unicast address of the next-hop node toward the Path Originator |
| DHC (optional) | 1 | Hop Count from either the Path Destination or the Delegate Path Destination |
| FN (optional) | 1 | Forwarding Number of the path entry |
| Path Remaining Time (optional) | 2 | Time remaining before the path entry expires |

| Field | Size (octets) | Notes |
|---|---|---|
| BTD (optional) | 1 | The index of the bearer to be used for forwarding messages directed to the Path Destination |
| BTO (optional) | 1 | The index of the bearer to be used for forwarding messages directed to the Path Originator |
| Subscription List Size (optional) | 1 | Number of addresses reported in the Subscription List field |
| Subscription List (optional) | Variable<br>(2 * Subscription List Size) | List of group addresses and virtual addresses that the Path Destination is subscribed to |

*Table 4.128: DF Forwarding Table List message format*

The OptionalFieldMap field shall list optional fields of the path entry that are reported in the message. The choice of which optional fields to include in the message is left to the implementation.

The Flags field shall include the values of the Boolean fields of the path entry in a bitmap.

All other fields in 錯誤! 找不到參照來源。 except for the Path Remaining Time field and the Subscription List Size field shall contain the values of the fields with the same names in the path entry.

The Path Remaining Time field shall contain a value representing the difference between the value of the Path Expiration Time field in the path entry and the current time. The format of the Path Remaining Time field shall match the format of the Path Lifetime state, as defined in 錯誤! 找不到參照來源。錯誤! 找不到參照來源。.

The Subscription List Size field shall contain an integer representing the number of addresses reported in the Subscription List field.

In the PO, DPO PD, DPD, NTO, and NTD fields, the  group addresses, virtual addresses, and the unassigned address are prohibited values.

Zero and 0xFF are Prohibited for the values in the DAR field and in the OAR field.

Zero is Prohibited for the values in the DHC field and in the Subscription List Size field.

The unassigned address, unicast address, and fixed group address are Prohibited for the address values of the Subscription List field.

The DF Forwarding Table List shall contain complete path entries in the index range specified by a corresponding DF Forwarding Table Get message (see Section 錯誤! 找不到參照來源。). The number of reported path entries may be less than the size of the index range.

### 4.3.4.17 DF Forwarding Table Status

The DF Forwarding Table Status message is an unacknowledged message used to report the status of the last operation performed on the Forwarding Table state of a node in a subnet (see Section 錯誤! 找不到參照來源。).

錯誤! 找不到參照來源。 defines the structure eof the DF Forwarding Table Status message.

| Field | Size (octets) | Notes |
|---|---|---|
| Status | 1 | Status Code for the requesting message |
| PO | 2 | Primary unicast address of the Path Originator |
| OAR | 1 | Secondary element address range assigned to the Path Originator |
| DPO | 2 | Primary unicast address of the Delegate Path Originator or the unassigned address |
| PD | 2 | Primary unicast address of the Path Destination |
| DAR | 1 | Secondary element address range assigned to the Path Destination |
| DPD | 2 | Primary unicast address of the Delegate Path Destination or the unassigned address |

*Table 4.129: DF Forwarding Table Status message parameters*

The fields in 錯誤! 找不到參照來源。 shall contain the values used in the last operation to add, update, or delete an entry in the Forwarding Table state of a subnet.

The Status field shall identify the Status Code for the last operation on the Forwarding Table state of a subnet. The allowed values for Status codes and their meanings are defined in Section 4.3.6. The Status Code shall be Success if the received request was redundant (for example, a request to add a path entry that is identical to an existing path entry or to delete a path entry that does not exist), with no further action taken.

In the PO field and the PD field, group addresses, virtual addresses, and the unassigned address are prohibited values.

In the DPO field and the DPD field, group addresses, virtual addresses, and the unassigned address are prohibited values.

## 4.3.5 Messages summary

### 4.3.5.1 Alphabetical summary of opcodes

| Message Name | Opcode |
|---|---|
| Config AppKey Add | 0x00 |

| Message Name | Opcode |
|---|---|
| Config AppKey Delete | 0x80 0x00 |
| Config AppKey Get | 0x80 0x01 |
| Config AppKey List | 0x80 0x02 |
| Config AppKey Status | 0x80 0x03 |
| Config AppKey Update | 0x01 |
| Config Beacon Get | 0x80 0x09 |
| Config Beacon Set | 0x80 0x0A |
| Config Beacon Status | 0x80 0x0B |
| Config Composition Data Get | 0x80 0x08 |
| Config Composition Data Status | 0x02 |
| Config Config Model Publication Set | 0x03 |
| Config Default TTL Get | 0x80 0x0C |
| Config Default TTL Set | 0x80 0x0D |
| Config Default TTL Status | 0x80 0x0E |
| Config Friend Get | 0x80 0x0F |
| Config Friend Set | 0x80 0x10 |
| Config Friend Status | 0x80 0x11 |
| Config GATT Proxy Get | 0x80 0x12 |
| Config GATT Proxy Set | 0x80 0x13 |
| Config GATT Proxy Status | 0x80 0x14 |
| Config Heartbeat Publication Get | 0x80 0x38 |
| Config Heartbeat Publication Set | 0x80 0x39 |
| Config Heartbeat Publication Status | 0x06 |
| Config Heartbeat Subscription Get | 0x80 0x3A |
| Config Heartbeat Subscription Set | 0x80 0x3B |
| Config Heartbeat Subscription Status | 0x80 0x3C |
| Config Key Refresh Phase Get | 0x80 0x15 |
| Config Key Refresh Phase Set | 0x80 0x16 |
| Config Key Refresh Phase Status | 0x80 0x17 |
| Config Low Power Node PollTimeout Get | 0x80 0x2D |
| Config Low Power Node PollTimeout Status | 0x80 0x2E |
| Config Model App Bind | 0x80 0x3D |

| Message Name | Opcode |
|---|---|
| Config Model App Status | 0x80 0x3E |
| Config Model App Unbind | 0x80 0x3F |
| Config Model Publication Get | 0x80 0x18 |
| Config Model Publication Status | 0x80 0x19 |
| Config Model Publication Virtual Address Set | 0x80 0x1A |
| Config Model Subscription Add | 0x80 0x1B |
| Config Model Subscription Delete | 0x80 0x1C |
| Config Model Subscription Delete All | 0x80 0x1D |
| Config Model Subscription Overwrite | 0x80 0x1E |
| Config Model Subscription Status | 0x80 0x1F |
| Config Model Subscription Virtual Address Add | 0x80 0x20 |
| Config Model Subscription Virtual Address Delete | 0x80 0x21 |
| Config Model Subscription Virtual Address Overwrite | 0x80 0x22 |
| Config NetKey Add | 0x80 0x40 |
| Config NetKey Delete | 0x80 0x41 |
| Config NetKey Get | 0x80 0x42 |
| Config NetKey List | 0x80 0x43 |
| Config NetKey Status | 0x80 0x44 |
| Config NetKey Update | 0x80 0x45 |
| Config Network Transmit Get | 0x80 0x23 |
| Config Network Transmit Set | 0x80 0x24 |
| Config Network Transmit Status | 0x80 0x25 |
| Config Node Identity Get | 0x80 0x46 |
| Config Node Identity Set | 0x80 0x47 |
| Config Node Identity Status | 0x80 0x48 |
| Config Node Reset | 0x80 0x49 |
| Config Node Reset Status | 0x80 0x4A |
| Config Relay Get | 0x80 0x26 |
| Config Relay Set | 0x80 0x27 |
| Config Relay Status | 0x80 0x28 |
| Config SIG Model App Get | 0x80 0x4B |
| Config SIG Model App List | 0x80 0x4C |

| Message Name | Opcode |
|---|---|
| Config SIG Model Subscription Get | 0x80 0x29 |
| Config SIG Model Subscription List | 0x80 0x2A |
| Config Vendor Model App Get | 0x80 0x4D |
| Config Vendor Model App List | 0x80 0x4E |
| Config Vendor Model Subscription Get | 0x80 0x2B |
| Config Vendor Model Subscription List | 0x80 0x2C |
| DF Directed Forwarding Get | TBD |
| DF Directed Forwarding Set | TBD |
| DF Directed Forwarding Status | TBD |
| DF Path Metric Get | TBD |
| DF Path Metric Set | TBD |
| DF Path Metric Status | TBD |
| DF Path Request Transmit Get | TBD |
| DF Path Request Transmit Set | TBD |
| DF Path Request Transmit Status | TBD |
| DF Neighbor Information Get | TBD |
| DF Neighbor Information Set | TBD |
| DF Neighbor Information Status | TBD |
| DF Forwarding Table Add | TBD |
| DF Forwarding Table Delete | TBD |
| DF Forwarding Table Get | TBD |
| DF Forwarding Table List | TBD |
| DF Forwarding Table Status | TBD |
| Health Attention Get | 0x80 0x04 |
| Health Attention Set | 0x80 0x05 |
| Health Attention Set Unacknowledged | 0x80 0x06 |
| Health Attention Status | 0x80 0x07 |
| Health Current Status | 0x04 |
| Health Fault Clear | 0x80 0x2F |
| Health Fault Clear Unacknowledged | 0x80 0x30 |
| Health Fault Get | 0x80 0x31 |
| Health Fault Status | 0x05 |

| Message Name | Opcode |
|---|---|
| Health Fault Test | 0x80 0x32 |
| Health Fault Test Unacknowledged | 0x80 0x33 |
| Health Period Get | 0x80 0x34 |
| Health Period Set | 0x80 0x35 |
| Health Period Set Unacknowledged | 0x80 0x36 |
| Health Period Status | 0x80 0x37 |

*Table 4.130: Alphabetical summary of opcodes*

#### 4.3.5.2 Numerical summary of opcodes

| Message Name | Opcode |
|---|---|
| Config AppKey Add | 0x00 |
| Config AppKey Update | 0x01 |
| Config Composition Data Status | 0x02 |
| Config Config Model Publication Set | 0x03 |
| Health Current Status | 0x04 |
| Health Fault Status | 0x05 |
| Config Heartbeat Publication Status | 0x06 |
| Config AppKey Delete | 0x80 0x00 |
| Config AppKey Get | 0x80 0x01 |
| Config AppKey List | 0x80 0x02 |
| Config AppKey Status | 0x80 0x03 |
| Health Attention Get | 0x80 0x04 |
| Health Attention Set | 0x80 0x05 |
| Health Attention Set Unacknowledged | 0x80 0x06 |
| Health Attention Status | 0x80 0x07 |
| Config Composition Data Get | 0x80 0x08 |
| Config Beacon Get | 0x80 0x09 |
| Config Beacon Set | 0x80 0x0A |
| Config Beacon Status | 0x80 0x0B |
| Config Default TTL Get | 0x80 0x0C |
| Config Default TTL Set | 0x80 0x0D |
| Config Default TTL Status | 0x80 0x0E |

| Message Name | Opcode |
|---|---|
| Config Friend Get | 0x80 0x0F |
| Config Friend Set | 0x80 0x10 |
| Config Friend Status | 0x80 0x11 |
| Config GATT Proxy Get | 0x80 0x12 |
| Config GATT Proxy Set | 0x80 0x13 |
| Config GATT Proxy Status | 0x80 0x14 |
| Config Key Refresh Phase Get | 0x80 0x15 |
| Config Key Refresh Phase Set | 0x80 0x16 |
| Config Key Refresh Phase Status | 0x80 0x17 |
| Config Model Publication Get | 0x80 0x18 |
| Config Model Publication Status | 0x80 0x19 |
| Config Model Publication Virtual Address Set | 0x80 0x1A |
| Config Model Subscription Add | 0x80 0x1B |
| Config Model Subscription Delete | 0x80 0x1C |
| Config Model Subscription Delete All | 0x80 0x1D |
| Config Model Subscription Overwrite | 0x80 0x1E |
| Config Model Subscription Status | 0x80 0x1F |
| Config Model Subscription Virtual Address Add | 0x80 0x20 |
| Config Model Subscription Virtual Address Delete | 0x80 0x21 |
| Config Model Subscription Virtual Address Overwrite | 0x80 0x22 |
| Config Network Transmit Get | 0x80 0x23 |
| Config Network Transmit Set | 0x80 0x24 |
| Config Network Transmit Status | 0x80 0x25 |
| Config Relay Get | 0x80 0x26 |
| Config Relay Set | 0x80 0x27 |
| Config Relay Status | 0x80 0x28 |
| Config SIG Model Subscription Get | 0x80 0x29 |
| Config SIG Model Subscription List | 0x80 0x2A |
| Config Vendor Model Subscription Get | 0x80 0x2B |
| Config Vendor Model Subscription List | 0x80 0x2C |
| Config Low Power Node PollTimeout Get | 0x80 0x2D |
| Config Low Power Node PollTimeout Status | 0x80 0x2E |

| Message Name | Opcode |
|---|---|
| Health Fault Clear | 0x80 0x2F |
| Health Fault Clear Unacknowledged | 0x80 0x30 |
| Health Fault Get | 0x80 0x31 |
| Health Fault Test | 0x80 0x32 |
| Health Fault Test Unacknowledged | 0x80 0x33 |
| Health Period Get | 0x80 0x34 |
| Health Period Set | 0x80 0x35 |
| Health Period Set Unacknowledged | 0x80 0x36 |
| Health Period Status | 0x80 0x37 |
| Config Heartbeat Publication Get | 0x80 0x38 |
| Config Heartbeat Publication Set | 0x80 0x39 |
| Config Heartbeat Subscription Get | 0x80 0x3A |
| Config Heartbeat Subscription Set | 0x80 0x3B |
| Config Heartbeat Subscription Status | 0x80 0x3C |
| Config Model App Bind | 0x80 0x3D |
| Config Model App Status | 0x80 0x3E |
| Config Model App Unbind | 0x80 0x3F |
| Config NetKey Add | 0x80 0x40 |
| Config NetKey Delete | 0x80 0x41 |
| Config NetKey Get | 0x80 0x42 |
| Config NetKey List | 0x80 0x43 |
| Config NetKey Status | 0x80 0x44 |
| Config NetKey Update | 0x80 0x45 |
| Config Node Identity Get | 0x80 0x46 |
| Config Node Identity Set | 0x80 0x47 |
| Config Node Identity Status | 0x80 0x48 |
| Config Node Reset | 0x80 0x49 |
| Config Node Reset Status | 0x80 0x4A |
| Config SIG Model App Get | 0x80 0x4B |
| Config SIG Model App List | 0x80 0x4C |
| Config Vendor Model App Get | 0x80 0x4D |
| Config Vendor Model App List | 0x80 0x4E |

| Message Name | Opcode |
|---|---|
| DF Directed Forwarding Get | TBD |
| DF Directed Forwarding Set | TBD |
| DF Directed Forwarding Status | TBD |
| DF Path Metric Get | TBD |
| DF Path Metric Set | TBD |
| DF Path Metric Status | TBD |
| DF Path Request Transmit Get | TBD |
| DF Path Request Transmit Set | TBD |
| DF Path Request Transmit Status | TBD |
| DF Neighbor Information Get | TBD |
| DF Neighbor Information Set | TBD |
| DF Neighbor Information Status | TBD |
| DF Forwarding Table Add | TBD |
| DF Forwarding Table Delete | TBD |
| DF Forwarding Table Get | TBD |
| DF Forwarding Table List | TBD |
| DF Forwarding Table Status | TBD |

*Table 4.131: Numerical summary of opcodes*

### 4.3.6 Summary of status codes

Table 4.132 defines status codes for messages that contain a Status parameter. Status messages are sent only in response to properly formatted messages (see Section 3.7.4.4).

| Status Code | Status Code Name |
|---|---|
| 0x00 | Success |
| 0x01 | Invalid Address |
| 0x02 | Invalid Model |
| 0x03 | Invalid AppKey Index |
| 0x04 | Invalid NetKey Index |
| 0x05 | Insufficient Resources |
| 0x06 | Key Index Already Stored |
| 0x07 | Invalid Publish Parameters |
| 0x08 | Not a Subscribe Model |

| Status Code | Status Code Name |
|---|---|
| 0x09 | Storage Failure |
| 0x0A | Feature Not Supported |
| 0x0B | Cannot Update |
| 0x0C | Cannot Remove |
| 0x0D | Cannot Bind |
| 0x0E | Temporarily Unable to Change State |
| 0x0F | Cannot Set |
| 0x10 | Unspecified Error |
| 0x11 | Invalid Binding |
| 0x12 | Invalid Path Entry |
| 0x13 | Node Not Qualified |
| 0x14-0xFF | RFU |

*Table 4.132: Summary of status codes*

## 4.4 Model definitions

This section defines models used throughout this specification.

Model definitions that are not required as part of this specification are defined in the Mesh Model specification [11] and follow the same format and architecture as mesh model definitions.

### 4.4.1 Configuration Server model

#### 4.4.1.1 Description

The Configuration Server is a root model (i.e., it does not extend any other models).

This model is used to represent a mesh network configuration of a device.

The model shall be supported by a primary element and shall not be supported by any secondary elements. The application-layer security on the Configuration Server model shall use the device key established during provisioning.

The model defines the state instances as shown in Table 4.133 below:

| Configuration Server States | | Bound States | | |
|---|---|---|---|---|
| State | Instance | Model | State | Instance |
| Secure Network Beacon | Primary | - | - | - |
| Composition Data | Primary | - | - | - |

| Configuration Server States | | Bound States | | |
|---|---|---|---|---|
| **State** | **Instance** | **Model** | **State** | **Instance** |
| Default TTL | Primary | - | - | - |
| GATT Proxy | Primary | Configuration Server | Node Identity | Primary |
| Friend | Primary | - | - | - |
| Relay | Primary | - | - | - |
| Model Publication | Primary | - | - | - |
| Subscription List | Primary | - | - | - |
| NetKey List | Primary | - | - | - |
| AppKey List | Primary | - | - | - |
| Model to AppKey List | Primary | - | - | - |
| Node Identity | Primary | - | - | - |
| Key Refresh Phase | Primary | - | - | - |
| Heartbeat Publish | Primary | - | - | - |
| Heartbeat Subscription | Primary | - | - | - |
| Network Transmit | Primary | - | - | - |
| Relay Retransmit | Primary | - | - | - |

*Table 4.133: Configuration Server states and bindings*

The structure of elements, states, and messages covered by this model is defined below in Table 4.134:

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x0000 | Secure Network Beacon (see Section 4.2.10) | Config Beacon Get | M | |
| | | | Config Beacon Set | M | |
| | | | Config Beacon Status | | M |
| | | Composition Data (see Section 4.2.1) | Config Composition Data Get | M | |
| | | | Config Composition Data Status | | M |
| | | Default TTL (see Section 4.2.7) | Config Default TTL Get | M | |
| | | | Config Default TTL Set | M | |
| | | | Config Default TTL Status | | M |
| | | GATT Proxy (see Section 4.2.11) | Config GATT Proxy Get | M | |
| | | | Config GATT Proxy Set | M | |
| | | | Config GATT Proxy Status | | M |

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | Friend (see Section 4.2.13) | Config Friend Get | M | |
| | | | Config Friend Set | M | |
| | | | Config Friend Status | | M |
| | | Relay (see Section 4.2.8) and Relay Retransmit (see Section 4.2.20) | Config Relay Get | M | |
| | | | Config Relay Set | M | |
| | | | Config Relay Status | | M |
| | | Model Publication (see Section 4.2.2) | Config Model Publication Get | M | |
| | | | Config Model Publication Set | M | |
| | | | Config Model Publication Virtual Address Set | M | |
| | | | Config Model Publication Status | | M |
| | | Subscription List (see Section 4.2.3) | Config Model Subscription Add | M | |
| | | | Config Model Subscription Virtual Address Add | M | |
| | | | Config Model Subscription Delete | M | |
| | | | Config Model Subscription Virtual Address Delete | M | |
| | | | Config Model Subscription Virtual Address Overwrite | M | |
| | | | Config Model Subscription Overwrite | M | |
| | | | Config Model Subscription Delete All | M | |
| | | | Config Model Subscription Status | | M |
| | | | Config SIG Model Subscription Get | M | |
| | | | Config SIG Model Subscription List | | M |
| | | | Config Vendor Model Subscription Get | M | |
| | | | Config Vendor Model Subscription List | | M |
| | | NetKey List (see Section 4.2.4) | Config NetKey Add | M | |
| | | | Config NetKey Update | M | |
| | | | Config NetKey Delete | M | |

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | | Config NetKey Status | | M |
| | | | Config NetKey Get | M | |
| | | | Config NetKey List | | M |
| | | AppKey List (see Section 4.2.5) | Config AppKey Add | M | |
| | | | Config AppKey Update | M | |
| | | | Config AppKey Delete | M | |
| | | | Config AppKey Status | | M |
| | | | Config AppKey Get | M | |
| | | | Config AppKey List | | M |
| | | Model to AppKey List (see Section 4.2.6) | Config Model App Bind | M | |
| | | | Config Model App Unbind | M | |
| | | | Config Model App Status | | M |
| | | | Config SIG Model App Get | M | |
| | | | Config SIG Model App List | | M |
| | | | Config Vendor Model App Get | M | |
| | | | Config Vendor Model App List | | M |
| | | Node Identity (see Section 4.2.12) | Config Node Identity Get | M | |
| | | | Config Node Identity Set | M | |
| | | | Config Node Identity Status | | M |
| | | N/A | Config Node Reset | M | |
| | | | Config Node Reset Status | | M |
| | | Key Refresh Phase (see Section 4.2.14) | Config Key Refresh Phase Get | M | |
| | | | Config Key Refresh Phase Set | M | |
| | | | Config Key Refresh Phase Status | | M |
| | | Heartbeat Publication (see Section 4.2.17) | Config Heartbeat Publication Get | M | |
| | | | Config Heartbeat Publication Set | M | |
| | | | Config Heartbeat Publication Status | | M |
| | | Heartbeat Subscription (see Section 4.2.18) | Config Heartbeat Subscription Get | M | |
| | | | Config Heartbeat Subscription Set | M | |
| | | | Config Heartbeat Subscription Status | | M |

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | Network Transmit (see Section 4.2.19) | Config Network Transmit Get | M | |
| | | | Config Network Transmit Set | M | |
| | | | Config Network Transmit Status | | M |

*Table 4.134: Configuration Server elements, states, and messages*

### 4.4.1.2 **Behavior**

This section describes behaviors for states and messages for this server model.

Configuration Server messages shall be secured using a DevKey.

#### *4.4.1.2.1 Secure Network Beacon state*

When an element receives a Config Beacon Get message, it shall respond with a Config Beacon Status message with the Beacon field set to the current Secure Network Beacon state.

When an element receives a Config Beacon Set message, it shall set the Secure Network Beacon state to the value of the Beacon field of the message and respond with a Config Beacon Status message with the Beacon field set to the current Secure Network Beacon state.

#### *4.4.1.2.2 Composition Data state*

When an element receives a Config Composition Data Get message with the Page field of the message containing a value of a Composition Data Page that the node contains, it shall respond with a Config Composition Data Status message with the Page field set to the page number of the Composition Data and the Data field set to the value of the Composition Data Page.

When an element receives a Config Composition Data Get message with the Page field of the message containing a reserved page number or a page number the node does not support, it shall respond with a Config Composition Data Status message with the Page field set to the largest page number of the Composition Data that the node supports and the Data field set to the value of the Composition Data Page for that page number.

Note: It is possible to read all supported Composition Data Pages by reading 0xFF first, and then reading one less than the returned page number until the page number is 0x00.

Note: In this specification, as only page 0 is defined, reading page 0xFF will always return page 0x00, but will in the future return larger page numbers first, before allowing the client to read page 0x00 later.

#### *4.4.1.2.3 Default TTL state*

When an element receives a Config Default TTL Get message, it shall respond with a Config Default TTL Status message with the TTL field set to the current Default TTL state.

When an element receives a Config Default TTL Set message, it shall set the Default TTL state to the value of the TTL field of the message and respond with a Config Default TTL Status message with the TTL field set to the current Default TTL state.

##### 4.4.1.2.4 GATT Proxy state

When an element receives a Config GATT Proxy Get message, it shall respond with a Config GATT Proxy Status message with the GATTProxy field set to the current GATT Proxy state.

When an element receives a Config GATT Proxy Set message and the node supports the Mesh GATT Proxy Service, it shall set the GATT Proxy state to the value of the GATTProxy field of the message and respond with a Config GATT Proxy Status message with the GATTProxy field set to the current GATT Proxy state.

When an element receives a Config GATT Proxy Set message and the node does not support the Mesh GATT Proxy Service, it shall respond with a Config GATT Proxy Status message with the GATTProxy field set to the current GATT Proxy state.

##### 4.4.1.2.5 Friend state

When an element receives a Config Friend Get message, it shall respond with a Config Friend Status message with the Friend field set to the current Friend state.

When an element receives a Config Friend Set message and the node supports the Friend feature, it shall set the Friend state to the value of the Friend field of the message and respond with a Config Friend Status message with the Friend field set to the current Friend state.

When an element receives a Config Friend Set message and the node does not support the Friend feature, it shall respond with a Config Friend Status message with the Friend field set to the current Friend state.

##### 4.4.1.2.6 Relay state

When an element receives a Config Relay Get message, it shall respond with a Config Relay Status message with the Relay field set to the current Relay state, and with the RelayRetransmitCount field set to the current Relay Retransmit Count state, and with the RelayRetransmitIntervalSteps field set to the current Relay Retransmit Interval Steps state.

When an element receives a Config Relay Set message and the node supports the Relay feature, it shall set the Relay state to the value of the Relay field of the message, and set the Relay Retransmit Count state to the value of the RelayRetransmitCount field of the message, and set the Relay Retransmit Interval Steps state to the value of the RelayRetransmitIntervalSteps field of the message and respond with a Config Relay Status message with the Relay field set to the current Relay state, and with the RelayRetransmitCount field set to the current Relay Retransmit Count state, and with the RelayRetransmitIntervalSteps field set to the current Relay Retransmit Interval Steps state.

When an element receives a Config Relay Set message and the node does not support the Relay feature, it shall respond with a Config Relay Status message with the Relay field set to the current Relay state, and setting the RelayRetransmitCount and RelayRetransmitIntervalSteps fields to 0x00.

#### *4.4.1.2.7 Model Publication state*

When an element receives a Config Model Publication Get message that is processed successfully (i.e., it does not result in any error conditions listed in Table 4.135), it shall respond with a Config Model Publication Status message with the current values of the identified Model Publication state, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success. When the PublishAddress is set to the unassigned address, the values of the AppKeyIndex, CredentialFlag, PublishTTL, PublishPeriod, PublishRetransmitCount, and PublishRetransmitIntervalSteps fields shall be set to 0x00.

| **Error Condition** | **Status Code Name (see Table 4.132)** |
|---|---|
| The model defined by ElementAddress and ModelIdentifier does not support the publish mechanism | Invalid Publish Parameters |
| The unicast address provided in ElementAddress is not known to the node | Invalid Address |
| The model identified by SIG Model ID or Vendor Model ID is not found in a given element | Invalid Model |
| The AppKey identified by AppKeyIndex is not known to the node | Invalid AppKey Index |
| The CredentialFlag cannot be set to 1 since the node does not support Low Power feature | Feature Not Supported |

*Table 4.135: Error conditions for Model Publication state*

When an element receives a Config Model Publication Get message, a Config Model Publication Set message, or a Config Model Publication Virtual Address Set message that is not successfully processed (i.e., it results in an error condition listed in Table 4.135), it shall respond with the Config Model Publication Status message, setting its fields to the values of the corresponding fields (i.e. the identically named fields) of the incoming message, setting the Status field to a status code (defined in Table 4.135), and setting all other fields to 0x00.

When an element receives a Config Model Publication Set message or a Config Model Publication Virtual Address Set message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.135), it shall update the identified Model Publication state to the corresponding field values (defined in Table 4.136) and respond with a Config Model Publication Status message with the current values of the identified Model Publication state, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success. When the Publish Address field of a Config Model Publication Set message is set to the unassigned address, the publication of the model shall be disabled and the AppKeyIndex, CredentialFlag, PublishTTL, PublishPeriod, PublishRetransmitCount, and PublishRetransmitIntervalSteps shall be ignored.

| State | Message Field |
|---|---|
| Publication Publish AppKey Index | AppKeyIndex |
| Publish Friendship Credentials Flag | CredentialFlag |
| Publication Publish TTL | PublishTTL |
| Publication Publish Period | PublishPeriod |
| Publish Retransmit Count | PublishRetransmitCount |
| Publish Retransmit Interval Steps | PublishRetransmitIntervalSteps |
| Publication Publish Address | PublishAddress |

*Table 4.136: Model Publication state to message field mappings*

When an element receives a Config Model Publication Set message or a Config Model Publication Virtual Address Set message that is not successfully processed (i.e., it results in an error conditions listed in Table 4.132), it shall respond with a Config Model Publication Status message setting the ElementAddress and ModelIdentifier fields to the corresponding fields of the incoming message, setting the Status field to a status code (defined in Table 4.132), and setting all other fields to 0x00.

#### 4.4.1.2.8 Subscription List state

When an element receives a Config Model Subscription Add message or a Config Model Subscription Virtual Address Add message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137) and requesting to add an Address that is not existing in the identified Subscription List, it shall add value of the Address field to the identified Subscription List and respond with a Config Model Subscription Status message setting the Address, ElementAddress, and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

| Error Condition | Status Code Name (see Table 4.132) |
|---|---|
| The model defined by ElementAddress and ModelIdentifier does not support subscription mechanism | Not a Subscribe Model |
| The device cannot store new address due to insufficient resources on device | Insufficient Resources |
| The unicast address provided in ElementAddress is not known to the node | Invalid Address |
| The model identified by SIG Model ID or Vendor Model ID is not found in a given element | Invalid Model |

*Table 4.137: Error conditions for Subscription List state*

When an Element receives a Config Model Subscription Add message or a Config Model Subscription Virtual Address Add message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137) and requesting to add an Address that is existing in the identified Subscription List, it shall respond with a Config Model Subscription Status message setting the Address, ElementAddress, and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an element receives a Config Model Subscription Add message or a Config Model Subscription Virtual Address Add message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config Model Subscription Status message, setting its fields to the values of the corresponding fields (i.e., the identically named fields) of the incoming message and setting the Status field to a status code (defined in Table 4.137), and setting all other fields to 0.

When an element receives a Config Model Subscription Delete message or a Config Model Subscription Virtual Address Delete message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137) and requesting to delete an Address that is existing in the identified Subscription List, it shall delete the value of the Address field from the identified Subscription List and respond with a Config Model Subscription Status message setting the Address, ElementAddress, and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an Element receives a Config Model Subscription Delete message or a Config Model Subscription Virtual Address Delete message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137) and requesting to delete an Address that is not existing in the identified Subscription List, it shall respond with a Config Model Subscription Status message setting the Address, ElementAddress, and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an element receives a Config Model Subscription Delete message or a Config Model Subscription Virtual Address Delete message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config Model Subscription Status message, setting its fields to the values of the corresponding fields of the incoming message and setting the Status field to a status code (defined in Table 4.137), and setting all other fields to 0.

When an element receives a Config Model Subscription Overwrite message or a Config Model Subscription Virtual Address Overwrite message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137), it shall clear the identified Subscription List, add the value of the Address field to the identified Subscription List, and respond with a Config Model Subscription Status message setting the Address, ElementAddress, and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an element receives a Config Model Subscription Overwrite message or a Config Model Subscription Virtual Address Overwrite message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config Model Subscription Status message, setting its fields to the values of the corresponding fields of the incoming message and setting the Status field to a status code (defined in Table 4.137).

When an element receives a Config Model Subscription Delete All message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.137), it shall clear the identified Subscription List and respond with a Config Model Subscription Status message, setting the

ElementAddress and ModelIdentifier fields as defined by the incoming message, setting the Address field to unassigned address value, and setting the Status field to Success.

When an element receives a Config Model Subscription Delete All message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config Model Subscription Status message, setting its fields to the values of the corresponding fields of the incoming message, setting the Address field to unassigned address value, and setting the Status field to a status code (defined in Table 4.137), and setting all other fields to 0.

When an element receives a Config SIG Model Subscription Get message that is processed successfully (i.e., it does not result in any error conditions listed in Table 4.137), it shall respond with a Config SIG Model Subscription List message with the current values of the identified Subscription List state, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an element receives a Config Vendor Model Subscription Get message that is processed successfully (i.e., it does not result in any error conditions listed in Table 4.137), it shall respond with a Config Vendor Model Subscription List message with the current values of the identified Subscription List state, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message and setting the Status field to Success.

When an element receives a Config SIG Model Subscription Get message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config SIG Model Subscription List message, setting its fields to the values of the corresponding fields of the incoming message, setting the Status field to a status code (defined in Table 4.137), and setting the Addresses field to a zero-length (empty) list.

When an element receives a Config Vendor Model Subscription Get message that is not successfully processed (i.e., it results in an error condition listed in Table 4.137), it shall respond with the Config Vendor Model Subscription List message, setting its fields to the values of the corresponding fields of the incoming message, setting the Status field to a status code (defined in Table 4.137), and setting the Addresses field to a zero-length (empty) list.

##### *4.4.1.2.9 NetKey List state*

When an element receives a Config NetKey Add message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.138), it shall add a new NetKey identified by the NetKeyIndex field to the NetKey List and respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

Note: When an element receives a Config NetKey Add message that identifies a NetKey that has already been added to the NetKey List, it responds with Success, because the result of adding the key again, with the same NetKey value, using the same NetKeyIndex will be the same as the result of adding the key the first time.

| Error Condition | Status Code Name |
| --- | --- |

| | **(see Table 4.132)** |
|---|---|
| The NetKey identified by NetKeyIndex is already stored in the node and the new NetKey value is different | Key Index Already Stored |
| The key identified by NetKeyIndex is not valid for this device for Config NetKey Update message | Invalid NetKey Index |
| The node cannot store the new key due to insufficient resources | Insufficient Resources |
| The requested delete operation cannot be performed due to general constraints | Cannot Remove |
| The requested update operation cannot be performed due to general constraints | Cannot Update |

*Table 4.138: Error conditions for NetKey List state*

When an element receives a Config NetKey Add message that is not successfully processed (i.e., it results in an error condition listed in Table 4.138), it shall respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.138).

When an element receives a Config NetKey Update message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.138), it shall update the value of the NetKey identified by NetKeyIndex field and respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config NetKey Update message that is not successfully processed (i.e., it results in an error condition listed in Table 4.138), it shall respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.138).

When an element receives a Config NetKey Delete message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.138), it shall delete NetKey identified by NetKeyIndex field from the NetKey List, delete all AppKeys bound to the deleted NetKey, and respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success. When an AppKey used in model Publication is deleted as a result of the processing of the Config NetKey Delete message, the publication for the appropriate models shall be disabled. When NetKey used in Heartbeat Publication is deleted as a result of the processing of the Config NetKey Delete message, the Publication for the appropriate NetKey shall be disabled. When a node that supports the Mesh Proxy Service and the NetKey of the subnet utilized in advertising using Node Identity is deleted, the Node Identity state of the subnet of the deleted NetKey shall be set to 0x00. When a node supports the Directed Forwarding feature and the feature is enabled in a subnet corresponding to a NetKey that is deleted, the states associated with the feature and with the subnet of the deleted NetKey shall be set to their default values (see Sections 4.2.22, **錯誤! 找不到參照來源。**, **錯誤! 找不到參照來源。**) and all the entries in the Forwarding Table state (see Section **錯誤! 找不到參照來源。**) and in the Discovery Table (see Section 3.6.8.4.2) shall be removed.

Note: When an element receives a Config NetKey Delete message that identifies a NetKey that is not in the NetKey List, it responds with Success, because the result of deleting the key that does not exist in the NetKey List will be the same as if the key was deleted from the NetKey List.

When an element receives a Config NetKey Delete message that is not successfully processed (i.e., it results in an error condition listed in Table 4.138), it shall respond with a Config NetKey Status message, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.138).

When an element receives a Config NetKey Get message, it shall respond with a Config NetKey List message, setting the NetKeyIndexes field to a list of all indexes of NetKeys known to the node.

A NetKey shall not be deleted from the NetKey List using a message secured with this NetKey.

##### 4.4.1.2.10 AppKey List state

When an element receives a Config AppKey Add message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.139), it shall add a new AppKey identified by AppKeyIndex field to the AppKey List, bind the new AppKey to the NetKey referenced by the NetKeyIndex, and respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to Success.

Note: When an element receives a Config AppKey Add message that identifies an AppKey that has already been added to the AppKey List, it responds with Success, because the result of adding the key again, with the same AppKey value, using the same AppKeyIndex will be the same as the result of adding the key the first time.

| Error Condition | Status Code Name (see Table 4.132) |
|---|---|
| The AppKey identified by AppKeyIndex is already stored in the node and the new AppKey is different | Key Index Already Stored |
| The node cannot store the new key due to insufficient resources | Insufficient Resources |
| The key identified by AppKeyIndex is not valid for this device | Invalid AppKey Index |
| The key identified by NetKeyIndex is not valid for this device | Invalid NetKey Index |
| The requested update operation cannot be performed due to general constraints | Cannot Update |
| The NetKeyIndexAndAppKeyIndex combination is not valid for a Config AppKey Update message | Invalid Binding |

*Table 4.139: Error Conditions for AppKey List state*

When an element receives a Config AppKey Add message that is not successfully processed (i.e., it results in an error condition listed in Table 4.139), it shall respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.139).

When an element receives a Config AppKey Update message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.139), it shall update the value of the AppKey identified by the AppKeyIndex field to the AppKey List and respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config AppKey Update message that is not successfully processed (i.e., it results in an error condition listed in Table 4.139), it shall respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.139).

When an element receives a Config AppKey Delete message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.139), it shall delete the AppKey identified by the AppKeyIndex field from the AppKey List and respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to Success. When an AppKey used in Model Publication is deleted as a result of the processing of the Config AppKey Delete message, the publication for the appropriate models shall be disabled.

Note: When an element receives a Config AppKey Delete message that identifies an AppKey that is not in the AppKey List, it responds with Success, because the result of deleting the key that does not exist in the AppKey List will be the same as if the key was deleted from the AppKey List.

When an element receives a Config AppKey Delete message that is not successfully processed (i.e., it results in an error condition listed in Table 4.139), it shall respond with a Config AppKey Status message, setting the NetKeyIndexAndAppKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.139).

When an element receives a Config AppKey Get message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.139), it shall respond with a Config AppKey List message, setting the AppKeyIndexes field to a list of all indexes of AppKeys bound to the NetKey identified by the NetKeyIndex field, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config AppKey Get message that is not successfully processed (i.e., it results in an error condition listed in Table 4.139), it shall respond with a Config AppKey List message, setting the NetKeyIndex field as defined by the incoming message, setting the Status field to a status code (defined in Table 4.139), and setting the AppKeyIndexes field to a zero-length (empty) list.

#### *4.4.1.2.11 Model to AppKey List state*

When an element receives a Config Model App Bind message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.140), it shall bind the AppKey referenced by the AppKeyIndex to the identified Model and respond with a Config Model App Status message, setting the

ElementAddress, AppKeyIndex, and ModelIdentifier fields as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config Model App Bind message that is not successfully processed (i.e., it results in an error condition listed in Table 4.140), it shall respond with the Config Model App Status message, setting its fields to the values of the corresponding fields (i.e., the identically named fields) of the incoming message, and setting the Status field to a status code (defined in Table 4.140).

| **Error Condition** | **Status Code Name (see Table 4.132)** |
|---|---|
| The model identified by SIG Model ID or Vendor Model ID is not found for a given element | Invalid Model |
| The unicast address provided in ElementAddress is not used by the node | Invalid Address |
| The key identified by AppKeyIndex is not stored in the node | Invalid AppKey Index |
| The node cannot store new binding due to insufficient resources | Insufficient Resources |
| The requested bind operation cannot be performed due to general constraints | Cannot Bind |

*Table 4.140: Error Conditions for Model to AppKey List state*

When an element receives a Config Model App Unbind message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.140), it shall unbind the AppKey referenced by the AppKeyIndex from the identified model and respond with a Config Model App Status message, setting the ElementAddress, AppKeyIndex, and ModelIdentifier fields as defined by the incoming message, and setting the Status field to Success. When the specified AppKeyIndex is used by the identified model as a Publish AppKeyIndex the Model Publication, the publication for the model shall be disabled.

When an element receives a Config Model App Unbind message that is not successfully processed (i.e., it results in an error condition listed in Table 4.140), it shall respond with the Config Model App Status message, setting its fields to the values of the corresponding fields of the incoming message, and setting the Status field to a status code (defined in Table 4.140).

When an element receives a Config SIG Model App Get message that is processed successfully (i.e., it does not result in any error conditions listed in Table 4.140), it shall respond with a Config SIG Model App List message with the current values of the identified Model to AppKey List, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config SIG Model App Get message that is not successfully processed (i.e., it results in an error condition listed in Table 4.140), it shall respond with the Config SIG Model App List message, setting its fields to the values of the ElementAddress and ModelIdentifier of the incoming message, setting the Status field to a status code (defined in Table 4.140), and setting the AppKeyIndexes field to a zero-length (empty) list.

When an element receives a Config Vendor Model App Get message that is processed successfully (i.e., it does not result in any error conditions listed in Table 4.140), it shall respond with a Config Vendor Model App List message with the current values of the identified Model to AppKey List, setting the ElementAddress and ModelIdentifier fields as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config Vendor Model App Get message that is not successfully processed (i.e., it results in an error condition listed in Table 4.140), it shall respond with the Config Vendor Model App List message, setting its fields to the values of the ElementAddress and ModelIdentifier of the incoming message, setting the Status field to a status code (defined in Table 4.140), and setting the AppKeyIndexes field to a zero-length (empty) list.

### *4.4.1.2.12 Node Identity state*

When an element receives a Config Node Identity Get message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.141), it shall respond with a Config Node Identity Status message with the Identity field set to the current Node Identity state identified by the NetKeyIndex field, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

| **Error Condition** | **Status Code Name (see Table 4.132)** |
|---|---|
| The key identified by the NetKeyIndex is not valid for this device | Invalid NetKey Index |
| The node cannot start advertising with Node Identity since the maximum number of parallel advertising is reached | Temporarily Unable to Change State |

*Table 4.141: Error Conditions for Node Identity state*

When an element receives a Config Node Identity Get message that is not successfully processed (i.e., it results in any error conditions listed in Table 4.141), it shall respond with a Config Node Identity Status message, setting the NetKeyIndex field as defined by the incoming message, setting the Identity field to 0x00, and setting the Status field to a status code (defined in Table 4.141).

When an element receives a Config Node Identity Set message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.141), it shall set the Node Identity state identified by the NetKeyIndex field to the value of the Identity field of the message and respond with a Config Node Identity Status message with the Identity field set to the current Node Identity state of the NetKey, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config Node Identity Set message that is not successfully processed (i.e., it results in any error conditions listed in Table 4.141), it shall respond with a Config Node Identity Status message, setting the NetKeyIndex and Identity fields as defined by the incoming message, and setting the Status field to a status code defined in Table 4.141.

#### 4.4.1.2.13 Reset

When an element receives a Config Node Reset message, it shall perform the Node Removal procedure (see Section 3.10.7) and respond with a Config Node Reset Status message.

#### 4.4.1.2.14 Key Refresh Phase state

When an element receives a Config Key Refresh Phase Get message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.142), it shall respond with a Config Key Refresh Phase Status message with the Phase field set to the current Key Refresh Phase state, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

| Error Condition | Status Code Name (see Table 4.132) |
|---|---|
| The key identified by the NetKeyIndex is not valid for this device | Invalid NetKey Index |

*Table 4.142: Error Conditions for Key Refresh Phase state*

When an element receives a Config Key Refresh Phase Get message that is not successfully processed (i.e., it results in any of the error conditions listed in Table 4.142), it shall respond with a Config Key Refresh Phase Status message with the Phase field set to 0x00, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.142).

When an element receives a Config Key Refresh Phase Set message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.142), it shall set the Key Refresh Phase state according to Table 4.18 and respond with a Config Key Refresh Phase Status message with the Phase field set to the current Key Refresh Phase state, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to Success.

When an element receives a Config Key Refresh Phase Set message that is not successfully processed (i.e., it results in any of the error conditions listed in Table 4.142), it shall respond with a Config Key Refresh Phase Status message with the Phase field set to 0x00, setting the NetKeyIndex field as defined by the incoming message, and setting the Status field to a status code (defined in Table 4.142).

#### 4.4.1.2.15 Heartbeat Publication state

When an element receives a Config Heartbeat Publication Get message, it shall respond with a Config Heartbeat Publication Status message with the current values of the identified Heartbeat Publication state, setting the Status field to Success, setting the Destination field to the value of the Heartbeat Publication Destination state, the CountLog field to the representation of the Heartbeat Publication Count state, the PeriodLog field to the value of the Heartbeat Publication Period Log state, the TTL field to the value of the Heartbeat Publication TTL state, the Features field to the value of the Heartbeat Publication Features state, and the NetKey Index to the value of the Heartbeat Publication NetKey Index state. When the Destination is set to the unassigned address, the values of the CountLog, PeriodLog, and TTL fields shall be set to 0x00.

| Error Condition | Status Code Name (see Table 4.132) |
|---|---|
| The key identified by the NetKeyIndex is not valid for this device | Invalid NetKey Index |

*Table 4.143: Error Conditions for Heartbeat Publication state*

When an element receives a Config Heartbeat Publication Set message that is not successfully processed (i.e., it results in any of the error conditions listed in Table 4.143), it shall respond with a Config Heartbeat Publication Status message, setting the Destination, CountLog, PeriodLog, and TTL fields to the values of corresponding fields of the incoming message and setting the Status field to a status code (defined in Table 4.143).

When an element receives a Config Heartbeat Publication Set message that is successfully processed (i.e., it does not result in any error conditions listed in Table 4.143), it shall update the identified Heartbeat Publication state to the corresponding field values (defined in Table 4.144) and the Heartbeat Publication Count state to the corresponding value (defined in the Table 4.145) and respond with a Config Heartbeat Publication Status message, setting the Status field to Success and setting the Destination, CountLog, PeriodLog, TTL, Features, and NetKeyIndex fields with the current values of the corresponding fields of the Heartbeat Publication state.

| State | Message Field |
|---|---|
| Heartbeat Publication Destination | Destination |
| Heartbeat Publication Period Log | PeriodLog |
| Heartbeat Publication Count Log | CountLog |
| Heartbeat Publication TTL | TTL |
| Heartbeat Publication Features | Features |
| Heartbeat Publication NetKey Index | NetKeyIndex |

*Table 4.144: Heartbeat Publication State to Message Field Mappings*

| Heartbeat Publication Count Log | Heartbeat Publication Count |
|---|---|
| 0x00 | 0x0000 |
| 0x01-0x11 | $2^{(\text{Heartbeat Publication Count Log}-1)}$ |
| 0x12–0xFE | Prohibited |
| 0xFF | 0xFFFF |

*Table 4.145: Heartbeat Publication Count Log to Heartbeat Publication Count Mappings*

#### 4.4.1.2.16 Heartbeat Subscription state

When an element receives a Config Heartbeat Subscription Get message, it shall respond with a Config Heartbeat Subscription Status message, setting the Status field to Success, setting the Source, Destination, PeriodLog, CountLog, MinHops and MaxHops fields with the current values of the corresponding fields (defined in Table 4.146) of the identified Heartbeat Subscription state. When the Heartbeat Subscription Source state or the Heartbeat Subscription Destination state is set to the unassigned address, the value of the Source and Destination fields of the Config Heartbeat Subscription Status message shall be set to the unassigned address and the values of the CountLog, PeriodLog, MinHops, and MaxHops fields shall be set to 0x00.

When an element receives a Config Heartbeat Subscription Set message, it shall update the identified Heartbeat Subscription state to the corresponding field values (defined in Table 4.146) and respond with a Config Heartbeat Subscription Status message, setting the Status field to Success and setting the Source, Destination, PeriodLog, CountLog, MinHops, and MaxHops fields with the current values of the corresponding fields of the Heartbeat Subscription state. If the Source or the Destination field is set to the unassigned address, or the PeriodLog field is set to 0x00, then the processing of received Heartbeat messages shall be disabled, the Heartbeat Subscription Source state shall be set to the unassigned address, the Heartbeat Subscription Destination state shall be set to the unassigned address, the Heartbeat Subscription MinHops state shall be unchanged, the Heartbeat Subscription MaxHops state shall be unchanged, and the Heartbeat Subscription Count state shall be unchanged. If the Source field is set to a unicast address, and the Destination field is set to a unicast address or group address, and the PeriodLog field is set to a non-zero value, then the processing of received Heartbeat messages shall be enabled, the Heartbeat Subscription MinHops state shall be set to 0x7F, the Heartbeat Subscription MaxHops state shall be set to 0x00, and the Heartbeat Subscription Count state shall be set to 0x0000.

| State | Message Field |
|---|---|
| Heartbeat Subscription Source | Source |
| Heartbeat Subscription Destination | Destination |
| Heartbeat Subscription Period Log | PeriodLog |
| Heartbeat Subscription Count Log | CountLog |
| Heartbeat Subscription Min Hops | MinHops |
| Heartbeat Subscription Max Hops | MaxHops |

*Table 4.146: Heartbeat Subscription state to message field mappings*

#### *4.4.1.2.17 PollTimeout List state*

When an element receives a Config Low Power Node PollTimeout Get message, it shall respond with a Config Low Power Node PollTimeout Status message with the PollTimeout field set to the current PollTimeout List state element identified by the value of the LPNAddress field.

#### *4.4.1.2.18 Network Transmit state*

When an element receives a Config Network Transmit Get message, it shall respond with a Config Network Transmit Status message that has the NetworkTransmitCount field set to the current Network Transmit Count state and the NetworkTransmitIntervalSteps field set to the current Network Transmit Interval Steps state.

When an element receives a Config Network Transmit Set message, it shall set the Network Transmit Count state to the value of the NetworkTransmitCount field and shall set the Network Transmit Interval Steps state to the value of the NetworkTransmitIntervalSteps fields of the message and shall respond with a Config Network Transmit Status message that has the NetworkTransmitCount field set to the current Network Transmit Count state and the NetworkTransmitIntervalSteps field set to the current Network Transmit Interval Steps state.

#### 4.4.1.3 Error handling behavior

When a node receives a message that requires it to store some information in the node's persistent memory and the storage is not successful, the node shall use the Storage Failure as a value of the Status Code. This might be either a permanent or a temporary situation.

When an error occurs that does not correspond to an error condition defined for a given state, the node shall use the Unspecified Error as a value of the Status Code (see Table 4.132).

### 4.4.2 Configuration Client model

#### 4.4.2.1 Description

The Configuration Client is a root model (i.e., it does not extend any other models).

The model is used to represent an element that can control and monitor the configuration of a node.

The model defines the elements and procedures listed in Table 4.147 below.

| **Element** | **SIG Model ID** | **Procedure** | **Messages** | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x0001 | Secure Network Beacon | Config Beacon Get | | M |
| | | | Config Beacon Set | | M |
| | | | Config Beacon Status | M | |
| | | Composition Data | Config Composition Data Get | | M |
| | | | Config Composition Data Status | M | |
| | | Default TTL | Config Default TTL Get | | M |
| | | | Config Default TTL Set | | M |
| | | | Config Default TTL Status | M | |
| | | GATT Proxy | Config GATT Proxy Get | | M |
| | | | Config GATT Proxy Set | | M |
| | | | Config GATT Proxy Status | M | |
| | | Friend | Config Friend Get | | M |
| | | | Config Friend Set | | M |

| Element | SIG Model ID | Procedure | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | | Config Friend Status | M | |
| | | Relay | Config Relay Get | | M |
| | | | Config Relay Set | | M |
| | | | Config Relay Status | M | |
| | | Model Publication | Config Model Publication Get | | M |
| | | | Config Model Publication Set | | M |
| | | | Config Model Publication Virtual Address Set | | M |
| | | | Config Model Publication Status | M | |
| | | Subscription List | Config Model Subscription Add | | M |
| | | | Config Model Subscription Virtual Address Add | | M |
| | | | Config Model Subscription Delete | | M |
| | | | Config Model Subscription Virtual Address Delete | | M |
| | | | Config Model Subscription Overwrite | | M |
| | | | Config Model Subscription Virtual Address Overwrite | | M |
| | | | Config Model Subscription Delete All | | M |
| | | | Config Model Subscription Status | M | |
| | | | Config SIG Model Subscription Get | | M |
| | | | Config SIG Model Subscription List | M | |
| | | | Config Vendor Model Subscription Get | | M |
| | | | Config Vendor Model Subscription List | M | |
| | | NetKey List | Config NetKey Add | | M |
| | | | Config NetKey Update | | M |
| | | | Config NetKey Delete | | M |
| | | | Config NetKey Status | M | |
| | | | Config NetKey Get | | M |
| | | | Config NetKey List | M | |
| | | AppKey List | Config AppKey Add | | M |
| | | | Config AppKey Update | | M |

| Element | SIG Model ID | Procedure | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | | Config AppKey Delete | | M |
| | | | Config AppKey Status | M | |
| | | | Config AppKey Get | | M |
| | | | Config AppKey List | M | |
| | | Model to AppKey List | Config Model App Bind | | M |
| | | | Config Model App Unbind | | M |
| | | | Config Model App Status | M | |
| | | | Config SIG Model App Get | | M |
| | | | Config SIG Model App List | M | |
| | | | Config Vendor Model App Get | | M |
| | | | Config Vendor Model App List | M | |
| | | Node Identity | Config Node Identity Get | | M |
| | | | Config Node Identity Set | | M |
| | | | Config Node Identity Status | M | |
| | | Reset | Config Node Reset | | M |
| | | | Config Node Reset Status | M | |
| | | Key Refresh Phase | Config Key Refresh Phase Get | | M |
| | | | Config Key Refresh Phase Set | | M |
| | | | Config Key Refresh Phase Status | M | |
| | | Heartbeat Publication | Config Heartbeat Publication Get | | M |
| | | | Config Heartbeat Publication Set | | M |
| | | | Config Heartbeat Publication Status | M | |
| | | Heartbeat Subscription | Config Heartbeat Subscription Get | | M |
| | | | Config Heartbeat Subscription Set | | M |
| | | | Config Heartbeat Subscription Status | M | |
| | | Network Transmit | Config Network Transmit Get | | M |
| | | | Config Network Transmit Set | | M |
| | | | Config Network Transmit Status | M | |

*Table 4.147: Configuration Client elements and procedures*

#### 4.4.2.2 Behavior

This section describes behaviors for procedures and messages for this client model.

An element can send any Configuration Client message at any time to query or change a configuration state of a peer element. Configuration Client messages shall be secured using a DevKey.

##### *4.4.2.2.1 Secure Network Beacon procedure*

To determine the Secure Network Beacon state of a Configuration Server, a Configuration Client shall send a Config Beacon Get message. The response is a Config Beacon Status message that contains the Secure Network Beacon state.

To set the Secure Network Beacon state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Beacon Set message. The response is a Config Beacon Status message that contains the Secure Network Beacon state.

Upon receiving a Config Beacon Status message, a Configuration Client can determine the Secure Network Beacon state of a Configuration Server.

##### *4.4.2.2.2 Composition Data procedure*

The Composition Data state of a server is composed or one or more pages. To determine the Composition Data state of a Configuration Server, a Configuration Client shall send a Config Composition Data Get message with the Page field value set to 0xFF. The response is a Config Composition Data Status message that contains the last page of the Composition Data state. If the Page field of the Config Composition Data Status message contains a non-zero value, then the Configuration Client shall send another Composition Data Get message with the Page field value set to one less than the Page field value of the Config Composition Data Status message.

##### *4.4.2.2.3 Default TTL procedure*

To determine the Default TTL state of a Configuration Server, a Configuration Client shall send a Config Default TTL Get message. The response is a Config Default TTL Status message that contains the Default TTL state.

To set the Default TTL state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Default TTL Set message. The response is a Config Default TTL Status message that contains the Default TTL state.

Upon receiving a Config Default TTL Status message, a Configuration Client can determine the current Default TTL state of a Configuration Server.

##### *4.4.2.2.4 GATT Proxy procedure*

To determine the GATT Proxy state of a Configuration Server, a Configuration Client shall send a Config GATT Proxy Get message. The response is a Config GATT Proxy Status message that contains the GATT Proxy state.

To set the GATT Proxy state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config GATT Proxy Set message. The response is a Config GATT Proxy Status message that contains the GATT Proxy state.

Upon receiving a Config GATT Proxy Status message, a Configuration Client can determine the current GATT Proxy state of a Configuration Server.

##### 4.4.2.2.5 Friend procedure

To determine the Friend state of a Configuration Server, a Configuration Client shall send a Config Friend Get message. The response is a Config Friend Status message that contains the Friend state.

To set the Friend state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Friend Set message. The response is a Config Friend Status message that contains the Friend state.

Upon receiving a Config Friend Status message, a Configuration Client can determine the Friend state of a Configuration Server.

##### 4.4.2.2.6 Relay procedure

To determine the Relay and Relay Retransmit states of a Configuration Server, a Configuration Client shall send a Config Relay Get message. The response is a Config Relay Status message that contains the Relay and Relay Retransmit states.

To set the Relay and Relay Retransmit states of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Relay Set message. The response is a Config Relay Status message that contains the Relay and Relay Retransmit states.

Upon receiving a Config Relay Status message, a Configuration Client can determine the current Relay and Relay Retransmit states of a Configuration Server.

##### 4.4.2.2.7 Model Publication procedure

To determine the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states of a particular model within the element, a Configuration Client shall send a Config Model Publication Get message. The response is a Config Model Publication Status message that contains a status and may contain the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states.

To set the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Publication Set message. The response is a Config Model Publication Status message that contains a status and may contain the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states.

To unset the Publish Address state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Publication Set message with the PublishAddress field set to Unassigned and with the AppKeyIndex, CredentialFlag, PublishTTL, PublishRetransmitCount, PublishRetransmitIntervalSteps, and PublishPeriod fields set to 0.

To set the Label UUID as the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, and Publish TTL states of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Publication Virtual Address Set message. The response is a Config Model Publication Status message that contains a status and may contain the Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states.

Upon receiving a Config Model Publication Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.135). If it's Success, the Configuration Client can also determine the current Publish Address, Publish AppKey Index, CredentialFlag, Publish Period, Publish Retransmit Count, Publish Retransmit Interval Steps, and Publish TTL states of a particular model within the element. If it's an error, the Status field shall contain the error condition; and the values of the Publish Address, Publish AppKey Index, CredentialFlag, PublishPeriod, PublishRetransmitCount, PublishRetransmitIntervalSteps, and PublishTTL fields shall be discarded.

##### 4.4.2.2.8 Subscription List procedure

To add the address to the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Add message. The response is a Config Model Subscription Status message that contains a status and may contain the added address value.

To add the Label UUID to the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Virtual Address Add message. The response is a Config Model Subscription Status message that contains a status and may contain the added address value.

To delete the address from the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Delete message. The response is a Config Model Subscription Status message that contains a status and may contain the deleted address value.

To delete the Label UUID from the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Virtual Address Delete message. The response is a Config Model Subscription Status message that contains a status and may contain the deleted address value.

To clear the Subscription List and add the address to the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Overwrite message. The response is a Config Model Subscription Status message that contains a status and may contain the added address value.

To clear the Subscription List and add the Label UUID to the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Virtual Address Overwrite message. The response is a Config Model Subscription Status message that contains a status and may contain the added address value.

To clear the Subscription List of the Subscription List state of a particular model within the element with acknowledgment, a Configuration Client shall send a Config Model Subscription Delete All message. The response is a Config Model Subscription Status message that contains a status.

Note: After adding a previously not known group address to one of the node's subscription lists, the node is not protected against a replay attack utilizing messages to that new group address. It is therefore strongly recommended that the Configuration Client run, for a brief period of time, a Heartbeat Subscription procedure on the node and a Heartbeat Publication procedure on all nodes that publish to the new group address to initialize the replay protection list of the node with the current value of the sequence numbers for all affected publishers.

Upon receiving a Config Model Subscription Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.137). If it's Success, the Configuration Client may determine the address that was used to change the Subscription List state of a particular model within the element. If it's an error, the Status field will contain the error condition.

To determine the Subscription List state of a particular SIG Model within the element, a Configuration Client shall send a Config SIG Model Subscription Get message. The response is a Config SIG Model Subscription List message that contains a status and may contain the Subscription List state.

To determine the Subscription List state of a particular Vendor Model within the element, a Configuration Client shall send a Config Vendor Model Subscription Get message. The response is a Config Vendor Model Subscription List message that contains a status and may contain the Subscription List state.

Upon receiving a Config SIG Model Subscription List message or a Config Vendor Model Subscription List message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.137). If it's Success, the Configuration Client can also determine the current Subscription List state of a particular model within the element. If it's an error, the Status field will contain the error condition, and the Addresses field will be set to a zero-length (empty) list.

##### *4.4.2.2.9 NetKey List procedure*

To add the NetKey identified by NetKeyIndex to the NetKey List state with acknowledgment, a Configuration Client shall send a Config NetKey Add message. The response is a Config NetKey Status message that contains a status and the NetKeyIndex value.

To update the NetKey identified by NetKeyIndex to the NetKey List state with acknowledgment, a Configuration Client shall send a Config NetKey Update message. The response is a Config NetKey Status message that contains a status and the NetKeyIndex value.

To delete the NetKey identified by NetKeyIndex to the NetKey List state with acknowledgment, a Configuration Client shall send a Config NetKey Delete message. The response is a Config NetKey Status message that contains a status and the NetKeyIndex value.

Upon receiving a Config NetKey Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.138) and the NetKeyIndex value. If it's Success, the Status field will be set to Success. If it's an error, the Status field will contain the error condition.

To determine the NetKey List of the Configuration Server, a Configuration Client shall send a Config NetKey Get message. The response is a Config NetKey List message that contains a NetKey List.

Upon receiving a Config NetKey List message, a Configuration Client can determine the current NetKey List of a Configuration Server.

##### *4.4.2.2.10 AppKey List procedure*

To add the AppKey to the AppKey List and bind it to the NetKey identified by the NetKeyIndex of a Configuration Server with acknowledgment, a Configuration Client shall send a Config AppKey Add message. The response is a Config AppKey Status message that contains a status, the added AppKey index, and the NetKeyIndex value.

To update the value of the AppKey from the AppKey List of the NetKey identified by the NetKeyIndex of a Configuration Server with acknowledgment, a Configuration Client shall send a Config AppKey Update message. The response is a Config AppKey Status message that contains a status, the added AppKey index, and the NetKeyIndex value.

To delete the AppKey from the AppKey List of the NetKey identified by the NetKeyIndex of a Configuration Server with acknowledgment, a Configuration Client shall send a Config AppKey Delete message. The response is a Config AppKey Status message that contains a status, the deleted AppKey index, and the NetKeyIndex value.

Upon receiving a Config AppKey Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.139). If it's Success, the Configuration Client can determine the AppKey index that was used to change the AppKey List of the NetKey identified by the NetKeyIndex of a Configuration Server. If it's an error, the Status field will contain the error condition.

To determine the AppKey List of the NetKey identified by the NetKeyIndex of a Configuration Server, a Configuration Client shall send a Config AppKey Get message. The response is a Config AppKey List message that contains a status and may contain the AppKey List.

Upon receiving a Config AppKey List message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.139). If it's Success, the Configuration Client can also determine the current AppKey List of the NetKey identified by the NetKeyIndex of a Configuration Server. If it's an error, the Status field will contain the error condition, and the AppKeyIndexes field will be set to a zero-length (empty) list.

#### *4.4.2.2.11 Model to AppKey List procedure*

To bind the AppKey to a model of a particular element of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Model App Bind message. The response is a Config Model App Status message that contains a status and other fields set to the values of the corresponding fields (i.e., the identically named fields) of the incoming message.

To unbind the AppKey from a model of a particular element of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Model App Unbind message. The response is a Config Model App Status message that contains a status and other fields set to the values of the corresponding fields of the incoming message.

Upon receiving a Config Model App Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.140). If it's Success, the Configuration Client can also determine the values of the ElementAddress, AppKeyIndex, and ModelIdentifier that were used to change the Model to AppKey List state. If it's an error, the Status field will contain the error condition.

To determine the Model to AppKey List of a particular SIG Model within the element, a Configuration Client shall send a Config SIG Model App Get message. The response is a Config SIG Model App List message that contains a status and may contain the Model to AppKey List.

Upon receiving a Config SIG Model App List message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.140). If it's Success, the Configuration Client can also determine the current Model to AppKey List of a particular SIG Model within the element. If it's an error, the Status field will contain the error condition, and the AppKeyIndexes field will be set to a zero-length (empty) list.

To determine the Model to AppKey List of a particular Vendor Model within the element, a Configuration Client shall send a Config Vendor Model App Get message. The response is a Config Vendor Model App List message that contains a status and may contain the Model to AppKey List.

Upon receiving a Config Vendor Model App List message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.140). If it's Success, the Configuration Client can also determine the current Model to AppKey List of a particular Vendor model within the element. If it's an error, the Status field will contain the error condition, and the AppKeyIndexes field will be set to a zero-length (empty) list.

#### *4.4.2.2.12 Node Identity procedure*

To determine the Node Identity state of the NetKey identified by the NetKeyIndex of a Configuration Server, a Configuration Client shall send a Config Node Identity Get message. The response is a Config Node Identity Status message that contains a status, NetKeyIndex value, and the Node Identity state of the NetKey.

To set the Node Identity state of the NetKey identified by the NetKeyIndex of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Node Identity Set message. The response is a Config Node Identity Status message that contains a status, NetKeyIndex value, and the Node Identity state.

Upon receiving a Config Node Identity Status message, a Configuration Client can determine the status that can be either Success or an error (see Table 4.141). If it's Success, the Configuration Client can also determine the current Node Identity state of the NetKey identified by the NetKeyIndex of a Configuration Server. If it's an error, the Status field will contain the error condition, and the Identity field will be set to zero.

##### 4.4.2.2.13 Reset procedure

To initiate the Node Removal procedure of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Node Reset message. The response is a Config Node Reset Status message.

Upon receiving a Config Node Reset Status message, a Configuration Client can determine that the Node Removal procedure was initiated on a Configuration Server.

##### 4.4.2.2.14 Key Refresh Phase procedure

To determine the Key Refresh Phase state of a Configuration Server, a Configuration Client shall send a Config Key Refresh Phase Get message. The response is a Config Key Refresh Phase Status message that contains a status, the NetKeyIndex value, and the Key Refresh Phase state.

To set the Key Refresh Phase state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Key Refresh Phase Set message. The response is a Config Key Refresh Phase Status message that contains a status, the NetKeyIndex value, and the Key Refresh Phase state.

Upon receiving a Config Key Refresh Phase Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.142). If it's Success, the Configuration Client can also determine the current Key Refresh Phase state of the NetKey identified by the NetKeyIndex of a Configuration Server. If it's an error, the Status field will contain the error condition.

##### 4.4.2.2.15 Heartbeat Publication procedure

A configuration client may use the Heartbeat Publication and Heartbeat Subscription procedures to map a topology of the subnet. Using the Heartbeat Publication procedure sets one node to publish a series of Heartbeat messages (see Section 3.6.5.10), and using the Heartbeat Subscription procedure sets another node to process received Heartbeat messages and report them to the configuration client.

To determine the Heartbeat Publication Destination, Heartbeat Publication Count Log, Heartbeat Publication Period Log, Heartbeat Publication TTL, Heartbeat Publication Features, and Heartbeat NetKey Index states of a node, a Configuration Client shall send a Config Heartbeat Publication Get message. The response is a Config Heartbeat Publication Status message that contains the Heartbeat Publication Destination, Heartbeat Publication Count Log, Heartbeat Publication Period Log, Heartbeat Publication TTL, Heartbeat Publication Features and Heartbeat Publication NetKey Index states.

To set the Heartbeat Publication Destination, Heartbeat Publication Count, Heartbeat Publication Period, Heartbeat Publication TTL, Publication Features, and Publication NetKey Index of a node with acknowledgment, a Configuration Client shall send a Config Heartbeat Publication Set message. The response is a Config Heartbeat Publication Status message that contains a Status of the operation and the Heartbeat Publication Destination, Heartbeat Publication Count Log, Heartbeat Publication Period Log, Heartbeat Publication TTL, Heartbeat Publication Features, and Heartbeat Publication NetKey Index states.

Upon receiving a Config Heartbeat Publication Status message, a Configuration Client can determine the status that can be either a Success or an error (see Table 4.143). If it's Success, the Configuration Client can also determine the current Heartbeat Publication Destination, Heartbeat Publication Count Log, Heartbeat Publication Period Log, Heartbeat Publication TTL, Heartbeat Publication Features, and Heartbeat Publication NetKey Index states of a node. If it's an error, the Status field will contain the error condition.

#### *4.4.2.2.16 Heartbeat Subscription procedure*

To determine the Heartbeat Subscription Source, Heartbeat Subscription Destination, Heartbeat Subscription Count Log, Heartbeat Subscription Period Log, Heartbeat Subscription Min Hops, and Heartbeat Subscription Max Hops states of a node, a Configuration Client shall send a Config Heartbeat Subscription Get message. The response is a Config Heartbeat Subscription Status message that contains the Heartbeat Subscription Source, Heartbeat Subscription Destination, Heartbeat Subscription Count Log, Heartbeat Subscription Period Log, Heartbeat Subscription Min Hops, and Heartbeat Subscription Max Hops states.

To set the Heartbeat Subscription Source, Heartbeat Subscription Destination, Heartbeat Subscription Count, and Heartbeat Subscription Period states of a node with acknowledgment, a Configuration Client shall send a Config Heartbeat Subscription Set message. The response is a Config Heartbeat Subscription Status message that contains a Status of the operation and the Heartbeat Subscription Source, Heartbeat Subscription Destination, Heartbeat Subscription Count Log, Heartbeat Subscription Period Log, Heartbeat Subscription Min Hops, and Heartbeat Subscription Max Hops states.

Upon receiving a Config Heartbeat Subscription Status message, a Configuration Client can determine the current Heartbeat Subscription Source, Heartbeat Subscription Destination, Heartbeat Subscription Count Log, Heartbeat Subscription Period Log, Heartbeat Subscription Min Hops, and Heartbeat Subscription Max Hops states of a node.

#### *4.4.2.2.17 PollTimeout List procedure*

To determine the current PollTimeout List state value of a Configuration Server, a Configuration Client shall send a Config Low Power Node PollTimeout Get message with the LPNAddress field set to the primary unicast address of the Low Power node. The response is a Config Low Power Node PollTimeout Status message that contains the current value of the PollTimeout List state for that LPNAddress.

#### *4.4.2.2.18 Network Transmit procedure*

To determine the Network Transmit state of a Configuration Server, a Configuration Client shall send a Config Network Transmit Get message. The response is a Config Network Transmit Status message that contains the Network Transmit state.

To set the Network Transmit state of a Configuration Server with acknowledgment, a Configuration Client shall send a Config Network Transmit Set message. The response is a Config Network Transmit Status message that contains the Network Transmit state.

Upon receiving a Config Network Transmit Status message, a Configuration Client can determine the Network Transmit state of a Configuration Server.

### 4.4.3 Health Server model

#### 4.4.3.1 Description

The Health Server is a root model (i.e., it does not extend any other models).

This model supports both publishing and subscribing.

This model is used to represent a mesh network diagnostics of a device.

The model shall be supported by a primary element and may be supported by any secondary elements. The application-layer security on the model is using application keys.

The model defines the following state instances as shown in Table 4.148 below:

| **Health Server States** | | **Bound States** | | |
|---|---|---|---|---|
| **State** | **Instance** | **Model** | **State** | **Instance** |
| Current Fault | Primary | - | - | - |
| Registered Fault | Primary | - | - | - |
| Health Period | Primary | - | - | - |
| Attention Timer | Primary | - | - | - |

*Table 4.148: Health Server states and bindings*

The structure of elements, states, and messages covered by this model is defined below in Table 4.149:

| **Element** | **SIG Model ID** | **States** | **Messages** | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x0002 | Current Fault (see Section 4.2.15.1) | Health Current Status | | M |
| | | Registered Fault (see Section 4.2.15.2) | Health Fault Get | M | |
| | | | Health Fault Clear | M | |
| | | | Health Fault Clear Unacknowledged | M | |
| | | | Health Fault Status | | M |

| **Element** | **SIG Model ID** | **States** | **Messages** | Rx | Tx |
|---|---|---|---|---|---|
| | | | Health Fault Test | M | |
| | | | Health Fault Test Unacknowledged | M | |
| | | Health Period (see Section 4.2.16) | Health Period Get | M | |
| | | | Health Period Set | M | |
| | | | Health Period Set Unacknowledged | M | |
| | | | Health Period Status | | M |
| | | Attention Timer (see Section 4.2.9) | Health Attention Get | M | |
| | | | Health Attention Set | M | |
| | | | Health Attention Set Unacknowledged | M | |
| | | | Health Attention Status | | M |

*Table 4.149: Health Server elements, states, messages*

### 4.4.3.2 Behavior

This section describes behaviors for states and messages for this server model.

#### *4.4.3.2.1 Current Fault state*

When the value of a Publish Period state is non-zero, and a Current Fault Fault Array (see Section 4.2.15.1) for any Company ID contains no records, an unsolicited Health Current Status message with the Company ID field set to one of the Company IDs supported by the Health Fault state and an empty FaultArray field shall be published every number of seconds as defined by the value of the Publish Period state. It is recommended that in this case the Company ID is set to the value of the CID field of the Composition Data state (see Section 4.2.1).

When the value of a Health Fast Period Divisor state is non-zero, and a Current Fault Fault Array (see Section 4.2.15.1) for at least one Company ID contains records, an unsolicited Health Current Status message set to the value of that Company ID and the FaultArray field containing a sequence of faults representing a sequence of faults in the Current Fault Fault Array (see Section 4.2.15.1) shall be published every number of seconds as defined by the value of the Publish Period divided by the value represented by the Health Fast Period Divisor state.

#### *4.4.3.2.2 Registered Fault state*

When an element receives a Health Fault Get message with the Company ID field that successfully identifies Health Fault state, it shall respond with a Health Fault Status message with the Company ID field set to the value as set in the incoming message, the Test ID field set to the ID of the most recently performed test for identified state, and the FaultArray field containing a sequence of faults representing a sequence of faults in the Registered Fault FaultArray (see Section 4.2.15.2).

When an element receives a Health Fault Test message with the Company ID field that successfully identified Health Fault state, it shall perform a test indicated by the Test ID and Company ID fields and respond with a Health Fault Status message with the Company ID field set to the value as set in the incoming message, the Test ID field set to the ID of the performed test, and a FaultArray field containing a sequence of faults representing a sequence of faults in the Registered Fault FaultArray (see Section 4.2.15.2).

When an element receives a Health Fault Test Unacknowledged message with the Company ID field that successfully identified Health Fault state, it shall perform a test indicated by the Test ID and Company ID fields.

When an element receives a Health Fault Clear message with the Company ID field that successfully identifies Health Fault state, it shall clear the identified Registered Fault state and respond with a Health Fault Status message with the Company ID field set to the values set in the incoming message, and an empty FaultArray field.

When an element receives a Health Fault Clear Unacknowledged message with the Company ID field that successfully identifies Health Fault state, it shall clear the identified Registered Fault state.

When an Element receives a Health Fault Get, or a Health Fault Test, or a Health Fault Test Unacknowledged, or a Health Fault Clear, or a Health Fault Clear Unacknowledged message that is not successfully processed (i.e. the Company ID field that does not identify any Health Fault state present in the node), it shall ignore the message.

##### 4.4.3.2.3 Health Period states

When an element receives a Health Period Get message, it shall respond with a Health Period Status message with the FastPeriodDivisor field set to the current Health Fast Period Divisor state.

When an element receives a Health Period Set message, it shall set the Health Fast Period Divisor state to the value of the FastPeriodDivisor field, and respond with a Health Period Status message with the FastPeriodDivisor field set to the current Health Fast Period Divisor state.

When an element receives a Health Period Set Unacknowledged message, it shall set the Health Fast Period Divisor state to the value of the FastPeriodDivisor field.

##### 4.4.3.2.4 Attention Timer state

When an element receives a Health Attention Get message, it shall respond with a Health Attention Status message with the Attention field set to the current Attention Timer state.

When an element receives a Health Attention Set message, it shall set the Attention Timer state to the value of the Attention field of the message and respond with a Health Attention Status message with the Attention field set to the current Attention Timer state.

When an element receives a Health Attention Set Unacknowledged message, it shall set the Attention Timer state to the value of the Attention field of the message.

An unsolicited Health Attention Status message with the Attention field set to the current Attention Timer state may be sent at any time.

### 4.4.4 Health Client model

#### 4.4.4.1 Description

The Health Client is a root model (i.e., it does not extend any other models).

If supported, the Health Client model shall be supported by a primary element and may be supported by any secondary elements. The application-layer security on the model is using application keys.

The model defines the elements and procedures listed in Table 4.150 below.

| Element | SIG Model ID | Procedure | Messages | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x0003 | Current Fault | Current Health Status | M | |
| | | Registered Fault | Health Fault Get | | M |
| | | | Health Fault Clear | | M |
| | | | Health Fault Clear Unacknowledged | | M |
| | | | Health Fault Status | M | |
| | | | Health Fault Test | | M |
| | | | Health Fault Test Unacknowledged | | M |
| | | Health Period | Health Period Get | | M |
| | | | Health Period Set | | M |
| | | | Health Period Set Unacknowledged | | M |
| | | | Health Period Status | M | |
| | | Attention Timer | Health Attention Get | | M |
| | | | Health Attention Set | | M |
| | | | Health Attention Set Unacknowledged | | M |
| | | | Health Attention Status | M | |

*Table 4.150: Health Client procedures and messages*

#### 4.4.4.2 Behavior

This section describes behaviors for procedures and messages for this client model.

An element can send any Health Client message at any time to query or change a state of a peer element.

##### 4.4.4.2.1 Current Fault procedure

Upon receiving a Health Current Status message, a Health Client can determine the Current Fault state of a Health Server.

##### 4.4.4.2.2 Registered Fault procedure

To determine the Registered Fault state identified by Company ID of a Health Server, a Health Client shall send a Health Fault Get message. The response is a Health Fault Status message that contains the Registered Fault state.

To execute a self-test identified by a Test ID and Company ID for a given element and determine the Registered Fault state identified by Company ID of a Health Server with acknowledgment, a Health Client shall send a Health Fault Test message. The response is a Health Fault Status message that contains the Registered Fault state identified by Company ID.

To execute a self-test identified by a Test ID and Company ID for a given element without acknowledgment, a Health Client shall send a Health Fault Test Unacknowledged message.

To clear the Registered Fault state identified by Company ID of a Health Server without acknowledgment, a Health Client shall send a Health Fault Clear Unacknowledged message.

To clear the Registered Fault state identified by Company ID of a Health Server with acknowledgment, a Health Client shall send a Health Fault Clear message. The response is a Health Fault Status message that contains the identified Registered Fault state.

Upon receiving a Health Fault Status message, a Health Client can determine the Registered Fault state of a Health Server.

##### 4.4.4.2.3 Health Period procedure

To determine the Health Fast Period Divisor state of a Health Server, a Health Client shall send a Health Period Get message. The response is a Health Period Status message that contains the Health Fast Period Divisor state.

To set the Health Fast Period Divisor state of a Health Server without acknowledgment, a Health Client shall send a Health Period Set Unacknowledged message.

To reliably set the Health Fast Period Divisor state of a Health Server, a Health Client shall send a Health Period Set message. The response is a Health Period Status message that contains the Health Fast Period Divisor state.

Upon receiving a Health Period Status message, a Health Client can determine the Health Fast Period Divisor state of a Health Server.

#### *4.4.4.2.4 Attention Timer procedure*

To determine the Attention Timer state of a Health Server, a Health Client shall send a Health Attention Get message. The response is a Health Attention Status message that contains the Attention Timer state.

To set the Attention Timer state of a Health Server with acknowledgment, a Health Client shall send a Health Attention Set message. The response is a Health Attention Status message that contains the Attention Timer state.

To set the Attention Timer state of a Health Server without acknowledgment, a Health Client shall send a Health Attention Set Unacknowledged message.

Upon receiving a Health Attention Status message, a Health Client can determine the current Attention Timer state of a Health Server.

## 4.4.5 Directed Forwarding Server model

### 4.4.5.1 Description

The Directed Forwarding Server is a root model (that is, it does not extend any other models).

This model is used to configure the behavior of a Directed Forwarding node separately in each subnet that the node belongs to.

The model shall be supported by a primary element and shall not be supported by any secondary elements. The application-layer security on the Directed Forwarding Server model shall use the device key established during provisioning.

The model defines the state instances in 錯誤! 找不到參照來源。.

| Directed Forwarding Server States | | Bound States | | |
|---|---|---|---|---|
| **State** | **Instance** | **Model** | **State** | **Instance** |
| Directed Forwarding | Primary | - | - | - |
| Path Metric | Primary | - | - | - |
| Path Request Transmit | Primary | - | - | - |
| Neighbor Information | Primary | - | - | - |
| Forwarding Table | Primary | - | - | - |

*Table 4.151: Directed Forwarding Server states and bindings*

The structure of elements, states, and messages covered by this model are defined in 錯誤! 找不到參照來源。.

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|

| Element | SIG Model ID | States | Messages | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x000x | Directed Forwarding (see Section 4.2.22) | DF Directed Forwarding Get | M | - |
| | | | DF Directed Forwarding Set | M | - |
| | | | DF Directed Forwarding Status | - | M |
| Primary | 0x000x | Path Metric (see Section 錯誤! 找不到參照來源。) | DF Path Metric Get | M | - |
| | | | DF Path Metric Set | M | - |
| | | | DF Path Metric Status | - | M |
| Primary | 0x000x | Path Request Transmit (see Section 錯誤! 找不到參照來源。錯誤! 找不到參照來源。) | DF Path Request Transmit Get | M | - |
| | | | DF Path Request Transmit Set | M | - |
| | | | DF Path Request Transmit Status | - | M |
| Primary | 0x000x | Neighbor Information (see Section 錯誤! 找不到參照來源。) | DF Neighbor Information Get | M | - |
| | | | DF Neighbor Information Set | M | - |
| | | | DF Neighbor Information Status | - | M |
| Primary | 0x000x | Forwarding Table (see Section 錯誤! 找不到參照來源。) | DF Forwarding Table Add | M | - |
| | | | DF Forwarding Table Delete | M | - |
| | | | DF Forwarding Table Status | - | M |
| | | | DF Forwarding Table Get | M | - |
| | | | DF Forwarding Table List | - | M |

M = Mandatory

*Table 4.152: Directed Forwarding Server elements, states, and messages*

### 4.4.5.2 **Behavior**

This section describes behaviors for states and messages for the Directed Forwarding Server model.

Directed Forwarding Server messages shall be secured using a device key (DevKey). When a Directed Forwarding Server message is transmitted in response to a Directed Forwarding Client message, itis transmitted over the same subnet from which the Directed Forwarding Client message was received.

#### *4.4.5.2.1 Directed Forwarding state*

When an element receives a DF Directed Forwarding Get message from a given subnet, it shall respond with a DF Directed Forwarding Status message with the Directed Forwarding field set to the current Directed Forwarding state of the subnet.

When an element receives a DF Directed Forwarding Set message from a given subnet, it shall set the Directed Forwarding state of the subnet to the value of the Directed Forwarding field of the message, and shall respond with a DF Directed Forwarding Status message with the Directed Forwarding field set to the current Directed Forwarding state of the subnet. If the values reported in the DF Directed Forwarding Set message are not valid according to requirements specified in Section 4.2.22, the current value of the Directed Forwarding state of the subnet shall not be changed.

##### 4.4.5.2.2 Path Metric state

When an element receives a DF Path Metric Get message from a given subnet, it shall respond with a DF Path Metric Status message with the Path Metric Type field and the Path Lifetime field set to the current values of the corresponding states of the Path Metric composite state of the subnet.

When an element receives a DF Path Metric Set message from a given subnet, it shall set the Path Metric Type state and the Path Lifetime state of the Path Metric composite state of the subnet to the values of the corresponding fields of the message, and shall respond with a DF Path Metric Status message with the Path Metric Type field and the Path Lifetime field set to the current values of the corresponding states of the Path Metric composite state of the subnet. If the values reported in the DF Path Metric Set message are not valid according to requirements specified in Section **錯誤! 找不到參照來源。**, the current value of the Path Metric state of the subnet shall not be changed.

##### 4.4.5.2.3 Path Request Transmit state

When an element receives a DF Path Request Transmit Get message from a given subnet, it shall respond with a DF Path Request Transmit Status message with the Concurrent Initialization field and the Path Request Interval field set to the current values of the corresponding states in the Path Request Transmit composite state of the subnet.

When an element receives a DF Path Request Transmit Set message from a given subnet, it shall set the Concurrent Initialization state and the Path Request Interval state in the Path Request Transmit composite state of the subnet to the values of the corresponding fields of the message, and shall respond with a DF Path Request Transmit Status message with the Concurrent Initialization field and the Path Request Interval field set to the current values of the corresponding state in the Path Request Transmit composite state of the subnet. If the values reported in the DF Path Request Transmit Set message are not valid according to requirements specified in Section **錯誤! 找不到參照來源。**, the current values of the Path Request Transmit state of the subnet shall not be changed.

##### 4.4.5.2.4 Neighbor Information state

When an element receives a DF Neighbor Information Get message from a given subnet, it shall respond with a DF Neighbor Information Status message with the Neighborhood field and the Neighbor Information Options field set to the current values of the Neighborhood state and of the Neighbor Information Options state of the subnet, respectively.

When an element receives a DF Neighbor Information Set message from a given subnet, it shall set the Neighborhood state of the subnet to the value of the Neighborhood field of the message and shall set the Neighbor Information Options state of the subnet to the value of the Neighbor Information Options field of the message. The element then shall respond with a DF Neighbor Information Status message with the Neighborhood field set to the current value of the Neighborhood state of the subnet and the Neighbor Information Options field set to the current value of the Neighbor Information Options state of the subnet.

If the values reported in the DF Neighbor Information Set message are not valid according to requirements specified in Section 錯誤! 找不到參照來源。, the current values of the Neighborhood state and the Neighbor Information Options state of the subnet shall not be changed.

#### 4.4.5.2.5 Forwarding Table state

When an element receives a DF Forwarding Table Add message from a given subnet ~~that~~, and the message is processed successfully (that is, it does not result in any error conditions listed in 錯誤! 找不到參照來源。), the element shall add a new path entry to the Forwarding Table state of the subnet~~,~~ if the entry does not exist; if the path already exists, the element shall update the existing path entry. In the path entry, the PO, OAR, PD, DAR, and Subscription List fields shall be set to the values in the incoming message and the Fixed field shall be set to 1; all other fields shall be ignored. The element also shall respond with a DF Forwarding Table Status message in which the values of the PO, OAR, PD, and DAR fields are set to the values defined by the incoming DF Forwarding Table Add message, the Status field is set to Success, and the DPO and DPD fields are set to the unassigned address.

| Error Condition | Status Code Name (see Table 4.132) |
|---|---|
| The addresses of the fixed path entry are not valid | Invalid Path Entry |

*Table 4.153: Error Conditions for Forwarding Table state*

When an element receives a DF Forwarding Table Delete message from a given subnet, and the message is processed successfully (that is, it does not result in any error conditions listed in 錯誤! 找不到參照來源。), the element shall delete the corresponding path entry from the Forwarding Table state of the subnet, if the entry exists. The element also shall respond with a DF Forwarding Table Status message in which the values of the PO, DPO, PD, and DDP fields are set to the values defined by the incoming DF Forwarding Table Add message, the Status field is set to Success, and the OAR and DAR fields are set to the values of the corresponding fields in the path entry that is to be deleted, if such an entry exists, or set to 0xFF, if no such entry exists.

When an element receives a DF Forwarding Table Add message from a given subnet, and the message is not successfully processed (that is, it results in an error condition listed in 錯誤! 找不到參照來源。), the element shall respond with a DF Forwarding Table Status message, in which the values of the PO, OAR, PD, and DAR fields as set to the values defined in the incoming DF Forwarding Table Add message, the Status field is set to an error status code (as defined in 錯誤! 找不到參照來源。), and the DPO and DPD fields are set to the unassigned address.

When an element receives a DF Forwarding Table Delete message from a given subnet, and the message is not successfully processed (that is, it results in an error condition listed in 錯誤! 找不到參照來源。), the element shall respond with a DF Forwarding Table Status message, shall set the values of the PO, DPO, PD, and DPD fields as defined by the incoming message, shall set the Status field to an error status code (as defined in 錯誤! 找不到參照來源。), and shall set the OAR and DAR fields to 0xFF.

When an element receives a DF Forwarding Table Get message from a given subnet, it shall respond with a DF Forwarding Table List message that contains complete path entries extracted from the Forwarding Table state of the subnet within the index range specified by the incoming message. If the value of the FirstIndex field of the DF Forwarding Table Get message is greater than the position of the last path entry in the Forwarding Table state of the subnet, the DF Forwarding Table List message shall be returned empty. Otherwise, the DF Forwarding Table List message shall contain all the path entries in the Forwarding Table state of the subnet in the index range from the FirstIndex value to the sum of the values of the FirstIndex and IndexRange fields that can fit the maximum message size.

The fields of each complete path entry in the DF Forwarding Table List message shall be set as defined in Section 錯誤! 找不到參照來源。.

### 4.4.6 Directed Forwarding Client model

#### 4.4.6.1 Description

The Directed Forwarding Client model is a root model (that is, it does not extend any other models).

The model is used to represent an element that can control and monitor the configuration of the states associated with the Directed Forwarding feature of a node separately in each subnet that the node belongs to.

The model defines the elements and procedures listed in 錯誤! 找不到參照來源。.

| Element | SIG Model ID | Procedure | Messages | Rx | Tx |
|---|---|---|---|---|---|
| Primary | 0x000x | Directed Forwarding | DF Directed Forwarding Get | - | M |
| | | | DF Directed Forwarding Set | - | M |
| | | | DF Directed Forwarding Status | M | - |
| Primary | 0x000x | Path Metric | DF Path Metric Get | - | M |
| | | | DF Path Metric Set | - | M |
| | | | DF Path Metric Status | M | - |
| Primary | 0x000x | Path Request Transmit | DF Path Request Transmit Get | - | M |
| | | | DF Path Request Transmit Set | - | M |
| | | | DF Path Request Transmit Status | M | - |
| Primary | 0x000x | Neighbor Info | DF Neighbor Information Get | - | M |
| | | | DF Neighbor Information Set | - | M |
| | | | DF Neighbor Information Status | M | - |
| Primary | 0x000x | Forwarding Table | DF Forwarding Table Add | - | M |
| | | | DF Forwarding Table Delete | - | M |
| | | | DF Forwarding Table Status | M | - |
| | | | DF Forwarding Table Get | - | M |

| Element | SIG Model ID | Procedure | Messages | Rx | Tx |
|---|---|---|---|---|---|
| | | | DF Forwarding Table List | M | - |

M = Mandatory

*Table 4.154: Directed Forwarding Client elements and procedures*

#### 4.4.6.2 Behavior

This section describes behaviors for procedures and messages for the Directed Forwarding Client model.

An element can send any Directed Forwarding Client message at any time to query or change the Directed Forwarding configuration state of a peer element in a subnet. Directed Forwarding Client messages shall be secured using a device key (DevKey).

##### *4.4.6.2.1 Directed Forwarding procedure*

To determine the Directed Forwarding state of a Directed Forwarding Server in a given subnet, a Directed Forwarding Client shall send a DF Directed Forwarding Get message over the subnet. The response is a DF Directed Forwarding Status message from the subnet that contains the Directed Forwarding state of the subnet.

To set the Directed Forwarding state of a Directed Forwarding Server with acknowledgment in a given subnet, a Directed Forwarding Client shall send a DF Directed Forwarding Set message over the subnet. The response is a DF Directed Forwarding Status message from the subnet; the message contains the Directed Forwarding state of the subnet.

##### *4.4.6.2.2 Path Metric procedure*

To determine the Path Metric Type value and the Path Lifetime value of the Path Metric state of a Directed Forwarding Server in a given subnet, a Directed Forwarding Client shall send a DF Path Metric Get message over the subnet. The response is a DF Path Metric Status message from the subnet, which contains the Path Metric state of the subnet, with the current values of the Path Metric Type state and the Path Lifetime state.

To set the Path Metric Type value and the Path Lifetime value of the Path Metric state of a Directed Forwarding Server in a given subnet with acknowledgment, a Directed Forwarding Client shall send a DF Path Metric Set message over the subnet. The response is a DF Path Metric Status message sent from the subnet, which contains the Path Metric state of the subnet, with the current values of the Path Metric Type state and the Path Lifetime state.

##### *4.4.6.2.3 Path Request Transmit procedure*

To determine the Concurrent Initialization state and the Path Request Interval state in the Path Request Transmit composite state of a Directed Forwarding Server in a given subnet, a Directed Forwarding Client shall send a DF Path Request Transmit Get message over the subnet. The response is a DF Path

Request Transmit Status message from the subnet that contains the Path Request Transmit state of the subnet, with the current values of the Concurrent Initialization state and the Path Request Interval state.

To set the Concurrent Initialization state and the Path Request Interval state in the Path Request Transmit composite state of a Directed Forwarding Server with acknowledgment in a given subnet, a Directed Forwarding Client shall send a DF Path Request Transmit Set message over the subnet. The response is a DF Path Request Transmit Status message from the subnet, which contains the Path Request Transmit state of the subnet with the current values of the Concurrent Initialization state and the Path Request Interval state.

##### *4.4.6.2.4 Neighbor Information procedure*

To determine the Neighborhood state and the Neighbor Information Options state of a Directed Forwarding Server in a given subnet, a Directed Forwarding Client shall send a DF Neighbor Information Get message over the subnet. The response is a DF Neighbor Information Status message from the subnet, which contains the Neighborhood state and the Neighbor Information Options state of the subnet.

To set the Neighborhood state and the Neighbor Information Options state of a Directed Forwarding Server in a given subnet with acknowledgment, a Directed Forwarding Client shall send a DF Neighbor Information Set message over the subnet. The response is a DF Neighbor Information Status message from the subnet, which contains the Neighborhood state and the Neighbor Information Options state of the subnet.

##### *4.4.6.2.5 Forwarding Table procedure*

To determine the Forwarding Table state of a Directed Forwarding Server in a given subnet, a Directed Forwarding Client shall send over the subnet a DF Forwarding Table Get message, specifying the limits of the range of indexes defined in Table 4.126. The response is a DF Forwarding Table List message from the subnet, which either contains complete path entries in the Forwarding Table state of the subnet within the specified index range or is empty.

Upon receiving a DF Forwarding Table List message from a subnet, a Directed Forwarding Client can have a partial or a full view of the Forwarding Table state of a Directed Forwarding Server in the subnet. To have a complete view of the Forwarding Table state of the subnet, a Directed Forwarding Client may repeat sending over the subnet DF Forwarding Table Get messages, specifying index ranges that do not include indexes of path entries already got with previous DF Forwarding Table List messages.

To add a fixed path entry or update an existing fixed path entry in the Forwarding Table state of a Directed Forwarding Server in a given subnet with acknowledgment, a Directed Forwarding Client shall send a DF Forwarding Table Add message over the subnet. The response is a DF Forwarding Table Status message from the subnet, which contains the fixed path entry to be added or updated.

To delete a fixed or a non-fixed path entry from the Forwarding Table state of a Directed Forwarding Server in a given subnet with acknowledgment, a Directed Forwarding Client shall send a DF Forwarding Table Delete message over the subnet. The response is a DF Forwarding Table Status message from the subnet, which contains the path entry to be deleted.

### 4.4.7 Summary of SIG Model IDs

Table 4.155 provides a summary of SIG Model IDs defined within this specification. For a complete list, refer to the Assigned Numbers page [4].

| Model Name | SIG Model ID |
|---|---|
| Configuration Server | 0x0000 |
| Configuration Client | 0x0001 |
| Health Server | 0x0002 |
| Health Client | 0x0003 |
| Directed Forwarding Server | TBD |
| Directed Forwarding Client | TBD |

*Table 4.155: Summary of SIG Model IDs*

# 5 Provisioning

Provisioning is a process of adding an unprovisioned device to a mesh network, managed by a Provisioner. A Provisioner provides the unprovisioned device with provisioning data that allows it to become a mesh node. The provisioning data includes a network key, the current IV Index, and the unicast address for each element.

A Provisioner is typically a smart phone or other mobile computing device. Although only a single Provisioner is required on a network to do provisioning, multiple Provisioners may be used. The method to share cached data and coordinate across multiple Provisioners is implementation specific.

To provision a device, the provisioning bearer must be established between a Provisioner and a device. A device can be identified to a Provisioner by its Device UUID and other supplementary information that may also be provided.

After the provisioning bearer is established, the Provisioner establishes a shared secret with the device using an Elliptic Curve Diffie-Hellman (ECDH) protocol. It then authenticates the device using OOB information that is specific to that device. Such OOB information may include a public key of the device, a long secret, the requirement to input a value into the device, or the requirement to output a value on that device. Such OOB information also enables the authentication of that device. Once the device has been authenticated, the provisioning data is transmitted to the device encrypted with a key derived from that shared secret. The device key is derived from the ECDHSecret and ProvisioningSalt.

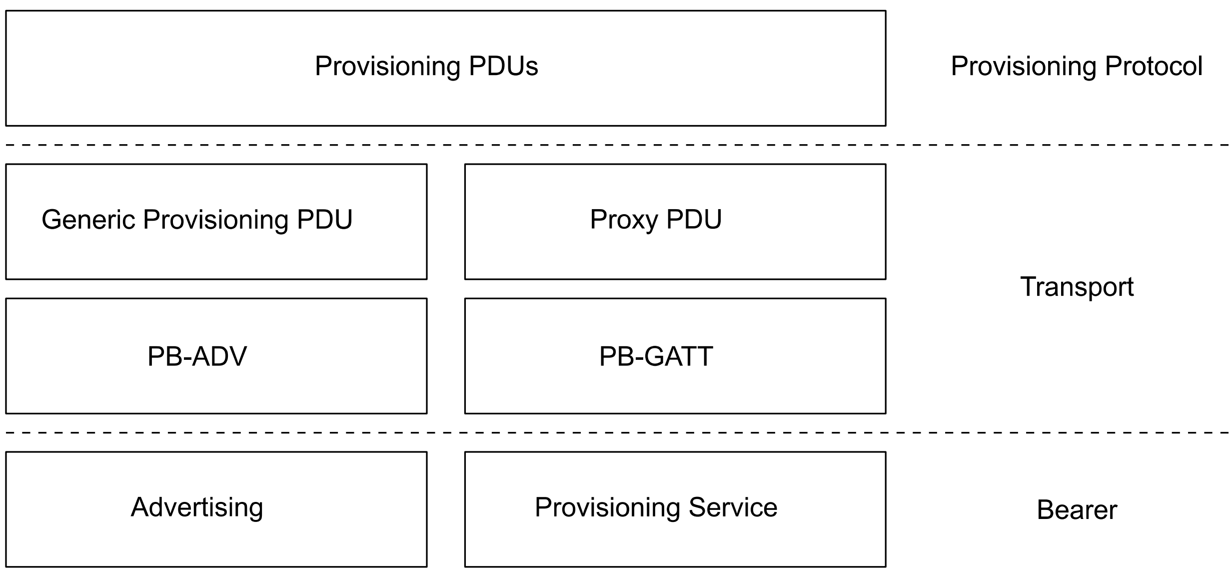


*Figure 5.1: Provisioning protocol stack*

Provisioning uses a layered architecture, illustrated by Figure 5.1. Provisioning of a device is done using the provisioning protocol that sends provisioning PDUs. The provisioning PDUs are transmitted to an unprovisioned device using a generic provisioning layer. This layer defines how the provisioning PDUs are transmitted as transactions that can be segmented and reassembled. These transactions are sent over a provisioning bearer. The provisioning bearers define how sessions are established such that the transactions from the generic provisioning layer can be delivered to a single device. Finally, at the bottom of the provisioning architecture are the bearers.

## 5.1 Endianness

Unless stated otherwise, all multiple-octet numeric values in this layer shall be “big endian”, as described in Section 3.1.1.1.

## 5.2 Provisioning bearer layer

A provisioning bearer layer enables the transportation of Provisioning PDUs between a Provisioner and an unprovisioned device. Two provisioning bearers are defined:

- PB-ADV (see Section 5.2.1)
- PB-GATT (see Section 5.2.2)

An unprovisioned device may support PB-ADV and may support PB-GATT. It is strongly recommended to support PB-ADV and PB-GATT.

A Provisioner shall support at least one of PB-ADV or PB-GATT. It is strongly recommended to support PB-ADV.

### 5.2.1 PB-ADV

PB-ADV is a provisioning bearer used to provision a device using Generic Provisioning PDUs (see Section 5.3) over the advertising channels. The provisioning mechanism is session based. An unprovisioned device shall support only one session at a time. There is no such limitation for a Provisioner. A session is established using the Link Establishment procedure (see Section 5.3.2).

The PB-ADV bearer is used for transmitting Generic Provisioning PDUs. The PB-ADV bearer MTU (Maximum Transmission Unit) size is 24 octets.

When using PB-ADV, a Generic Provisioning PDU shall be sent using the PB-ADV AD Type identified by «PB-ADV», as defined in [4].

A device supporting PB-ADV should perform passive scanning with a duty cycle as close to 100% as possible in order to avoid missing any incoming Generic Provisioning PDUs.

The PB-ADV AD Type contains a PB-ADV PDU. The format of this AD Type is defined in Table 5.1.

| Field | Size (octets) | Description |
|---|---|---|
| Length | 1 | Length of the AD Type and Contents |
| AD Type | 1 | «PB-ADV» |

| | | |
|---|---|---|
| Contents | variable | PB-ADV PDU |

*Table 5.1: PB-ADV AD Type*

Any advertisement using the PB-ADV AD Type shall be non-connectable and non-scannable undirected advertising events. If a node receives a PB-ADV AD Type in a connectable or scannable advertising event, the message shall be ignored.

The format of the PB-ADV PDU is defined in Table 5.2.

| Field | Size (octets) | Description |
|---|---|---|
| Link ID | 4 | The identifier of a link |
| Transaction Number | 1 | The number for identifying a transaction |
| Generic Provisioning PDU | 1–24 | Generic Provisioning PDU being transferred |

*Table 5.2: PB-ADV PDU format*

The Link ID is used to identify a link between two devices.

The Transaction Number field contains a one-octet value used to identify each individual Generic Provisioning PDU sent by the device. When a Provisioning PDU that does not fit in a single PB-ADV PDU is segmented, all segments are sent using the same Transaction Number field value. When a Provisioning PDU is retransmitted, the Transaction Number field is not changed.

Transport specific messages are defined to establish and terminate the link between two devices (see Section 5.3.1.4).

The following rules shall be implemented when sending a PB-ADV PDU:

When the PB-ADV PDU contains a Provisioning Bearer Control PDU, the Transaction Number field shall be set to 0 and ignored upon reception.

When the Provisioner is sending a Provisioning PDU for the first time over an open provisioning link, it shall start with a Transaction Number field value of 0x00. The Provisioner shall increment the field value by one for each new Provisioning PDU it is sending for the duration of the provisioning link. If the field value has reached 0x7F, it shall wrap to 0x00 on sending the next Provisioning PDU.

When the unprovisioned device is sending a Provisioning PDU for the first time over an open provisioning link, it shall start with a Transaction Number field value of 0x80. The Device shall increment the field value by one for each new Provisioning PDU it is sending for the duration of the provisioning link. If the field value has reached 0xFF, it shall wrap to 0x80 on sending the next Provisioning PDU.

When a device is receiving a Provisioning PDU, it shall set the Transaction Number field to the value of the Transaction Number field of the PB-ADV PDUs being received during the transaction.

When a device is sending a Transaction Acknowledgement PDU, the Transaction Number field shall be set to the value of the Transaction Number field of the PB-ADV PDUs transporting the Provisioning PDU being acknowledged.

### 5.2.2 PB-GATT

PB-GATT is a provisioning bearer used to provision a device using Proxy PDUs (see Section 6.3) to encapsulate Provisioning PDUs (see Section 5.4) within the Mesh Provisioning Service (see Section 7.1). PB-GATT is provided for support when a Provisioner does not support PB-ADV due to limitations of the application interfaces.

Note: It is recommended that the connection interval for the connection between a Provisioner and device is between 250 and 1000 milliseconds (implementation specific) to enable very low power operation for the device and allow the device to calculate the Diffie-Hellman shared secret without wasting significant energy to maintain an idle link.

The Mesh Provisioning Server shall be able to receive a single Proxy PDU (see Section 6.3) in a single Write Command ATT PDU.

The Mesh Provisioning Server shall use a single Handle Value Notification ATT PDU to send Provisioning PDUs to a Provisioner.

If the negotiated ATT_MTU is less than a required Proxy PDU size, the transmission of the Mesh Provisioning Data In and Mesh Provisioning Out characteristics always needs to be fragmented and reassembled. Each PDU shall be fully reassembled before processing.

The Mesh Provisioning Server shall be able to receive a Proxy PDU in one or several ATT PDUs. The Mesh Provision Server shall use one or several Handle Value Notification ATT PDUs to send a Proxy PDU to the Provision Client depending on the size of the message and negotiated ATT_MTU.

Mesh Provisioning Data In and Mesh Provisioning Data Out Characteristic Format are using Proxy PDU Format defined in Section 6.3.1.

## 5.3 Generic Provisioning layer

The Generic Provisioning layer is responsible for transport of Generic Provisioning PDUs over an unreliable connectionless provisioning bearer. This layer also defines Generic Provisioning PDUs.

The Generic Provisioning PDU format consists of a Generic Provisioning Control (GPC) field followed by a variable length Generic Provisioning Payload field as illustrated in Figure 5.2 and defined in Table 5.3.

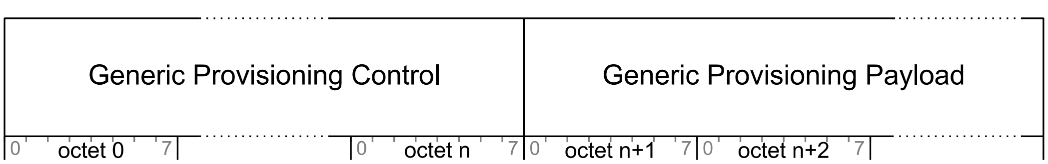


*Figure 5.2: Generic Provisioning PDU format*

| Field | Size (octets) | Description |
|---|---|---|
| Generic Provisioning Control | 1–17 | Generic Provisioning Control field |
| Generic Provisioning Payload | 0–64 | Generic Provisioning Payload (segments of the Provisioning PDU) |

*Table 5.3: Generic Provisioning PDU format*

The two least significant bits of the first octet of the Generic Provisioning Control field contain a Generic Provisioning Control Format (GPCF) field that determines the format of the Generic Provisioning Control field. The GPCF field is an enumeration with the values shown in Table 5.4.

| Value | Description |
|---|---|
| 0b00 | Transaction Start |
| 0b01 | Transaction Acknowledgment |
| 0b10 | Transaction Continuation |
| 0b11 | Provisioning Bearer Control |

*Table 5.4: Generic Provisioning Control Format field values*

The format of the GPC field for each format type is defined in Section 5.3.1.

## 5.3.1 Generic Provisioning PDU types

### 5.3.1.1 Transaction Start PDU

The Transaction Start PDU is used to start the transmission of a segmented message. The Generic Provisioning Control field of a Transaction Start PDU is illustrated in Figure 5.3 and shown in Table 5.5.

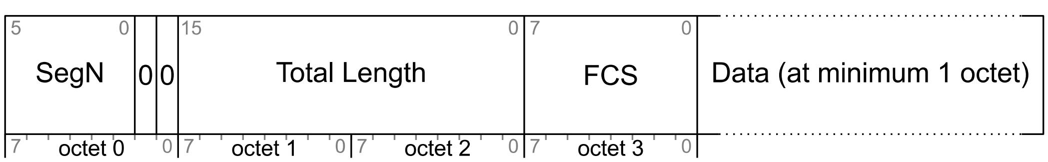


*Figure 5.3: Transaction Start PDU*

| Field | Size (bits) | Description |
|---|---|---|
| SegN | 6 | The last segment number |
| GPCF | 2 | 0b00 = Transaction Start |
| TotalLength | 16 | The number of octets in the Provisioning PDU |
| FCS | 8 | Frame Check Sequence of the Provisioning PDU |

*Table 5.5: Generic Provisioning Control field for Transaction Start PDU*

The SegN field shall be set to the last segment number (zero-based) of this transaction.

The GPCF field shall be set to 0b00.

The TotalLength field shall be set to the number of octets in the Provisioning PDU.

When transmitted using PB-ADV, the FCS field is calculated as defined by 3GPP TS 27.010 with the Polynomial (x^8 + x^2 + x^1 + 1) and is calculated over the Provisioning PDU only.

The Generic Provisioning Payload shall contain segment 0 of the Provisioning PDU.

#### 5.3.1.2 Transaction Acknowledgment PDU

The Transaction Acknowledgment PDU is used to acknowledge a Provisioning PDU. The Generic Provisioning Control field of a Transaction Acknowledgment PDU is illustrated in Figure 5.4 and shown in Table 5.6.

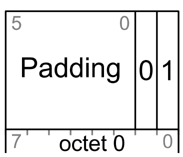


*Figure 5.4: Transaction Acknowledgment PDU*

| Field | Size (bits) | Description |
|---|---|---|
| Padding | 6 | 0b000000; all other values Prohibited |
| GPCF | 2 | 0b01 = Transaction Acknowledgment |

*Table 5.6: Generic Provisioning Control field for Transaction Acknowledgment PDU*

The GPCF field shall be set to 0b01.

The Generic Provisioning Payload is zero length.

#### 5.3.1.3 Transaction Continuation PDU

The Transaction Continuation PDU is used to send additional segments of a Provisioning PDU. The Generic Provisioning Control field of a Transaction Continuation PDU is illustrated in Figure 5.5 and shown in Table 5.7.

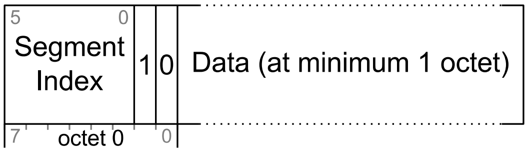


*Figure 5.5: Transaction Continuation PDU*

| **Field** | **Size (bits)** | **Description** |
|---|---|---|
| SegmentIndex | 6 | Segment number of the transaction |
| GPCF | 2 | 0b10 = Transaction Continuation |

*Table 5.7: Generic Provisioning Control field for Transaction Continuation PDU*

The SegmentIndex field shall be set to the segment number contained within this PDU.

The GPCF field shall be set to 0b10.

The Generic Provisioning Payload shall contain segment SegmentIndex of the Provisioning PDU.

#### 5.3.1.4 Provisioning Bearer Control

The Provisioning Bearer Control PDU is used to manage sessions on bearers that have no inherent session management. The Generic Provisioning Control field of a Provisioning Bearer Control PDU is illustrated in Figure 5.6 and shown in Table 5.8. The Provisioning Bearer Control PDUs are defined in the following sections.

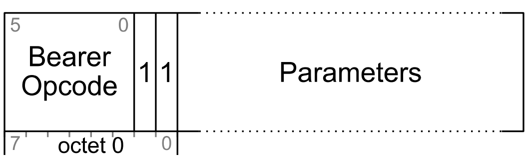


*Figure 5.6: Provisioning Bearer Control PDU*

| **Field** | **Size (bits)** | **Description** |
|---|---|---|
| BearerOpcode | 6 | The opcode for the provisioning bearer control PDUs |
| GPCF | 2 | 0b11 = Provisioning Bearer Control |
| Parameters | variable | Parameters defined by each BearerOpcode |

*Table 5.8: Generic Provisioning Control field for Provisioning Bearer Control PDU*

The BearerOpcode values are defined in Table 5.9.

| **Value** | **Message** | **Notes** |
|---|---|---|

| | | |
|---|---|---|
| 0x00 | Link Open | Open a session on a bearer with a device |
| 0x01 | Link ACK | Acknowledge a session on a bearer |
| 0x02 | Link Close | Close a session on a bearer |
| 0x03–0x3F | RFU | Reserved for Future Use |

*Table 5.9: BearerOpcode field values*

The GPCF field shall be set to 0b11.

The Generic Provisioning Payload is zero length.

The Parameters of each message are defined in the sections that follow.

#### 5.3.1.4.1 Link Open message

The Link Open message is used to establish a link between a Provisioner and an unprovisioned device. The device shall acknowledge this message with the Link ACK message.

Since the device can handle only one link at a time, the reception of a Link Open message shall be ignored when another link is active.

The Parameters field for the Link Open message is defined in Table 5.10 and the message is illustrated in Figure 5.7.

| Field | Size (octets) | Description |
|---|---|---|
| Device UUID | 16 | This is the Device UUID of the chosen unprovisioned device |

*Table 5.10: Parameters field of Link Open message*

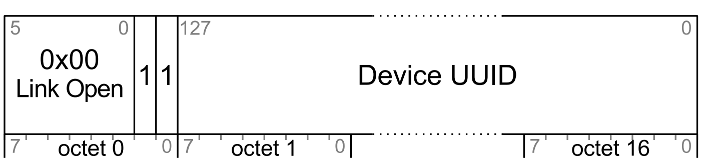


*Figure 5.7: Link Open message format*

#### 5.3.1.4.2 Link ACK message

The Link Ack message is sent to acknowledge the receipt of the Link Open message.

There are no Parameters for this message.

The Link Ack message is illustrated in Figure 5.8.

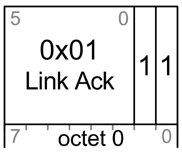


*Figure 5.8: Link Ack message format*

#### 5.3.1.4.3 Link Close message

The Link Close message is used to close a link. Since this message is not acknowledged, the sender shall repeat this message at least three times. Both sides of the link may send this message. This message shall be accepted and processed regardless of the setting of the Reason field.

The Parameters field for the Link Close message is defined in Table 5.11 and the message is illustrated in Figure 5.9.

| Field | Size (octets) | Description |
|---|---|---|
| Reason | 1 | The reason for closing the link |

*Table 5.11: Parameters field of Link Close message*

The Reason field values are defined in Table 5.12.

| Value | Reason | Notes |
|---|---|---|
| 0x00 | Success | The provisioning was successful |
| 0x01 | Timeout | The provisioning transaction timed out |
| 0x02 | Fail | The provisioning failed |
| 0x03–0xFF | Unrecognized | Unrecognized reason that may be defined in the future. |

*Table 5.12: Reason field values*

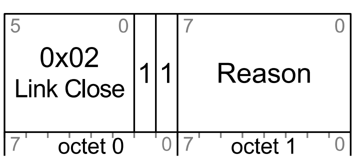


*Figure 5.9: Link Close message format*

### 5.3.2 Link Establishment procedure

The Link Establishment procedure is used to establish a session for a bearer that does not have inherent session management. A session is identified by using a Link ID that is static for the duration of the session and shall be randomly generated to prevent collisions between sessions.

A link is established between a Provisioner and an unprovisioned device for sending provisioning messages. The unprovisioned device is identified by a Device UUID (see Section 3.10.3).

The Provisioner shall scan for unprovisioned devices. Upon receiving an Unprovisioned Device beacon, the Provisioner may establish a link with the device identified by the Device UUID.

The link establishment is started by the Provisioner by sending the Link Open message. The Link Open message contains the Device UUID of the device. On PB-ADV, the PB-ADV PDU format includes a Link ID field.

Unless already being provisioned, i.e., the device has received a Provisioning Invite PDU, the device upon receiving the Link Open message shall accept it by replying with a Link ACK message with the same Link ID.

The link may be closed at any time after link establishment by sending the Link Close message. Either side of the link may send the Link Close message.

The Link Open, Link ACK, and Link Close messages are defined in Section 5.3.1.4.

The message sequence for establishment of a link by ID between a Provisioner and an unprovisioned device is illustrated by Figure 5.10 below.

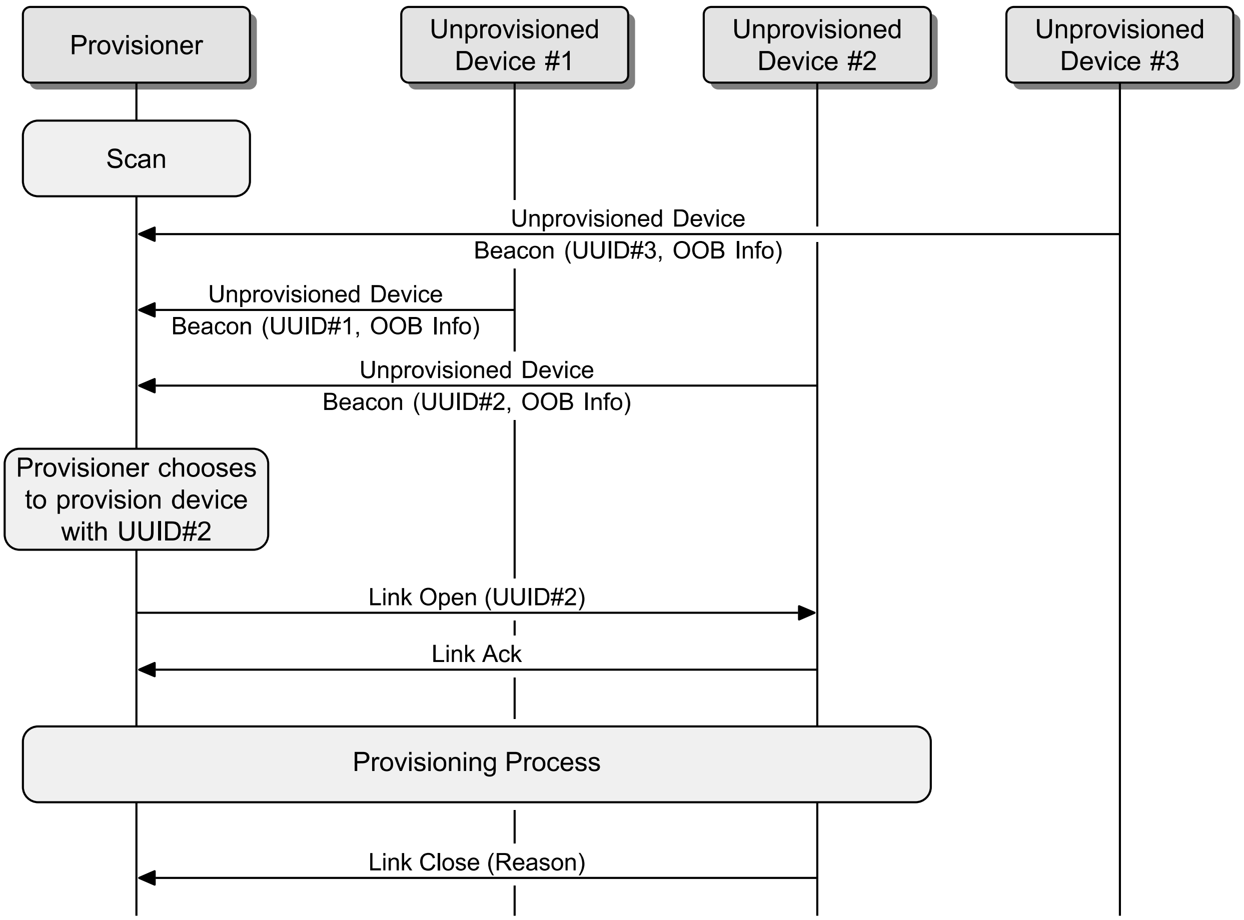


*Figure 5.10: Establishment of Link by ID between a Provisioner and an unprovisioned device*

### 5.3.3 Generic Provisioning behavior

Each Generic Provisioning PDU shall be sent after a random delay between 20 and 50 milliseconds.

Each Provisioning PDU (see Section 5.4) is transmitted as a separate transaction. A transaction may be composed of one or more segments.

The number of segments required to send a Provisioning PDU is determined by the size of the Provisioning PDU. Segments are indexed from 0 to 63. Segment 0 shall be sent using a Transaction Start PDU. All other segments shall be sent using a Transaction Continuation PDU. Each segment of the Provisioning PDU is placed into the Generic Provisioning Payload field of the respective Generic Provisioning PDU.

Each bearer has its own constraints on the maximum size of a Generic Provisioning PDU that can be transmitted by that bearer. Each Generic Provisioning PDU shall be the length of the full MTU for that bearer, except for the last segment of a transaction.

The sender shall send all segments of a transaction in sequence. If the sender does not receive a Transaction Acknowledgment message, the sender shall retransmit all segments of a transaction.

If the sender receives a Transaction Acknowledgment message, then the transaction has completed.

If the sender receives a message with other PDU types, then it shall be dropped.

If the sender does not receive a Transaction Acknowledgment message within 30 seconds after sending the first message in a transaction, the sender shall cancel the transaction, cancel the provisioning process and close the link.

The receiver shall determine the number of segments for a transaction from the Transaction Start PDU.

On the PB-ADV bearer, when the receiver has received all segments of a transaction, the receiver shall calculate the FCS for the received Provisioning PDU, and if it matches the FCS field in the Transaction Start PDU, it shall send a Transaction Acknowledgment PDU after a random delay between 20 and 50 milliseconds.

When a Transaction Acknowledgment PDU has been sent for a given transaction and another segment of the same transaction has been received, another Transaction Acknowledgment PDU shall be sent and segments shall be dropped.

## 5.4 Provisioning protocol

This section defines requirements for Provisioning PDUs, behavior, and security.

### 5.4.1 Provisioning PDUs

The Provisioning PDUs are used to communicate between a Provisioner and a device. Provisioning PDUs are transported using the Generic Provisioning layer.

The first octet of the Provisioning PDU is the Type field and defines the format of the Parameters of the Provisioning PDU.

The Provisioning PDU format is defined in Table 5.13 and illustrated in Figure 5.11.

| Field | Size (bits) | Description |
|---|---|---|
| Padding | 2 | 0b00. All other values are Prohibited |
| Type | 6 | Provisioning PDU Type values (see Table 5.14) |
| Parameters | variable | Message parameters |

*Table 5.13: Provisioning PDU format*

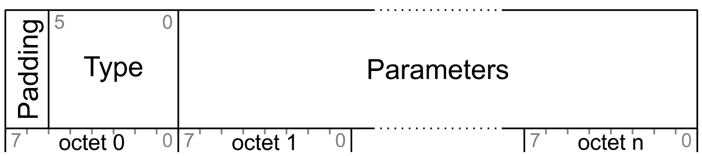


*Figure 5.11: Provisioning PDU format*

The Provisioning PDU Type values are defined in Table 5.14.

| Type | Name | Description |
|---|---|---|
| 0x00 | Provisioning Invite | Invites a device to join a mesh network |
| 0x01 | Provisioning Capabilities | Indicates the capabilities of the device |
| 0x02 | Provisioning Start | Indicates the provisioning method selected by the Provisioner based on the capabilities of the device |
| 0x03 | Provisioning Public Key | Contains the Public Key of the device or the Provisioner |
| 0x04 | Provisioning Input Complete | Indicates that the user has completed inputting a value |
| 0x05 | Provisioning Confirmation | Contains the provisioning confirmation value of the device or the Provisioner |
| 0x06 | Provisioning Random | Contains the provisioning random value of the device or the Provisioner |
| 0x07 | Provisioning Data | Includes the assigned unicast address of the primary element, a network key, NetKey Index, Flags and the IV Index |
| 0x08 | Provisioning Complete | Indicates that provisioning is complete |
| 0x09 | Provisioning Failed | Indicates that provisioning was unsuccessful |
| 0x0A–0xFF | RFU | Reserved for Future Use |

*Table 5.14: Provisioning PDU types*

#### 5.4.1.1 Provisioning Invite

A Provisioner sends this PDU to indicate to the device that the provisioning process is starting. The format of the parameters for this PDU is defined in Table 5.15.

| Field | Size (octets) | Notes |
|---|---|---|
| Attention Duration | 1 | Attention Timer state (See Section 4.2.9) |

*Table 5.15: Provisioning Invite PDU parameters format*

### 5.4.1.2 Provisioning capabilities

The device sends this PDU to indicate its supported provisioning capabilities to a Provisioner. The format of the parameters for this PDU is defined in Table 5.16.

| Field | Size (octets) | Notes |
|---|---|---|
| Number of Elements | 1 | Number of elements supported by the device (Table 5.17) |
| Algorithms | 2 | Supported algorithms and other capabilities (see Table 5.18) |
| Public Key Type | 1 | Supported public key types (see Table 5.19) |
| Static OOB Type | 1 | Supported static OOB Types (see Table 5.20) |
| Output OOB Size | 1 | Maximum size of Output OOB supported (see Table 5.21) |
| Output OOB Action | 2 | Supported Output OOB Actions (see Table 5.22) |
| Input OOB Size | 1 | Maximum size in octets of Input OOB supported (see Table 5.23) |
| Input OOB Action | 2 | Supported Input OOB Actions (see Table 5.24) |

*Table 5.16: Provisioning capabilities PDU parameters format*

The Number of Elements values are defined in Table 5.17.

| Value | Description |
|---|---|
| 0x00 | Prohibited |
| 0x01–0xFF | The number of elements supported by the device |

*Table 5.17: Number of Elements field values*

The Algorithm values are defined in Table 5.18.

| Bit | Description |
|---|---|
| 0 | FIPS P-256 Elliptic Curve |
| 1–15 | Reserved for Future Use |

*Table 5.18: Algorithms field values*

At least one Algorithm shall be supported by a device.

The Public Key Type values are defined in Table 5.19.

| Bit | Description |
|---|---|
| 0 | Public Key OOB information available |
| 1–7 | Prohibited |

*Table 5.19: Public Key Type field values*

The size of the Public Key OOB information is determined by the selected Algorithm.

The Static OOB Type values are defined in Table 5.20.

| Bit | Description |
|---|---|
| 0 | Static OOB information available |
| 1–7 | Prohibited |

*Table 5.20: Static OOB Type field values*

The maximum size of the Static OOB information is 16 octets.

The Output OOB Size defines the number of digits that can be output (e.g., displayed or spoken) when the value Output Numeric in the Output OOB Action field (see Table 5.22) is selected. The Output OOB Size values are defined in Table 5.21.

| Value | Description |
|---|---|
| 0x00 | The device does not support output OOB |
| 0x01–0x08 | Maximum size in octets supported by the device |
| 0x09–0xFF | Reserved for Future Use |

*Table 5.21: Output OOB Size field values*

The Output OOB Action values are defined in Table 5.22.

| Bit | Description | Data Type |
|---|---|---|
| 0 | Blink | Numeric |
| 1 | Beep | Numeric |
| 2 | Vibrate | Numeric |
| 3 | Output Numeric | Numeric |
| 4 | Output Alphanumeric | Array of octets |
| 5–15 | Reserved for Future Use | n/a |

*Table 5.22: Output OOB Action field values*

The Input OOB Size defines the number of digits that can be entered when the value Input Numeric in the Input OOB Action field (see Table 5.24) is selected. The Input OOB Size values are defined in Table 5.23.

| Value | Description |
|---|---|
| 0x00 | The device does not support Input OOB |
| 0x01–0x08 | Maximum supported size in octets supported by the device |
| 0x09–0xFF | Reserved for Future Use |

*Table 5.23: Input OOB Size field values*

The Input OOB Actions are defined in Table 5.24.

| Bit | Description | Data Type |
|---|---|---|
| 0 | Push | Numeric |
| 1 | Twist | Numeric |
| 2 | Input Number | Numeric |
| 3 | Input Alphanumeric | Array of octets |
| 4–15 | Reserved for Future Use | n/a |

*Table 5.24: Input OOB Action field values*

#### 5.4.1.3 Provisioning Start

A Provisioner sends this PDU to indicate the method it has selected from the options in the Provisioning Capabilities PDU. The format of the parameters for this PDU is defined in Table 5.25.

| Field | Size (octets) | Notes |
|---|---|---|
| Algorithm | 1 | The algorithm used for provisioning (see Table 5.26) |
| Public Key | 1 | Public Key used (see Table 5.27) |
| Authentication Method | 1 | Authentication Method used (see Table 5.28) |
| Authentication Action | 1 | Selected Output OOB Action (see Table 5.29) or Input OOB Action (see Table 5.31) or 0x00 |
| Authentication Size | 1 | Size of the Output OOB used (see Table 5.30) or size of the Input OOB used (see Table 5.32) or 0x00 |

*Table 5.25: Provisioning Start PDU parameters format*

The Algorithm values are defined in Table 5.26.

| Value | Description |
|---|---|
| 0x00 | FIPS P-256 Elliptic Curve |
| 0x01–0xFF | Reserved for Future Use |

*Table 5.26: Algorithm field values*

The Public Key values are defined in Table 5.27.

| Value | Description |
|---|---|
| 0x00 | No OOB Public Key is used |

| | |
|---|---|
| 0x01 | OOB Public Key is used |
| 0x02–0xFF | Prohibited |

*Table 5.27: Public Key field values*

The Authentication Method values are defined in Table 5.28.

| Value | Description |
|---|---|
| 0x00 | No OOB authentication is used |
| 0x01 | Static OOB authentication is used |
| 0x02 | Output OOB authentication is used |
| 0x03 | Input OOB authentication is used |
| 0x04–0xFF | Prohibited |

*Table 5.28: Authentication Method field values*

When the Authentication Method 0x00 (Authentication with No OOB) method is used, the Authentication Action field shall be set to 0x00 and the Authentication Size field shall be set to 0x00.

When the Authentication Method 0x01 (Authentication with Static OOB) method is used, the Authentication Size shall be set to 0x00 and the Authentication Action field shall be set to 0x00.

When the Authentication Method 0x02 (Authentication with Output OOB) is used, the values defined in Table 5.29 and Table 5.30 shall be used to determine the Authentication Action and the Authentication Size.

The Output OOB Action values for the Authentication Size Action field are defined in Table 5.29.

| Value | Description |
|---|---|
| 0x00 | Blink |
| 0x01 | Beep |
| 0x02 | Vibrate |
| 0x03 | Output Numeric |
| 0x04 | Output Alphanumeric |
| 0x05–0xFF | Reserved for Future Use |

*Table 5.29: Output OOB Action values for the Authentication Action field*

The Output OOB Size for the Authentication Size field values are defined in Table 5.30.

| Value | Description |
|---|---|
| 0x00 | Prohibited |

| | |
|---|---|
| 0x01–0x08 | The Output OOB Size in characters to be used |
| 0x09–0xFF | Reserved for Future Use |

*Table 5.30: Output OOB Size values for the Authentication Size field*

When the Authentication Method 0x03 (Authentication with Input OOB) method is used, the values defined in Table 5.31 and Table 5.32 shall be used to determine the Authentication Action and the Authentication Size.

The Input OOB Action values for the Authentication Action field are defined in Table 5.31.

| Value | Description |
|---|---|
| 0x00 | Push |
| 0x01 | Twist |
| 0x02 | Input Numeric |
| 0x03 | Input Alphanumeric |
| 0x04–0xFF | Reserved for Future Use |

*Table 5.31: Input OOB Action values for the Authentication Action field*

The Input OOB Size values for the Authentication Size field are defined in Table 5.32.

| Value | Description |
|---|---|
| 0x00 | Prohibited |
| 0x01–0x08 | The Input OOB size in characters to be used |
| 0x09–0xFF | Reserved for Future Use |

*Table 5.32: Input OOB Size values for the Authentication Size field*

#### 5.4.1.4 Provisioning Public Key

This Provisioner sends this PDU to deliver the public key to be used in the ECDH calculations. The format of the parameters for this PDU is defined in Table 5.33.

| Field | Size (octets) | Notes |
|---|---|---|
| Public Key X | 32 | The X component of public key for the FIPS P-256 algorithm |
| Public Key Y | 32 | The Y component of public key for the FIPS P-256 algorithm |

*Table 5.33: Provisioning Public Key PDU Parameters Format*

#### 5.4.1.5 Provisioning Input Complete

The device sends this PDU when the user completes the input operation.

There are no parameters for this PDU.

#### 5.4.1.6 Provisioning Confirmation

The Provisioner or the device sends this PDU to the peer to confirm the values exchanged so far including the OOB Authentication value and the random number that has yet to be exchanged. The format of the parameters for this PDU is defined in Table 5.34.

| Field | Size (octets) | Notes |
|---|---|---|
| Confirmation | 16 | The values exchanged so far including the OOB Authentication value |

*Table 5.34: Provisioning Confirmation PDU Parameters Format*

#### 5.4.1.7 Provisioning Random

A Provisioner or device sends this PDU to enable the peer device to validate the confirmation. The format of the parameters for this PDU is defined in Table 5.35.

| Field | Size (octets) | Notes |
|---|---|---|
| Random | 16 | The final input to the confirmation |

*Table 5.35: Provisioning Random PDU parameters format*

#### 5.4.1.8 Provisioning Data

A Provisioner sends this PDU to deliver provisioning data to the device. The format of the parameters for this PDU is defined in Table 5.36.

| Field | Size (octets) | Notes |
|---|---|---|
| Encrypted Provisioning Data | 25 | An encrypted and authenticated network key, NetKey Index, Key Refresh Flag, IV Update Flag, current value of the IV Index, and unicast address of the primary element (see Section 5.4.2.5) |
| Provisioning Data MIC | 8 | PDU Integrity Check value |

*Table 5.36: Provisioning Data PDU parameters format*

#### 5.4.1.9 Provisioning Complete

The device sends this PDU to indicate that it has successfully received and processed the provisioning data.

There are no parameters for this PDU.

#### 5.4.1.10 Provisioning Failed

The device sends this PDU if it fails to process a received provisioning protocol PDU. The format of the parameters for this PDU is defined in Table 5.37.

| Field | Size (octets) | Notes |
|---|---|---|
| Error Code | 1 | This represents a specific error in the provisioning protocol encountered by a device |

*Table 5.37: Provisioning Failed PDU parameters format*

The Provisioning Error Codes are defined in Table 5.38.

| Value | Name | Description |
|---|---|---|
| 0x00 | Prohibited | Prohibited |
| 0x01 | Invalid PDU | The provisioning protocol PDU is not recognized by the device |
| 0x02 | Invalid Format | The arguments of the protocol PDUs are outside expected values or the length of the PDU is different than expected |
| 0x03 | Unexpected PDU | The PDU received was not expected at this moment of the procedure |
| 0x04 | Confirmation Failed | The computed confirmation value was not successfully verified |
| 0x05 | Out of Resources | The provisioning protocol cannot be continued due to insufficient resources in the device |
| 0x06 | Decryption Failed | The Data block was not successfully decrypted |
| 0x07 | Unexpected Error | An unexpected error occurred that may not be recoverable |
| 0x08 | Cannot Assign Addresses | The device cannot assign consecutive unicast addresses to all elements |
| 0x09–0xFF | RFU | Reserved for Future Use |

*Table 5.38: Provisioning error codes*

### 5.4.2 Provisioning behavior

Provisioning is performed using a five-step process: beaconing, invitation, exchanging public keys, authentication, and distribution of the provisioning data, as illustrated by Figure 5.12.

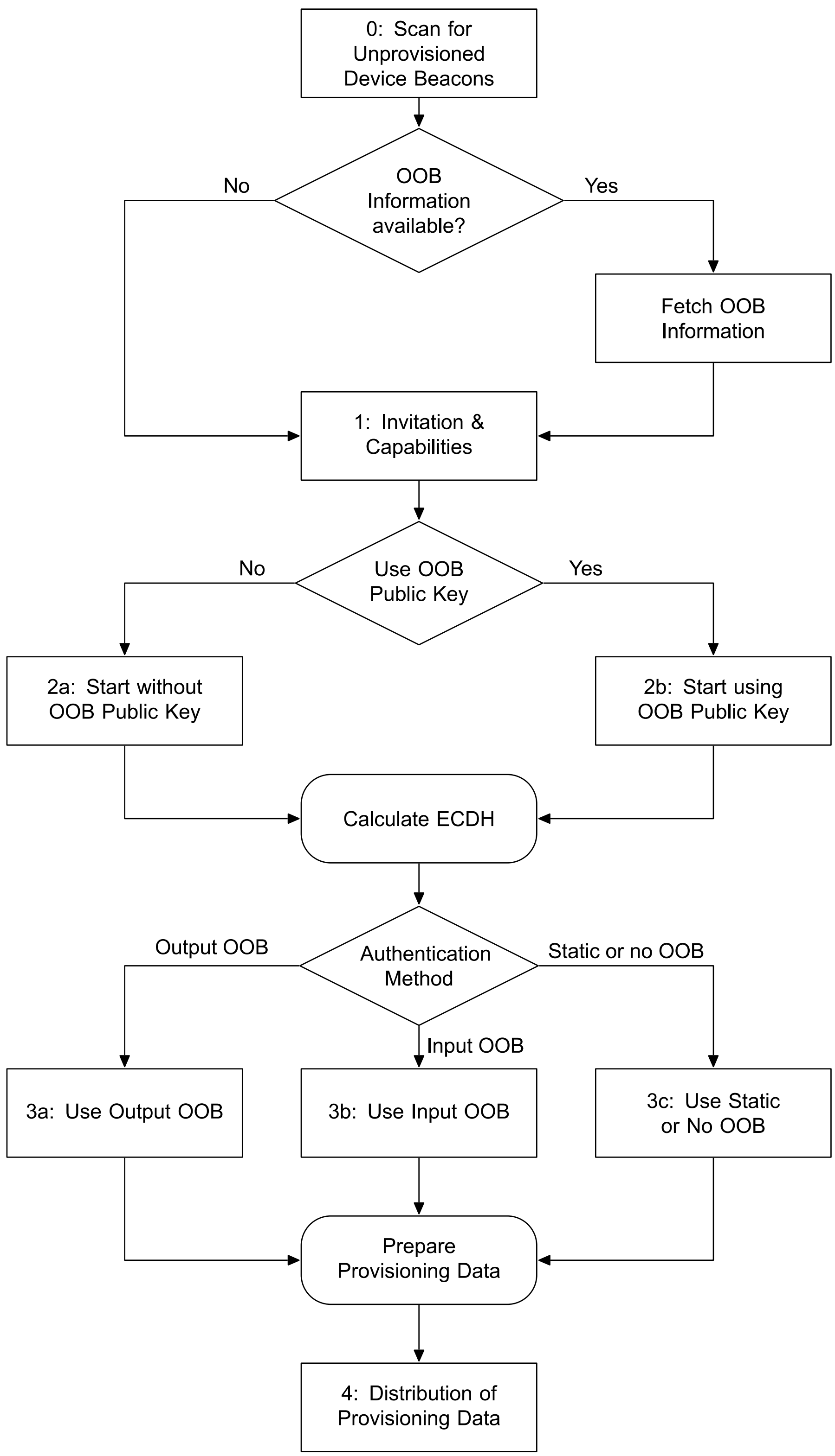
0: Scan for Unprovisioned Device Beacons
OOB Information available?
No
Yes
Fetch OOB Information
1: Invitation & Capabilities
Use OOB Public Key
No
Yes
2a: Start without OOB Public Key
2b: Start using OOB Public Key
Calculate ECDH
Authentication Method
Output OOB
Static or no OOB
Input OOB
3a: Use Output OOB
3b: Use Input OOB
3c: Use Static or No OOB
Prepare Provisioning Data
4: Distribution of Provisioning Data

*Figure 5.12: Provisioning behavior*

#### 5.4.2.1 Beaconing

A device that supports PB-ADV, has not been provisioned, and is not in the process of being provisioned, shall advertise the Unprovisioned Device beacon, as defined in Section 3.9.2, otherwise a device shall not advertise the Unprovisioned Device beacon. When a device has not been provisioned, it is recommended to use anonymous advertising [2], a non-resolvable private address, or a resolvable private address. This beacon may indicate availability of OOB data that allows a Provisioner to prompt the user to collect this OOB data before the next step.

#### 5.4.2.2 Invitation

After establishing a provisioning bearer, a Provisioner shall send a Provisioning Invite PDU and the device shall respond with a Provisioning Capabilities PDU. The Provisioning Invite PDU includes an Attention Duration field, used to determine how long the primary element of the device identifies itself using the Attention Timer, as described in Section 4.2.9. If the provisioning bearer is dropped, the device shall set the Attention Timer state of the primary element to 0x00 (Off). This Provisioning Capabilities PDU includes the information on the number of elements the device supports, the set of security algorithms supported, the availability of its public key using an OOB technology, the ability for this device to output a value to the user, the ability for this device to allow a value to be input by the user, and if the device has a block of OOB data that can be used for authentication.

When using the values from the Output OOB Action field, Blink, Beep, or Vibrate, the device shall select a random integer number between 1 and the value inclusive. That random number shall be output as a sequence of events (e.g., by blinking, beeping, or vibrating with a duty cycle of 500 milliseconds on and 500 milliseconds off) with a gap of at least 3 seconds between sequences to allow the user to determine the end of the sequence. When using the value Output Numeric, the value is output (e.g., displayed or spoken) using digits (i.e., ASCII character codes 0x30-0x39). When using the value Output Alphanumeric, the value is output using ASCII digits and uppercase letters (i.e., ASCII character codes 0x30-0x39 and 0x41-0x5A).

When using the value Push from the Input OOB Action field, the Provisioner shall select a random integer number between 1 and the value inclusive. That random number shall be input by the number of push actions. When using the value Twist, the value shall be input by the number of twist actions until the value on the control has been entered. When using the value Input Numeric, the value is input by entering numbers, probably using a numeric keyboard. When using the value Input Alphanumeric, the value is input by entering characters, probably using an alphanumeric keyboard. When the Input OOB Action field has the values Push or Twist, the value is considered to have been input after no further input actions are detected for more than 5 seconds. When the Input OOB Action field has the value Input Numeric or Input Alphanumeric, the value is considered to have been input locally on that device (e.g., by pressing an Enter key).

The message sequence for provisioning invitation is illustrated by Figure 5.13 below.

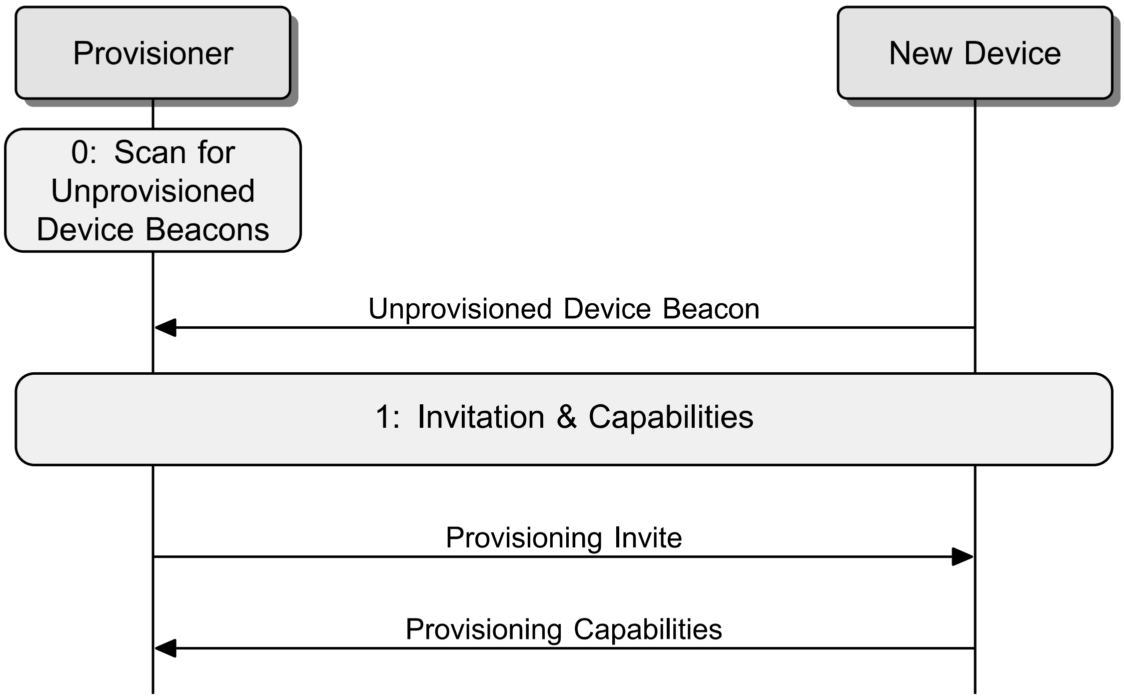


*Figure 5.13: Provisioning invitation*

#### 5.4.2.3 Exchanging public keys

This step has two possibilities depending on the availability of the unprovisioned device public key at a Provisioner's side. Combined with the three possibilities of the authentication step (see Section 5.4.2.4), there are six possible exchange/authentication paths.

Once the Provisioner has determined that it can provision the device, it shall send a Provisioning Start PDU that details which of the six possible paths that the Provisioner has chosen to use.

Upon receiving the Provisioning Start PDU from the Provisioner, the device shall set the Attention Timer to 0x00.

The Provisioner shall select a single algorithm from those offered to it by the New Device in the Provisioning Capability PDU. If the Provisioner does not understand a bit set in this algorithm bit field, it shall ignore that bit and only select from the algorithms it does understand. The Provisioner should choose the strongest algorithm.

If the public key was not available using an OOB technology, then the public keys are exchanged between both devices.

The device shall send its public key if the key is not delivered OOB.

The message sequence for public key exchange when the unprovisioned device public key is unknown is illustrated by Figure 5.14 below.

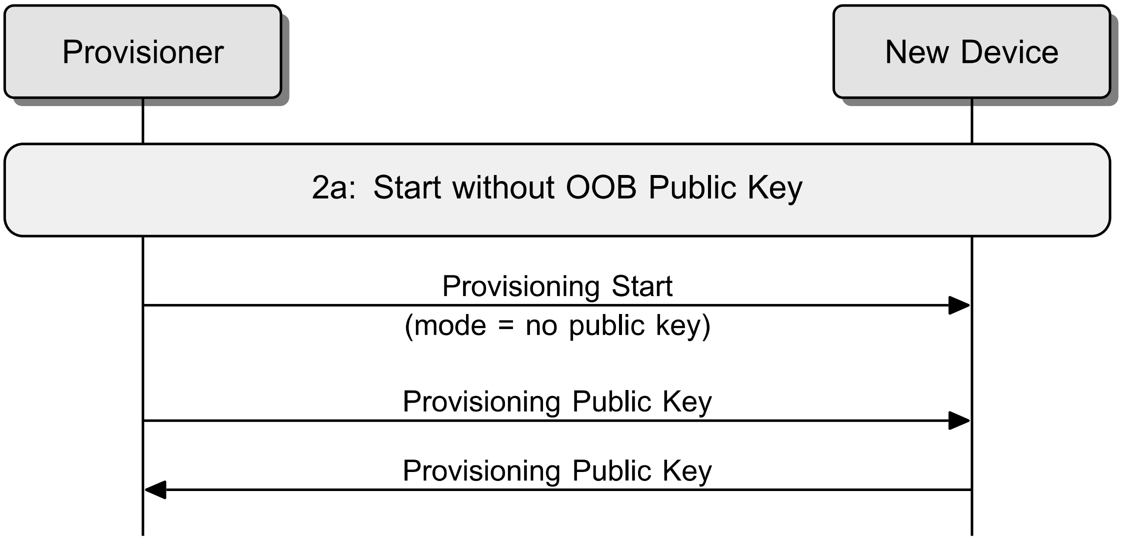


*Figure 5.14: Public key exchange when unprovisioned device public key is unknown*

Otherwise, if the public key is available via an OOB mechanism, then an ephemeral public key shall be transmitted from the Provisioner to the device, and a static public key shall be read from the device using the appropriate OOB technology.

The message sequence for public key exchange when the unprovisioned device public key is OOB is illustrated by Figure 5.15 below.

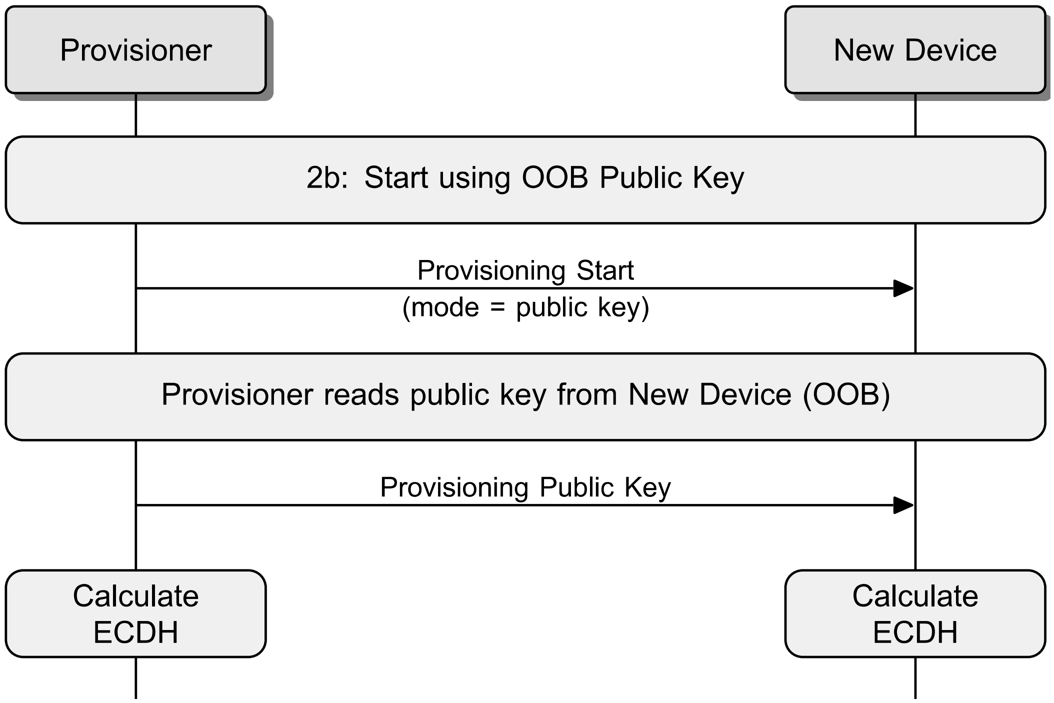


*Figure 5.15: Public key exchange when unprovisioned device public key is out-of-band*

Once the Public Key of the peer device is known, the ECDHSecret shall be computed using:

ECDHSecret = P-256(private key, peer public key)

#### 5.4.2.4 Authentication

If Output OOB authentication is used, then the Provisioning Start PDU would include a request to output either a single digit or multiple digit value on the device using a prescribed action. Examples of output actions may include making a noise, blinking a light, voice, or displaying symbols on a display.

For example, if the device is a door lock that includes an LED, then it would be possible to use the Output OOB authentication with the action that would blink that LED.

The device shall pick a random number and output that number, and the user of the Provisioner shall input the number observed to authenticate that device.

Once input of the number has been performed, a confirmation exchange followed by a random number exchange is performed. The confirmation values are a cryptographic hash of all the values exchanged so far, the random number that is yet to be revealed, and the number that has been input. Once the random numbers are exchanged, each device can authenticate their peer.

The authentication value from Output OOB, Input OOB, or Static OOB is used in AuthValue for the purpose of computing the confirmation value as described below.

The confirmation value of the Provisioner is computed using:

$$\text{ConfirmationProvisioner} = \text{AES-CMAC}_{\text{ConfirmationKey}}\ (\text{RandomProvisioner} \,||\, \text{AuthValue})$$

The confirmation value of the device is computed using:

$$\text{ConfirmationDevice} = \text{AES-CMAC}_{\text{ConfirmationKey}}\ (\text{RandomDevice} \,||\, \text{AuthValue})$$

Where:

ConfirmationKey = k1(ECDHSecret, ConfirmationSalt, “prck”)

ConfirmationSalt = s1(ConfirmationInputs)

ConfirmationInputs = ProvisioningInvitePDUValue || ProvisioningCapabilitiesPDUValue || ProvisioningStartPDUValue || PublicKeyProvisioner || PublicKeyDevice

The ProvisioningInvitePDUValue is the value of the Provisioning Invite PDU fields (excluding the opcode) that was sent or received.

The ProvisioningCapabilitiesPDUValue is the value of the Provisioning Capabilities PDU fields (excluding the opcode) that was sent or received.

The ProvisioningStartPDUValue is the value of the Provisioning Start PDU fields (excluding the opcode) that was sent or received.

The PublicKeyProvisioner is the value of the Public Key X and Public Key Y fields from the Public Key PDU that was sent by the Provisioner.

The PublicKeyDevice is the value of the Public Key X and Public Key Y fields from the Public Key PDU that was sent by the Device or from the delivered OOB public key.

RandomProvisioner is a string of random bits generated by the Provisioner’s random number generator. The random number generator shall be compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1].

RandomDevice is a string of random bits generated by the device’s random number generator. The random number generator shall be compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1].

The AuthValue is a 128-bit value. The computation of AuthValue depends on the data type of the Output OOB Action, Input OOB Action, or Static OOB Type that is used.

If the data type is Binary, the AuthValue is an array of octets. If the value is shorter than 128 bits, the remaining bits shall be set to 0.

For example, if the value is [0x12, 0x34, 0x56], the AuthValue is an array consisting of [0x12, 0x34, 0x56, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00].

If the data type is Numeric, the number shall be represented as an unsigned 128-bit value.

If the data type is Alphanumeric, the AuthValue shall be a concatenation of ASCII codes of the characters.

For example, if the Authentication with Output OOB method is used with Output OOB Action as Blink and value outputted is 5, then the AuthValue shall be 0x00000000000000000000000000000005. The AuthValue is then encoded as defined in Section 5.1 in order to compute confirmation values, resulting in an array consisting of [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05].

For example, if the Authentication with Output OOB method is used with Output OOB Action as Display Number and the displayed number is 019655, then the AuthValue shall be 0x00000000000000000000000000004CC7. The AuthValue is then encoded as defined in Section 5.1 in order to compute confirmation values, resulting in an array consisting of [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4C, 0xC7].

For example, if the Authentication with Output OOB method is used with Output OOB Action as Display String and the displayed string is ”123ABC”, then the AuthValue shall be 0x31323341424300000000000000000000, resulting in an array consisting of [0x31, 0x32, 0x33, 0x41, 0x42, 0x43, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00].

For example, if the Authentication with No OOB method is used, the AuthValue shall be set to 0x00000000000000000000000000000000, which means it is not authenticated, resulting in an array consisting of [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00].

The Provisioner shall send the Provisioning Random PDU after it has received the Provisioning Confirmation PDU. The device shall send the Provisioning Random after verifying the confirmation value against the random number it has received.

The message sequence for authentication with Output OOB is illustrated by Figure 5.16 below.

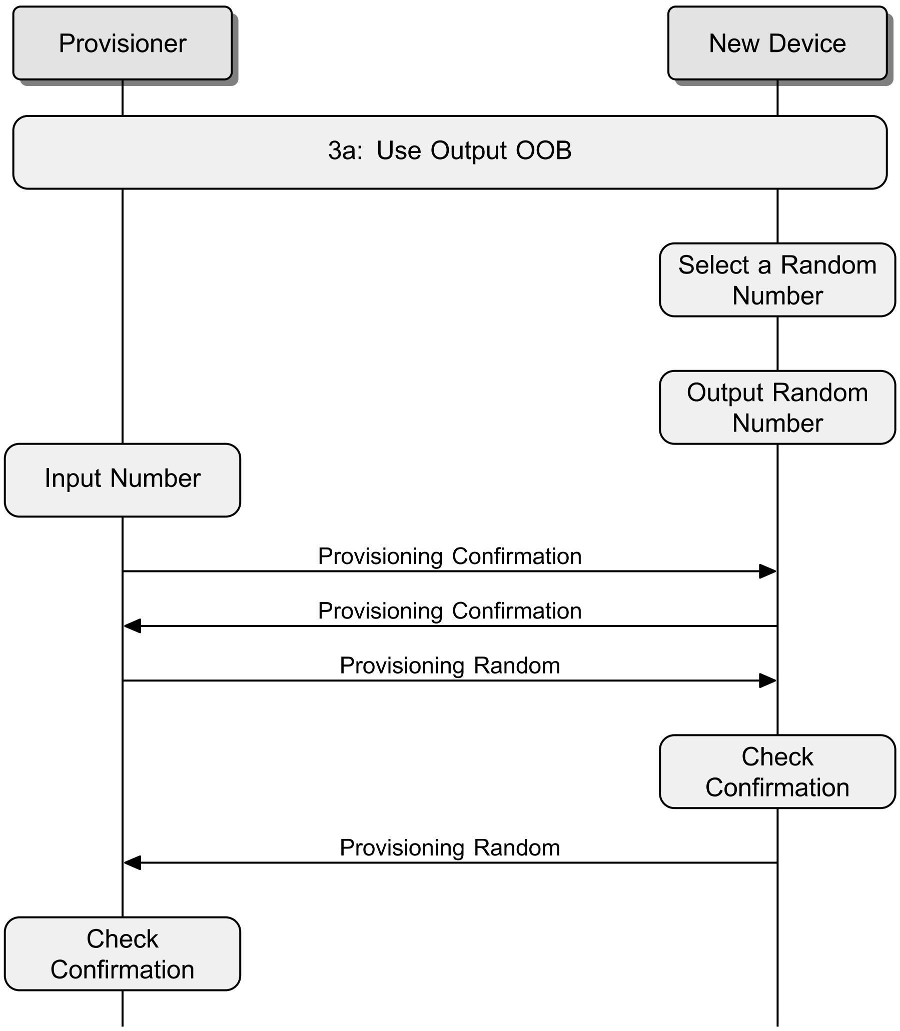


*Figure 5.16: Authentication with Output OOB*

If input OOB authentication is used, then the Provisioner shall generate a random number and display that number to the user. It shall then prompt the user to input that value into the device using an appropriate action.

For example, a light switch may allow the user to input the random number by pressing a button an appropriate number of times.

Once input of the number has been performed, the device shall send the Provisioning Input Complete PDU to the Provisioner to confirm that the device has an input value. A confirmation exchange followed by a random number exchange is performed. The confirmation values are a cryptographic hash of all the values exchanged so far, the random number that is yet to be revealed, and the number that has been input. Once the random numbers are exchanged, each device can authenticate the peer.

The message sequence for authentication with Input OOB is illustrated by Figure 5.17 below.

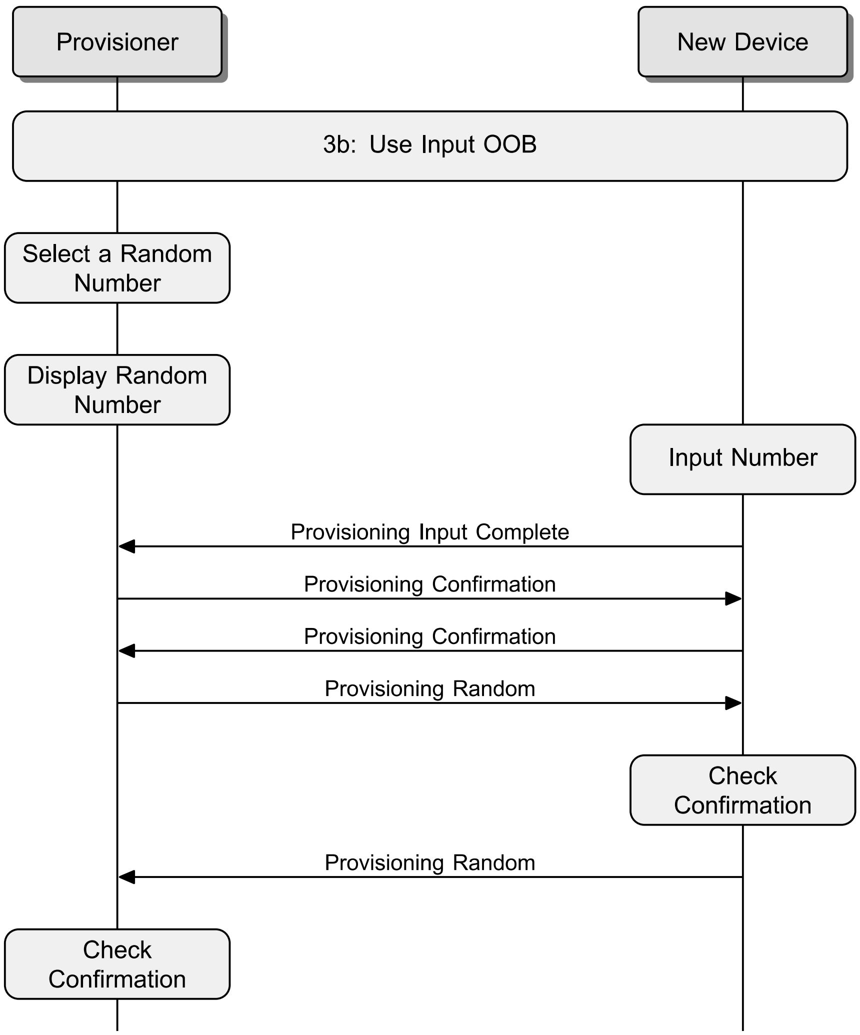


*Figure 5.17: Authentication with Input OOB*

If static OOB authentication is used, or if no output, input, or static authentication was possible, then the Provisioner shall immediately use the confirmation and random number exchanges as detailed above. If a static OOB value is available, then this value shall be included as part of the confirmation value. If no static OOB value is available, then this value shall have a value of zero.

The message sequence for authentication with Static OOB or No OOB is illustrated by Figure 5.18 below.

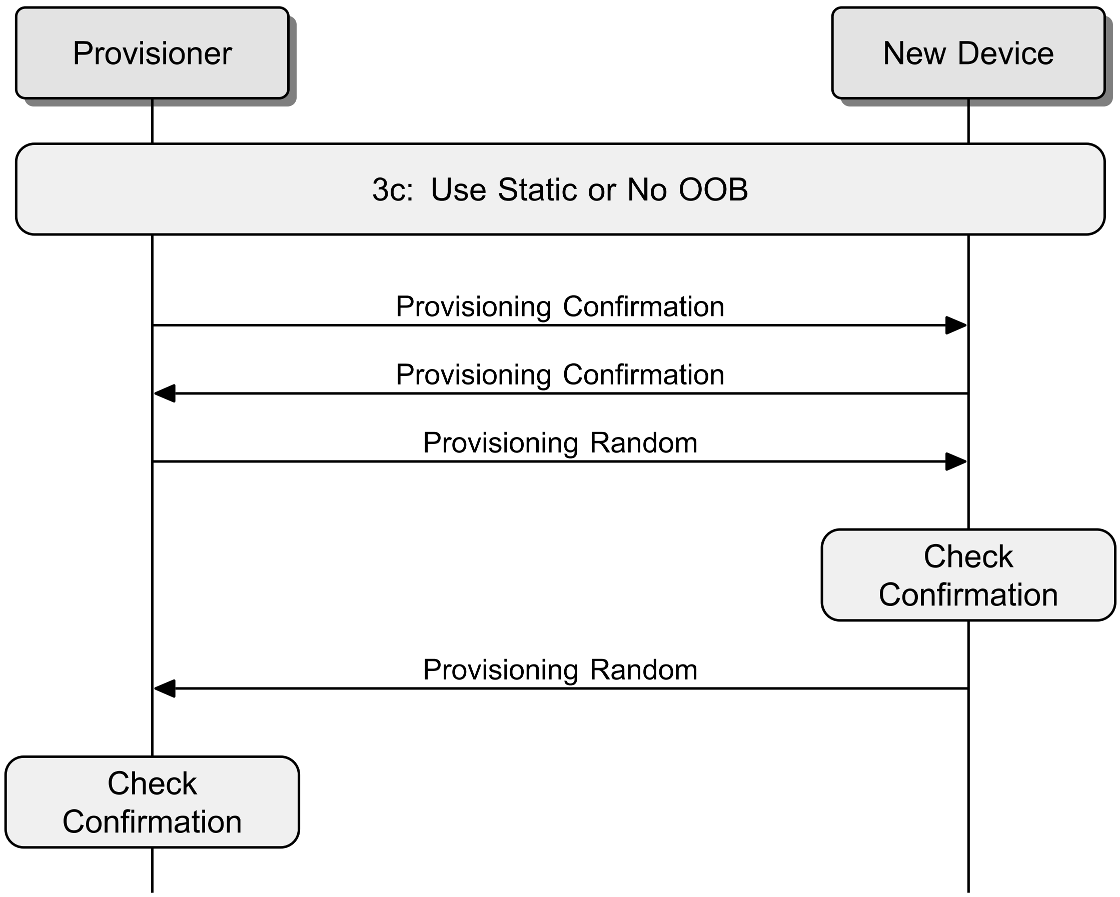


*Figure 5.18: Authentication with static OOB or no OOB*

### 5.4.2.5 Distribution of provisioning data

Once the device has been authenticated, the Provisioner and device shall use the calculated Diffie-Hellman shared secret ECDHSecret and generate a session key from that shared secret. That session key shall then be used to encrypt and authenticate the provisioning data. The Provisioner then shall send the Provisioning Data PDU containing the encrypted and authenticated provisioning data to the device.

The provisioning data format is described in Table 5.39.

| Field | Size (octets) | Notes |
|---|---|---|
| Network Key | 16 | NetKey |
| Key Index | 2 | Index of the NetKey |
| Flags | 1 | Flags bitmask |
| IV Index | 4 | Current value of the IV Index |

| | | |
|---|---|---|
| Unicast Address | 2 | Unicast address of the primary element |

*Table 5.39: Provisioning data format*

The Network Key shall contain the NetKey.

The Key Index field shall identify the global NetKey Index of the Network Key and shall be encoded as defined in Section 4.3.1.1.

The Flags field is defined in Table 5.40 as:

| Bits | Definition |
|---|---|
| 0 | Key Refresh Flag<br>0: Key Refresh Phase 0<br>1: Key Refresh Phase 2 |
| 1 | IV Update Flag<br>0: Normal operation<br>1: IV Update active |
| 2–7 | Reserved for Future Use |

*Table 5.40: Flags field definition*

The IV Index field shall contain the current value of the IV Index.

The Unicast Address shall contain the Unicast Address of the primary element of the node being added to the network.

The Session key shall be derived using the formula:

ProvisioningSalt = s1(ConfirmationSalt || RandomProvisioner || RandomDevice)

SessionKey = k1(ECDHSecret, ProvisioningSalt, “prsk”)

The nonce shall be the 13 least significant octets of:

SessionNonce = k1(ECDHSecret, ProvisioningSalt, “prsn”)

The provisioning data shall be encrypted and authenticated using:

Provisioning Data = Network Key || Key Index || Flags || IV Index || Unicast Address

Encrypted Provisioning Data, Provisioning Data MIC = AES-CCM (SessionKey, SessionNonce, Provisioning Data)

The size of the Provisioning Data MIC is 8 octets.

The Encrypted Provisioning Data and Provisioning Data MIC shall be used as fields in the Provisioning Data PDU. The Provisioner shall send the Provisioning Data PDU to the device.

The device shall then compute the device key as defined in Section 3.8.6.1.

The message sequence for distribution of provisioning data is illustrated by Figure 5.19 below.

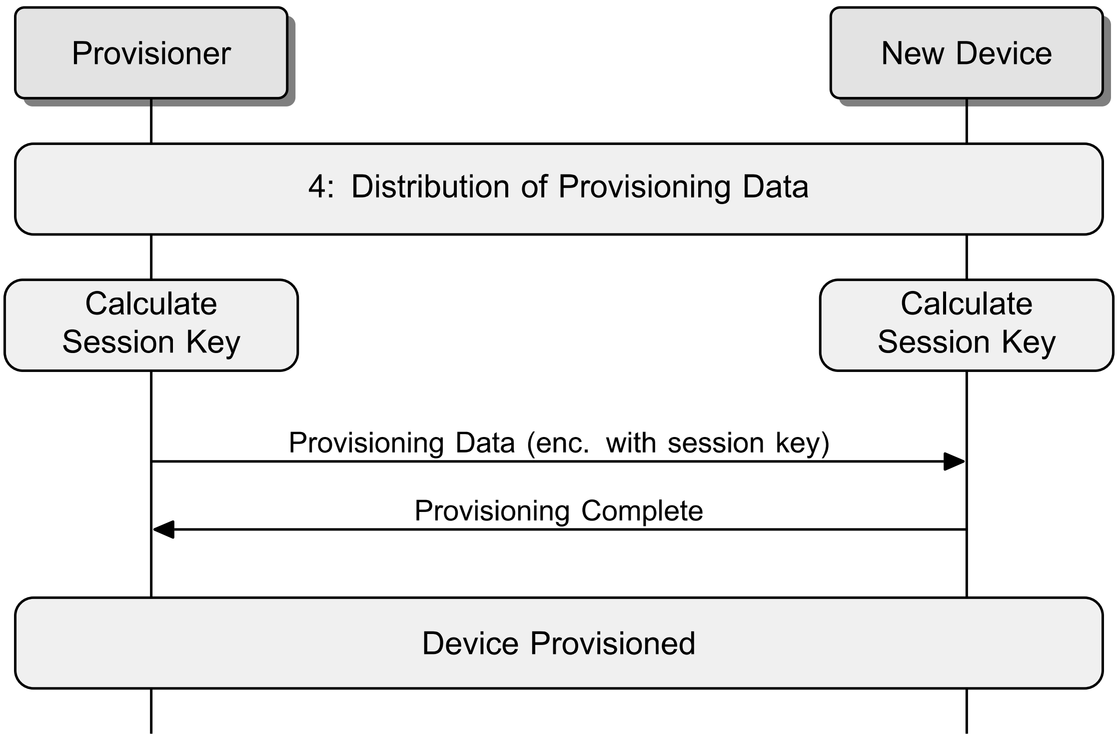


*Figure 5.19: Distribution of provisioning data*

The Provisioner and device calculate the device key using the k1 key derivation function based on the established Diffie-Hellman shared secret ECDHSecret.

Upon receiving the Provisioning Data PDU from the Provisioner, the device shall decrypt and authenticate the provisioning data. Upon successful authentication of the provisioning data, the device shall set the network key (identified by the Key Index), set the IV index, set the IV Update procedure state based on the IV Update Flag, set the Key Refresh Phase based on the Key Refresh Flag, and assign unicast addresses to all device elements starting from primary element using consecutive range of addresses starting from provided unicast address value. Upon successful completion of the address assigning procedure, the device shall respond with a Provisioning Complete PDU to confirm that it has been provisioned. If the address assigning cannot be successfully completed, the device shall assume that provisioning failed and respond with Provisioning Failed PDU with the Error Code parameter set to Cannot Assign Addresses.

Note: After the processing the Provisioning Data PDU from the Provisioner, the 96 hour time limits for changing the IV Update procedure state, as defined in the IV Update procedure, do not apply.

Upon receiving the Provisioning Complete PDU from the device, the Provisioner shall assume that provisioning process is completed successfully and the device is using a consecutive range of address starting from the value of the unicast address. The length of the address range is reported to the Provisioner in Provisioning Capabilities PDU (see Section 5.4.1.2). As a final step in procedure, the Provisioner shall disconnect the provisioning bearer. The device is now a node in the mesh network.

The Provisioner must not reuse unicast addresses that have been allocated to a device and sent in a Provisioning Data PDU until the Provisioner receives an Unprovisioned Device beacon or Service Data for the Mesh Provisioning Service from that same device, identified using the Device UUID of the device.

### 5.4.3 Provisioning security

All devices and Provisioners shall support the FIPS P-256 Elliptic Curve Algorithm.

Provisioning may be secure or insecure. Secure Provisioning requires the following method:

- FIPS P-256 Elliptic Curve Algorithm, a Public Key Type that is not transferred in band (i.e., ”OOB Public Key is used” is selected), and a Static OOB of any size.
- FIPS P-256 Elliptic Curve Algorithm; OOB Action of Input Numeric, Input Alphanumeric, Output Numeric, or Output Alphanumeric; and OOB Size of at least 6 octets.

Otherwise, provisioning is Insecure Provisioning.

It is recommended that devices and Provisioners support Secure Provisioning. A Provisioner may have a policy of only provisioning devices using Secure Provisioning. Devices not supporting Secure Provisioning will not be able to be provisioned by a Provisioner that is only using Secure Provisioning.

#### 5.4.3.1 FIPS P-256 Elliptic Curve definition

The FIPS-P256 curve is defined in FIPS 186-3 [12].

Elliptic curves are specified by p, a, and b in the form of:

$$E: y^2 = x^3 + ax + b \pmod{p}$$

For each value of b, a unique curve can be developed. In NIST P-256:

a = mod (-3, p)

b is defined and its method of generation can be verified by using SHA-1 (with a given seed s and using $b^2c = -27 \pmod{p}$)

The following parameters are given:

- The prime modulus p, order r, base point x-coordinate Gx, base point y- coordinate Gy.
- The integers p and r are given in decimal form; bit strings and field elements are given in hex.

```
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951
r = 115792089210356248762697446949407573529996955224135760342422259061068512044369

b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b

Gx = 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296

Gy = 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5
```

The function P-256 is defined as follows. Given an integer u, $0 < u < r$, and a point V on the curve E, the value P-256(u, V) is computed as the x-coordinate of the $u^{th}$ multiple uV of the point V.

The private keys shall be between 1 and r/2, where r is the Order of the Abelian Group on the elliptic curve (e.g., between 1 and $2^{256}/2$).

### 5.4.3.2 Provisioning key derivation

Figure 5.20 and Figure 5.21 illustrate the derivation of the provisioning keys.

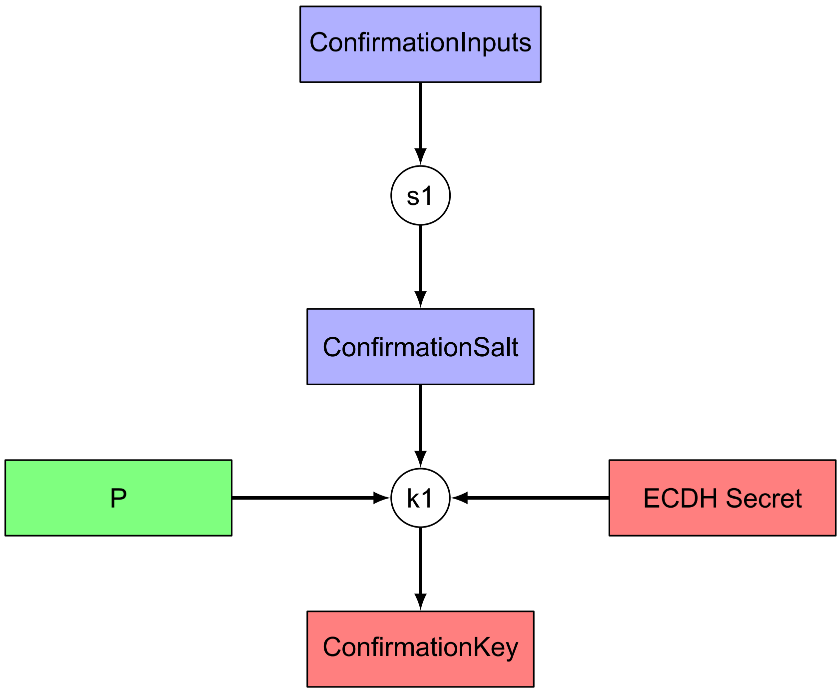


*Figure 5.20: ConfirmationKey derivation*

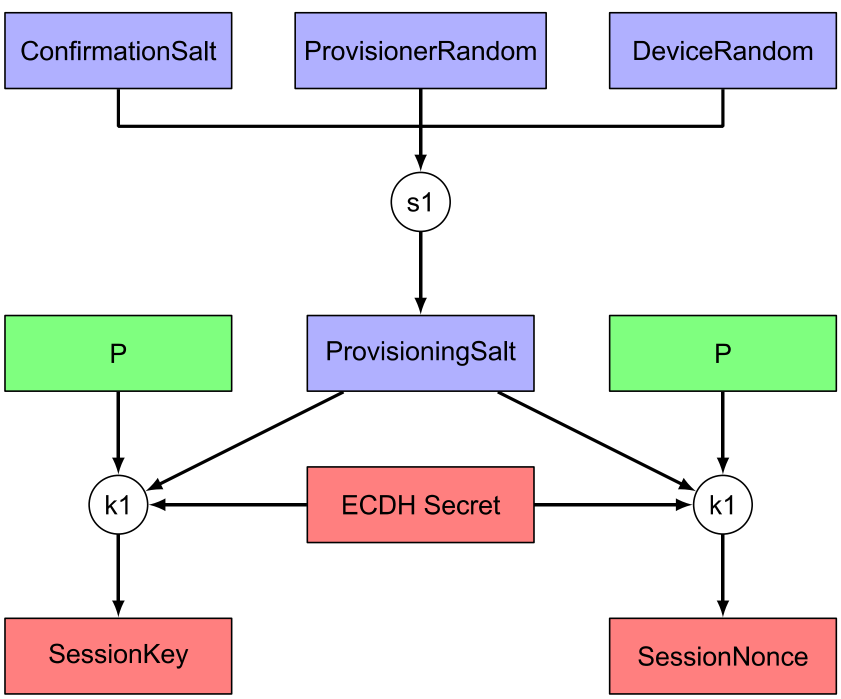


*Figure 5.21: SessionKey and SessionNonce Key derivation*

### 5.4.4 Provisioning errors

When the provisioning protocol fails for any reason, the devices shall permanently delete any stored details. There is no recovery procedure and the whole provisioning procedure must be started from the beginning if the Provisioner chooses to do so.

The provisioning protocol is asymmetric when handling protocol errors. When the Provisioner encounters an error in the provisioning protocol, it shall immediately disconnect the provisioning bearer. Upon an unexpected situation where the provisioning bearer closes, the provisioning protocol has failed.

When the device encounters an error other than a timeout in the provisioning protocol, it shall send the Provisioning Failed PDU with an appropriate Error Code and wait for the closing of the provisioning bearer. The Provisioner, upon receiving the Provisioning Failed PDU, shall assume that the provisioning failed and immediately disconnect the provisioning bearer.

The provisioning protocol shall have a minimum timeout of 60 seconds that is reset each time a provisioning protocol PDU is sent or received. If a PDU is not received before the timeout expires, then the protocol has failed. In the case of protocol timeout, the device shall not send a Provisioning Failed PDU.

# 6 Proxy protocol

The proxy protocol enables nodes to send and receive Network PDUs, mesh beacons, proxy configuration messages and Provisioning PDUs over a connection-oriented bearer.

For example, a node could support GATT but not be able to advertise the Mesh Message AD Type. This node will establish a GATT connection with another node that supports the GATT bearer and the advertising bearer, using the Proxy protocol to forward messages between these bearers.

## 6.1 Endianness

Unless stated otherwise, all multiple-octet numeric values in this protocol shall be “big endian”, as described in Section 3.1.1.1.

## 6.2 Proxy roles

The proxy protocol defines two roles: the Proxy Server and the Proxy Client.

The Proxy Server is a node that supports a mesh bearer using the Proxy protocol and at least one other mesh bearer. For example, the Proxy Server can forward mesh messages between the advertising bearer and the GATT bearer.

The Proxy Client supports a mesh bearer using the Proxy protocol. For example, the Proxy Client can use the GATT bearer to send mesh messages to a node that supports the advertising bearer.

## 6.3 Proxy PDU

A Proxy Client and a Proxy Server exchange Proxy PDUs. Proxy PDUs can contain Network PDUs, mesh beacons, proxy configuration messages or Provisioning PDUs. A single Proxy PDU can contain a full message or a segment of a message. The size of the Proxy PDU is determined by the user of the Proxy protocol. For example, the GATT bearer defines the size of the Proxy PDU based on the ATT_MTU.

### 6.3.1 PDU format

The format of a Proxy PDU is illustrated in Figure 6.1 and defined in Table 6.1.

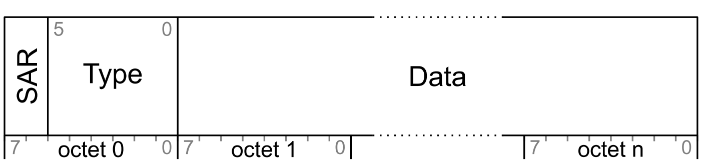


*Figure 6.1: Proxy PDU format*

| Field Name | Size (bits) | Notes |
|---|---|---|
| SAR | 2 | Message segmentation and reassembly information |
| MessageType | 6 | Type of message contained in the PDU |

| Field Name | Size (bits) | Notes |
|---|---|---|
| Data | Variable | Full message or message segment |

*Table 6.1: Proxy PDU format*

The SAR field shown in Table 6.2 identifies the message segmentation and defines the contents of the Data field:

| Value | Description |
|---|---|
| 0b00 | Data field contains a complete message |
| 0b01 | Data field contains the first segment of a message |
| 0b10 | Data field contains a continuation segment of a message |
| 0b11 | Data field contains the last segment of a message |

*Table 6.2: SAR field values*

The MessageType field shown in Table 6.3 identifies the type of message contained in the Proxy PDU:

| Type | Name | Description |
|---|---|---|
| 0x00 | Network PDU | The message is a Network PDU as defined in Section 3.4.4. |
| 0x01 | Mesh Beacon | The message is a mesh beacon as defined in Section 3.9. |
| 0x02 | Proxy Configuration | The message is a proxy configuration message as defined in Section 6.5. |
| 0x03 | Provisioning PDU | The message is a Provisioning PDU as defined in Section 5.4.1. |
| 0x04–0x3F | RFU | Reserved for Future Use. |

*Table 6.3: MessageType values*

The Data field contains a message defined by the MessageType field segmented as defined by the SAR field.

### 6.3.2 Segmentation

Messages sent using the Proxy protocol may be larger than an individual Proxy PDU. To enable such messages to be transmitted, segmentation and reassembly is used.

When sending a message that is less than or equal to the maximum size of a Proxy PDU, the SAR field shall be set to 0b00 and the Data field shall contain the complete message.

When sending a message that is greater than the maximum size of a Proxy PDU, the message is divided into segments that will fill each Proxy PDU except for the last Proxy PDU that may or may not be filled. These segments shall be sent in order and the SAR field of the first segment shall be set to 0b01, the SAR field of the last segment shall be set to 0b11, and all other segments shall have the SAR field set to 0b10.

## 6.4 Proxy filtering

In order to reduce the number of Network PDUs exchanged between a Proxy Client and a Proxy Server, a proxy filter can be used.

The output filter of the network interface (see Section 3.4.5) instantiated by the Proxy Server can be configured by the Proxy Client. This allows the Proxy Client to explicitly request to receive only mesh messages with certain destination addresses. For example, a Proxy Client that is subscribed to a group address may want to only receive packets addressed to the unicast address of one of its elements and to that group address. Thus, the Proxy Client has full control over the packets it receives using the Proxy protocol.

### 6.4.1 Filter types

A proxy filter can be either a white list filter or a black list filter.

A white list filter has an associated white list, which is a list of destination addresses that are of interest for the Proxy Client. The white list filter blocks all destination addresses except those that have been added to the white list.

A black list filter has an associated black list, which is a list of destination addresses that the Proxy Client does not want to receive. The black list filter accepts all destination addresses except those that have been added to the black list.

The white list filter with an empty list is the default filter type. The Proxy Client can change the filter type as well as configure the addresses in the proxy filter.

## 6.5 Proxy configuration messages

In addition to Network PDUs, mesh beacons and Provisioning PDUs, the Proxy Client and Proxy Server can exchange proxy configuration messages using the Proxy protocol.

Proxy configuration messages are used to configure the proxy filters.

The format of a proxy configuration message is identical to a Network PDU as defined in Table 3.7.

The CTL field shall be set to 1.

The TTL field shall be set to 0.

The DST field shall be set to the unassigned address.

The TransportPDU field shall be formatted as shown in Table 6.4 and shall be secured using the master security credentials as defined in Section 3.8.6.3.1, with the proxy nonce as defined in Section 3.8.5.4.

| Field Name | Size (octets) | Notes |
|---|---|---|

| Field Name | Size (octets) | Notes |
|---|---|---|
| Opcode | 1 | Message opcode |
| Parameters | variable | Message parameters |

*Table 6.4: Proxy TransportPDU field format*

The Opcode field values are defined in Table 6.5:

| Value | Opcode | Notes |
|---|---|---|
| 0x00 | Set Filter Type | Sent by a Proxy Client to set the proxy filter type. |
| 0x01 | Add Addresses To Filter | Sent by a Proxy Client to add addresses to the proxy filter list. |
| 0x02 | Remove Addresses From Filter | Sent by a Proxy Client to remove addresses from the proxy filter list. |
| 0x03 | Filter Status | Acknowledgment by a Proxy Server to a Proxy Client to report the status of the proxy filter list. |
| 0x04–0xFF | RFU | Reserved for Future Use |

*Table 6.5: Proxy Configuration message opcodes*

The Parameters field is specific to each opcode and is defined in the following subsections.

### 6.5.1 Set Filter Type

The Set Filter Type message can be sent by a Proxy Client to change the proxy filter type and clear the proxy filter list.

The parameters of this message are defined in Table 6.6:

| Field | Size (octets) | Notes |
|---|---|---|
| FilterType | 1 | The proxy filter type. |

*Table 6.6: Set Filter Type Message Format*

The values for the FilterType field are defined in Table 6.7:

| Value | Notes |
|---|---|
| 0x00 | White list filter |
| 0x01 | Black list filter |
| 0x02–0xFF | Prohibited |

*Table 6.7: FilterType Values*

### 6.5.2 Add Addresses to Filter

The Add Addresses to Filter message is sent by a Proxy Client to add destination addresses to the proxy filter list.

The parameters of this message are defined in Table 6.8:

| Field | Size (octets) | Notes |
|---|---|---|
| AddressArray | 2 * N | List of addresses where N is the number of addresses in this message. |

*Table 6.8: Add Addresses to Filter message format*

The AddressArray field contains a sequence of addresses to be added to the proxy filter list. It may contain any combination of unicast addresses, virtual addresses, and group addresses.

Note: Each address in the AddressArray is a 16-bit value and therefore the 16-bit virtual address and not the Label UUID is used.

### 6.5.3 Remove Addresses from Filter

The Remove Addresses from Filter message is sent by a Proxy Client to remove destination addresses from the proxy filter list.

The parameters of this message are defined in Table 6.9:

| Field | Size (octets) | Notes |
|---|---|---|
| AddressArray | 2 * N | List of addresses where N is the number of addresses in this message. |

*Table 6.9: Remove Addresses from Filter message format*

The AddressArray field contains a sequence of addresses to be removed from the proxy filter list. It may contain any combination of unicast addresses, virtual addresses, or group addresses.

Note: Each address in the AddressArray is a 16-bit value and therefore the 16-bit virtual address and not the Label UUID is used.

### 6.5.4 Filter Status

The Filter Status message is sent by a Proxy Server to report the status of the proxy filter.

The parameters of this message are defined in Table 6.10:

| Field | Size (octets) | Notes |
|---|---|---|

| | | |
|---|---|---|
| FilterType | 1 | White list or black list. |
| ListSize | 2 | Number of addresses in the proxy filter list. |

*Table 6.10: Filter Status message format*

The values for the FilterType field are defined in Table 6.7.

The ListSize field contains the number of addresses in the proxy filter list.

## 6.6 Proxy Server behavior

When a Proxy Client connects to a Proxy Server, an instance of a new bearer is connected to the network layer via a network interface (see Section 3.4.5).

Upon connection, the Proxy Server shall initialize the proxy filter as a white list filter and the white list shall be empty.

If the proxy filter is a white list filter, upon receiving a valid Mesh message from the Proxy Client, the Proxy Server shall add the unicast address contained in the SRC field of the message to the white list.

If the proxy filter is a black list filter, upon receiving a valid Mesh message from the Proxy Client, the Proxy Server shall remove the unicast address contained in the SRC field of the message from the black list.

Upon connection, the Proxy Server shall send a Secure Network Beacon for each known subnet to the Proxy Client.

Upon successfully processing a Secure Network Beacon with new values for the IV Index field or the Flags field, the Proxy Server shall send this Secure Network Beacon to the Proxy Client.

When the Proxy Server is added to a new subnet, it shall send a Secure Network Beacon for that subnet to the Proxy Client.

Upon receiving a Secure Network Beacon from the Proxy Client, the Proxy Server shall process it as defined in Section 3.9.3.1.

When sending a proxy configuration message, a Proxy Server shall set the SRC field to the unicast address of its primary element and the SEQ field shall use the sequence number of its primary element.

If a Proxy Server receives a Set Filter Type message, it shall set the proxy filter type as requested in the message parameter, and it shall clear the proxy filter list. The Proxy Server shall then respond with a Filter Status message.

If the Proxy Server receives an Add Addresses to Filter message, then it shall add these addresses to the proxy filter list and shall respond with a Filter Status message. If one or more addresses contained in the message are already in the list, the Proxy Server shall not add these addresses. If the Proxy Server runs out of space in the proxy filter list, the Proxy Server shall not add these addresses. If the AddressArray field contains the unassigned address, the Proxy Server shall ignore that address.

If the Proxy Server receives a Remove Addresses from Filter message, it shall remove these addresses from the proxy filter list and shall respond with a Filter Status message. If one or more addresses contained in the message were not in the list, the Proxy Server shall ignore these addresses. If the AddressArray field contains the unassigned address, the Proxy Server shall ignore that address.

Upon receiving a message with an unexpected value of the SAR field, the Proxy Server shall disconnect.

Upon receiving a message with the Message Type field set to a value that is Reserved for Future Use, the Proxy Server shall ignore this message.

The timeout for the SAR transfer is 20 seconds. When the timeout expires, the Proxy Server shall disconnect.

## 6.7 Proxy Client behavior

The Proxy Client may send proxy configuration messages to configure the proxy filter.

When sending a proxy configuration message, the Proxy Client shall set the SRC field to the unicast address of its primary element and the SEQ field shall use the sequence number of its primary element. The Proxy Client can determine the state of the proxy filter list when it receives a Filter Status message.

Upon receiving a message with an unexpected value of the SAR field, the Proxy Client shall disconnect.

Upon receiving a message with the Message Type field set to a value that is Reserved for Future Use, the Proxy Client shall ignore this message.

The timeout for the SAR transfer is 20 seconds. When the timeout expires, the Proxy Client shall disconnect.

## 6.8 MSC examples

### 6.8.1 White list filtering

The MSC shown in Figure 6.2 illustrates the white list filtering performed by the Proxy Server, as configured by the Proxy Client.

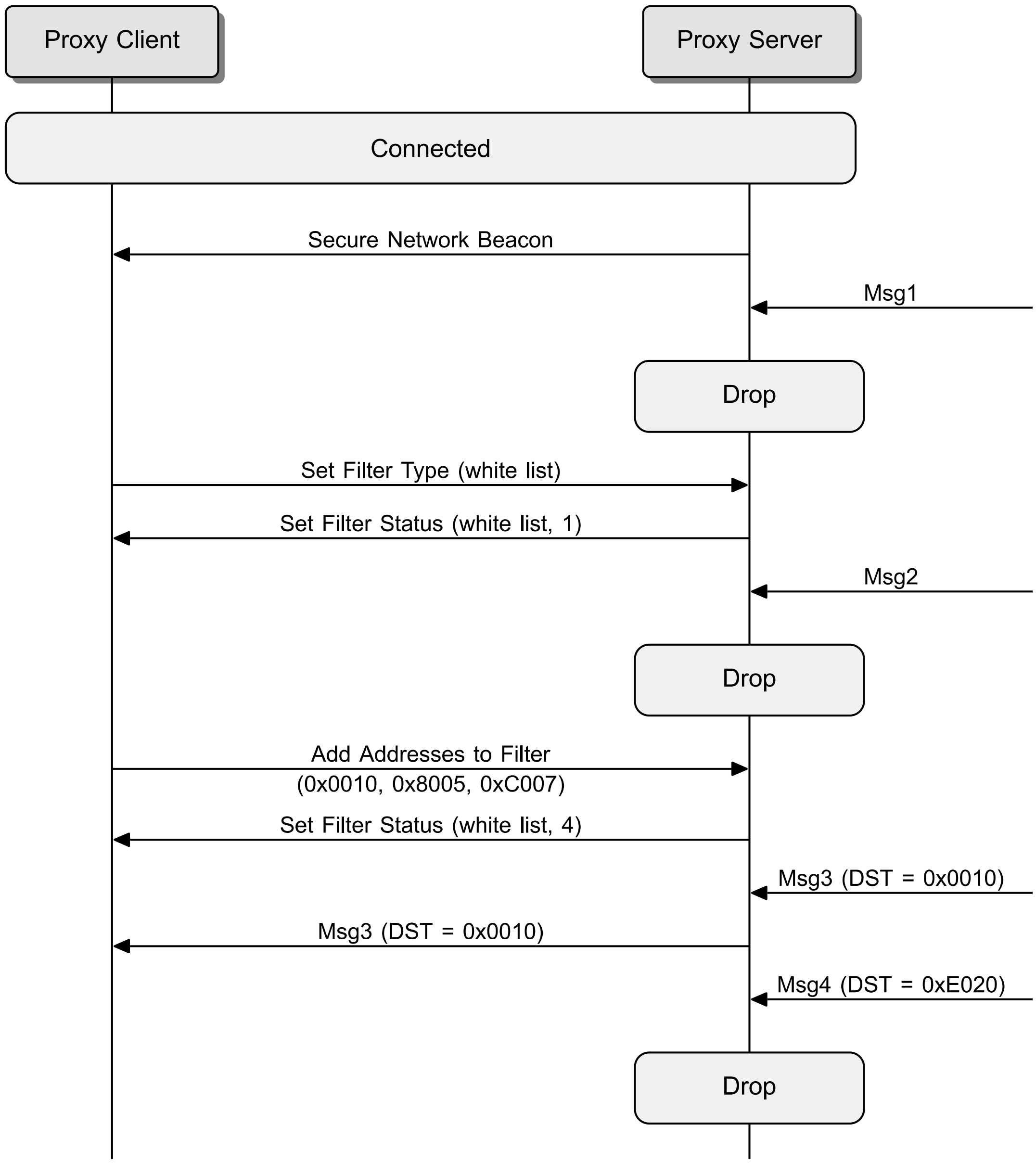


*Figure 6.2: White list filtering*

### 6.8.2 Black list filtering

The MSC shown in Figure 6.3 illustrates the black list filtering performed by the Proxy Server, as configured by the Proxy Client.

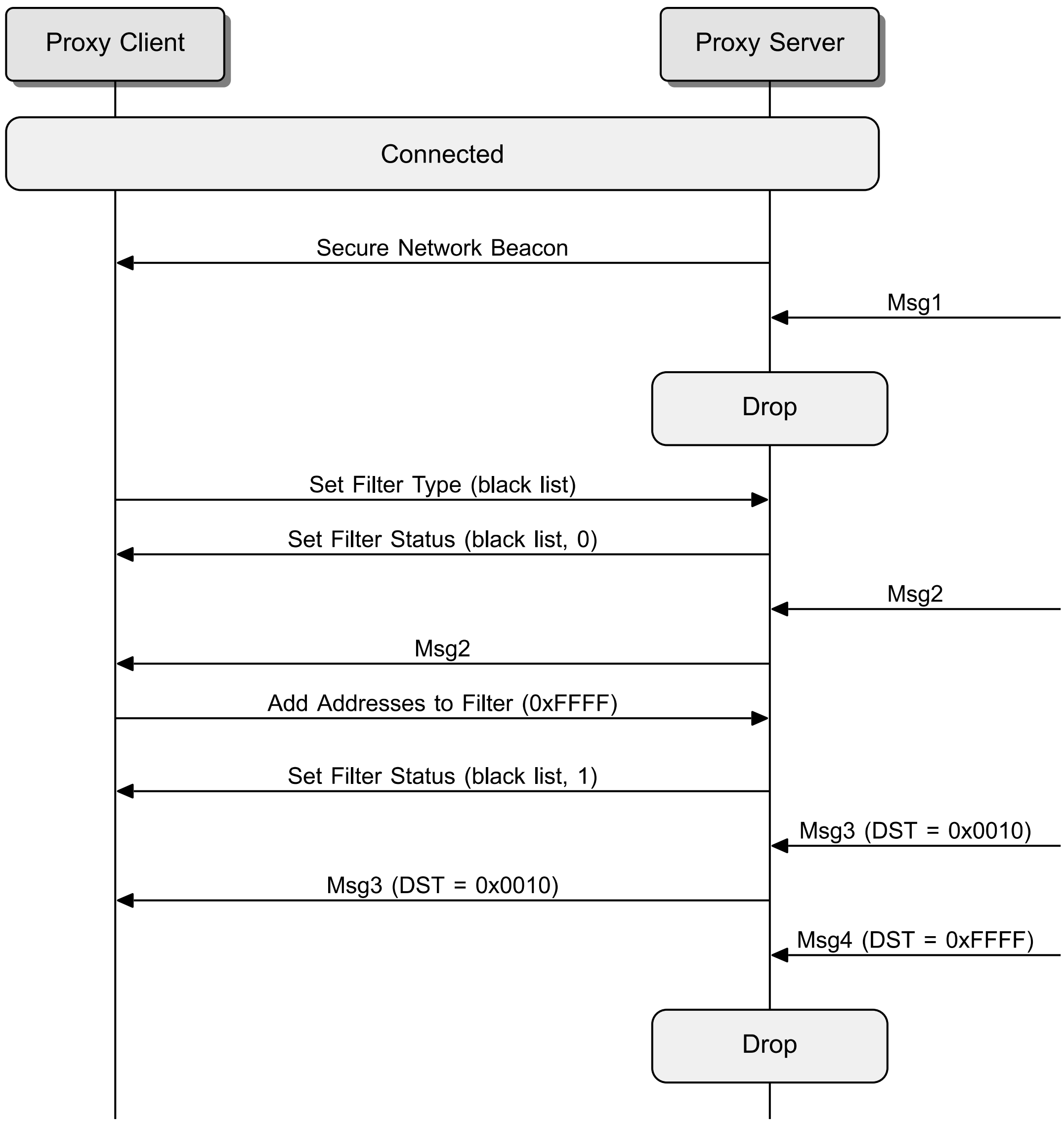


*Figure 6.3: Black list filtering*

# 7 Mesh GATT services

This section defines two GATT-based services: The Mesh Provisioning Service and the Mesh Proxy Service.

A device may support the Mesh Provisioning Service or the Mesh Proxy Service or both. If both are supported, only one of these services shall be exposed in the GATT database at a time.

## 7.1 Mesh Provisioning Service

### 7.1.1 Introduction

The Mesh Provisioning Service allows a Provisioning Client to provision a Provisioning Server to allow it to participate in the mesh network.

#### 7.1.1.1 Conformance

If a device claims conformance to this service, all capabilities indicated as mandatory for this service shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated.

#### 7.1.1.2 Service dependency

This service has no dependencies on other GATT-based services.

#### 7.1.1.3 Bluetooth specification release compatibility

This service is compatible with any Bluetooth Core Specification that includes the Generic Attribute Profile (GATT) and the Bluetooth Low Energy Controller portions of the Core Specification.

- Bluetooth Core Specification 4.0 or later [1].

#### 7.1.1.4 GATT sub-procedure requirements

Requirements in this section represent a minimum set of requirements for a server. Other GATT sub-procedures may be used if supported by both client and server.

Table 7.1 summarizes *additional* GATT sub-procedures required beyond those required by all GATT servers.

| GATT Sub-procedure | Requirements |
|---|---|

| GATT Sub-procedure | Requirements |
|---|---|
| Write Without Response | M |
| Notifications | M |
| Write Characteristic Descriptors | M |

*Table 7.1: Additional GATT sub-procedure requirements*

#### 7.1.1.5 Transport dependencies

The Mesh Provisioning Service operates over the low energy transport only. The specified functionality enables devices to participate in low energy mesh networks, and therefore support for the Bluetooth low energy transport is required.

#### 7.1.1.6 Application error codes

No application error codes are defined by this service.

### 7.1.2 Service requirements

#### 7.1.2.1 Declaration

The Mesh Provisioning Service shall be instantiated as a «Primary Service».

There shall only be one instance of the Mesh Provisioning Service on an unprovisioned Provisioning Server.

After the Provisioning Server has been provisioned, the Mesh Provisioning Service on that Provisioning Server shall cease to be present in the GATT database for that GATT Client as long as the node remains provisioned.

The Service UUID shall be set to «Mesh Provisioning Service», as defined in [4].

#### 7.1.2.2 Behaviors

The Mesh Provisioning Service may be used by a Provisioning Client to provision a Provisioning Server to participate in a mesh network.

##### *7.1.2.2.1 Advertising*

The Provisioning Server is only active and advertising when the device is not provisioned. The Provisioning Server shall be discoverable.

The advertisement shall follow the format as defined in Table 7.2.

| AD Type | Total Length | Requirement | Comments |
|---|---|---|---|

| AD Type | Total Length | Requirement | Comments |
|---|---|---|---|
| «Flags» | 3 | M | |
| «Incomplete List of 16-bit Service UUIDs» or<br>«Complete List of 16-bit Service UUIDs» | 4 | M | Include «Mesh Provisioning Service» |
| «Service Data» | 22 | M | Service Data AD Type for «Mesh Provisioning Service» followed by Service Data for this service |

*Table 7.2: Advertisement Data for the Mesh Provisioning Service*

The format of the Service Data for the «Mesh Provisioning Service» is defined in Table 7.3 and illustrated in Figure 7.1.

| Field | Size (octets) | Notes |
|---|---|---|
| Device UUID | 16 | See Section 3.10.3 |
| OOB Information | 2 | See Table 3.74 |

*Table 7.3: Service Data for Mesh Provisioning Service*

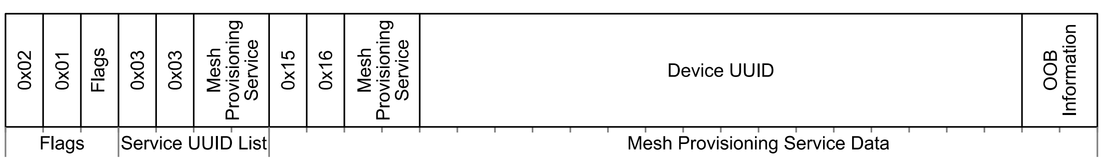


*Figure 7.1: PB-GATT Advertising Data*

If PB-ADV is supported, an unprovisioned Provisioning Server that supports PB-GATT has to interleave the connectable advertising for PB-GATT with the Unprovisioned Device beacon (see Section 3.9.2).

It is required to include the Service UUID List to allow discovery of the Mesh Provisioning service.

URI, Appearance, TX Power Level, and Local Name AD types can optionally be included in the scan response data. The Provisioning Server should support these data types to allow a Provisioning Client to enhance the user experience while provisioning.

### *7.1.2.2.2 ATT_MTU*

The Provisioning Server should support an ATT_MTU size equal to or larger than 69 octets to accommodate the longest known Provisioning message (see Section 5.3).

### 7.1.3 Mesh Provisioning Service characteristics

The characteristics shown in Table 7.4 are exposed in the Mesh Provisioning Service. Only one instance of each characteristic is permitted within this service. The characteristic formats and UUIDs are defined in [4].

Where a characteristic can be notified, a Client Characteristic Configuration descriptor shall be included in that characteristic as required by the Core Specification [1].

| Characteristic Name | Requirement | Mandatory Properties | Optional Properties | Security Permissions |
|---|---|---|---|---|
| Mesh Provisioning Data In | M | Write Without Response | None | Writeable with or without authentication or authorization when Unprovisioned |
| Mesh Provisioning Data Out | M | Notify | None | Notifiable with or without authentication or authorization when Unprovisioned |

*Table 7.4: Mesh Provisioning Service characteristics*

#### 7.1.3.1 Mesh Provisioning Data In characteristic

The Mesh Provisioning Data In characteristic can be written to send a Proxy PDU message (see Section 6.3.1) containing Provisioning PDU (see Section 5.3) to the Provisioning Server.

The Mesh Provisioning Data In characteristic is identified using the UUID «Mesh Provisioning Data In», as defined in [4].

The characteristic value is 66 octets long to accommodate the longest known Proxy PDU containing Provisioning PDU.

##### *7.1.3.1.1 Characteristic behavior*

The Mesh Provisioning Data In characteristic is used to transmit Proxy PDU message containing Provisioning PDU from a Provisioning Client to a Provisioning Server.

A Provisioning Server is not required to support bonding.

The Mesh Provisioning Data In characteristic shall support Proxy PDU messages containing Provisioning PDUs and shall not support other Proxy PDU type messages.

#### 7.1.3.2 Mesh Provisioning Data Out characteristic

The Mesh Provisioning Data Out characteristic can be notified to send a Proxy PDU message containing Provisioning PDU from a Provisioning Server to a Provisioning Client.

The Mesh Provisioning Data Out characteristic is identified using the UUID «Mesh Provisioning Data Out», as defined in [4].

The characteristic value is 66 octets long to accommodate the longest known Proxy PDU message containing Provisioning PDU.

##### *7.1.3.2.1 Characteristic behavior*

The Mesh Provisioning Data Out characteristic is used to Proxy PDU message containing Provisioning PDU from a Provisioning Server to a Provisioning Client.

A Provisioning Server is not required to support bonding. As a bonding relationship is not established between the Provisioning Client and the Provisioning Server, the Provisioning Client shall enable notifications (write value 0x0001) to the Mesh Provisioning Data Out Client Characteristic Configuration Descriptor after a connection is established and before sending the first message of the Proxy PDU.

The Mesh Provisioning Data Out characteristic shall support Proxy PDU messages containing Provisioning PDUs and shall not support other Proxy PDU type messages.

## 7.2 Mesh Proxy Service

### 7.2.1 Introduction

The Mesh Proxy Service is used to enable a server to send and receive Proxy PDUs with a client.

#### 7.2.1.1 Conformance

If a device claims conformance to this service, all capabilities indicated as mandatory for this service shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated.

#### 7.2.1.2 Service dependency

This service has no dependencies on other GATT-based services.

#### 7.2.1.3 Bluetooth specification release compatibility

This service is compatible with any Bluetooth Core Specification that includes the Generic Attribute Profile (GATT) and the Bluetooth Low Energy Controller portions of the Core Specification.

- Bluetooth Core Specification 4.0 or later [1].

#### 7.2.1.4 GATT sub-procedure requirements

Requirements in this section represent a minimum set of requirements for a server. Other GATT sub-procedures may be used if supported by both client and server.

Table 7.5 summarizes *additional* GATT sub-procedures required beyond those required by all GATT servers.

| GATT Sub-procedure | Requirements |
|---|---|
| Write Without Response | M |
| Notifications | M |
| Write Characteristic Descriptors | M |

*Table 7.5: Additional GATT sub-procedure requirements*

#### 7.2.1.5 Transport dependencies

The Mesh Proxy Service operates over the Bluetooth low energy transport only. The specified functionality enables devices to participate in low energy mesh networks, and therefore support for the Bluetooth low energy transport is required.

#### 7.2.1.6 Application error codes

No application error codes are defined by this service.

### 7.2.2 Service requirements

#### 7.2.2.1 Declaration

The Mesh Proxy Service shall be instantiated as a «Primary Service».

A device shall only have one instance of the Mesh Proxy Service.

The Service UUID shall be set to «Mesh Proxy Service», as defined in [4].

#### 7.2.2.2 Behaviors

The GATT Proxy state (see Section 4.2.11) indicates if the Mesh Proxy Service is supported, and if supported, it indicates and controls the status of the Mesh Proxy Service.

If the GATT Proxy state is 0x00 or 0x01, the Mesh Proxy Service shall be present in the GATT database of a provisioned device and the rules for advertising and the GATT characteristics behavior apply as discussed in the following sections.

If the GATT Proxy state is 0x02 the Mesh Proxy Service shall not be present in the GATT database and the UUID for the "Mesh Proxy Service" shall not be indicated in the advertisements.

###### 7.2.2.2.1 Advertising

The server shall use the GAP General Discoverable mode with connectable undirected advertising events in the format described in Table 7.6 below.

| AD Type | Total Length | Requirement | Comments |
|---|---|---|---|
| «Flags» | 3 | M | |
| «Incomplete List of 16-bit Service UUIDs» or<br>«Complete List of 16-bit Service UUIDs» | 4 | M | Include «Mesh Proxy Service» |
| «Service Data» | 21 | M | Service Data AD Type for «Mesh Proxy Service» followed by Service Data for this service |

*Table 7.6: Advertisement Data for the Mesh Proxy Service*

The format of Service Data for the «Mesh Proxy Service» is defined in Table 7.7.

| Field | Size (octets) | Notes |
|---|---|---|
| Identification Type | 1 | See Table 7.8 |
| Identification Parameters | Variable | Format determined by Identification Type field |

*Table 7.7: Service Data for Mesh Proxy Service*

The Identification Type field values are defined in Table 7.8.

| Type Value | Description |
|---|---|
| 0x00 | Network ID type |
| 0x01 | Node Identity type |
| 0x02–0xFF | Reserved for Future Use |

*Table 7.8: Identification Type values*

A node that supports the Proxy feature and has the Proxy feature enabled shall support advertising with Network ID.

A node that does not support the Proxy feature or has the Proxy feature disabled shall not advertise with Network ID.

A node may advertise with Node Identity regardless of the Proxy feature being supported or enabled.

The Identification Parameters for each Identification Type is defined in the subsections below.

The «Mesh Proxy Service» shall be included in the Incomplete List of 16-bit Service UUIDs or the Complete List of 16-bit Service UUIDs.

##### 7.2.2.2.2 Advertising with Network ID

The format of the Service Data for the «Mesh Proxy Service» when Advertising with the Network ID is defined in Table 7.9 and illustrated in Figure 7.2.

When a server is a member of multiple subnets, it shall interleave the advertising of each subnet.

| Field | Size (octets) | Notes |
|---|---|---|
| Identification Type | 1 | 0x00 (Network ID type) |
| Network ID | 8 | Identifies the network |

*Table 7.9: Service Data for Mesh Proxy Service with Network ID*

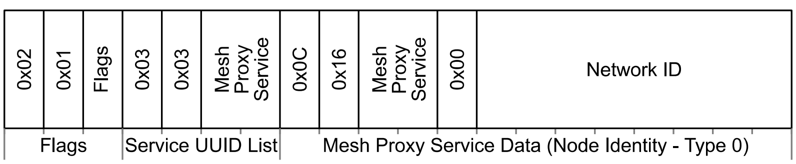


*Figure 7.2: Advertising with Network ID (Identification Type 0x00)*

The Network ID field shall be set as defined in Section 3.8.6.3.2.

##### 7.2.2.2.3 Advertising with Node Identity

Advertising using the Node Identity is used to identify a node based on the unicast address of the primary element and the network key of a subnet to which the node belongs. This can be useful when large amounts of data need to be delivered to a node via GATT for cases when the node cannot be easily identified or is not advertising. This advertisement may be used to initiate a GATT connection to the selected node.

If both PB-GATT and Mesh Proxy Service are supported, immediately after provisioning is completed using PB-GATT (see Section 5.2.2), the Mesh Proxy Service shall start advertising with Node Identity using the provisioned network.

When the server starts advertising as a result of user interaction, the server shall interleave the advertising of each subnet it is a member of. When the server starts advertising as a result of the Node Identity state being enabled, the server shall only advertise using the subnet that it was enabled on. The duration of advertising in these two cases is limited to 60 seconds.

The format of the Service Data for the «Mesh Proxy Service» when Advertising with Node Identity is defined in Table 7.10 and illustrated in Figure 7.3.

| Field | Size (octets) | Notes |
|---|---|---|
| Identification Type | 1 | 0x01 (Node Identity type) |
| Hash | 8 | Function of the included random number and identity information. |
| Random | 8 | 64-bit random number |

*Table 7.10: Service Data for Mesh Proxy Service with Node Identity*

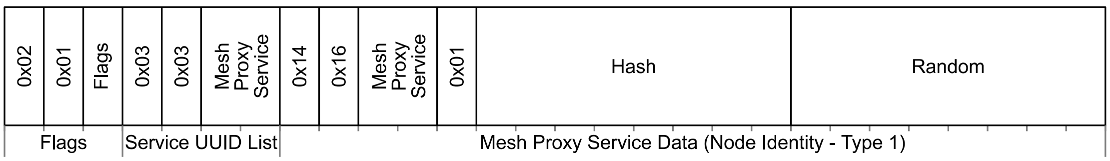


*Figure 7.3: Advertising with Node Identity (Identification Type 0x01)*

The Hash field is calculated as shown below:

Hash = e(IdentityKey, Padding | Random | Address) mod $2^{64}$

Where:

Padding – 48 bits of padding, all bits set to 0.

Random – 64 bit random value.

Address – The unicast address of the node.

The Random field is the 64 bit random value used in the Hash field calculation.

The creation of the encrypted node identity is illustrated in Figure 7.4.

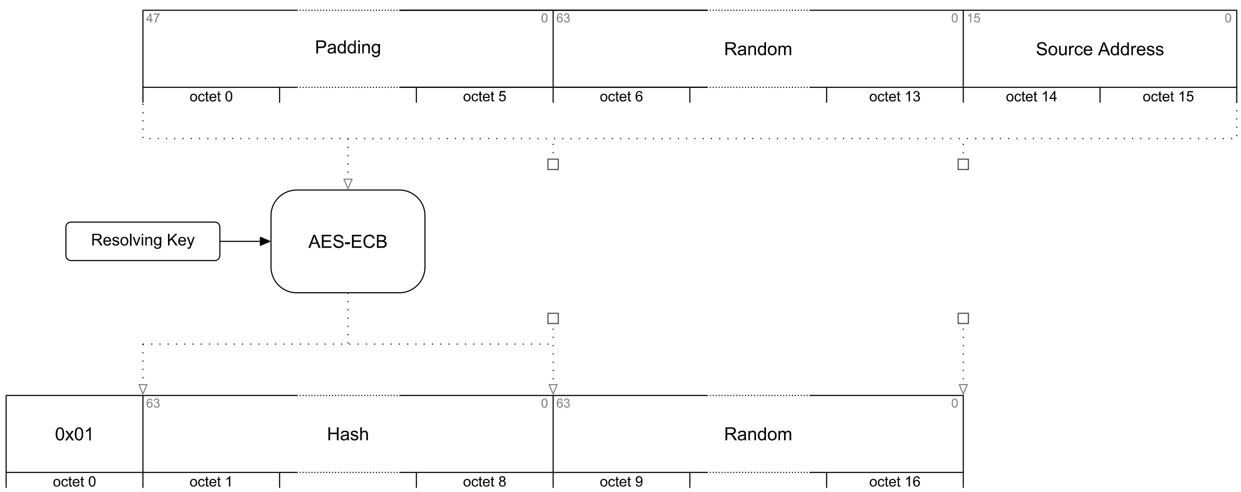


*Figure 7.4: Encrypted node identity creation*

#### *7.2.2.2.4 ATT_MTU*

The server should support an ATT_MTU size equal to or larger than 33 octets to be able to pass the content of a full Proxy PDU (see Section 6.5).

### 7.2.3 Mesh Proxy Service characteristics

The characteristics shown in Table 7.11 are exposed by the server. Only one instance of each characteristic is permitted within this service. The characteristic formats and UUIDs are defined in [4].

Where a characteristic can be notified, a Client Characteristic Configuration descriptor shall be included in that characteristic as required by the Core Specification [1].

| Characteristic Name | Requirement | Mandatory Properties | Optional Properties | Security Permissions |
|---|---|---|---|---|
| Mesh Proxy Data In | M | Write Without Response | None | Writeable with or without authentication or authorization when Proxy Service enabled |
| Mesh Proxy Data Out | M | Notify | None | Notifiable with or without authentication or authorization when Proxy Service enabled |

*Table 7.11: Mesh Proxy Service characteristics*

#### 7.2.3.1 Mesh Proxy Data In characteristic

The Mesh Proxy Data In characteristic is used by the client to send Proxy PDUs (see Section 6.3) to the server.

The Mesh Proxy Data In characteristic is identified using the UUID «Mesh Proxy Data In», as defined in [4]. The characteristic value has the same format as the Proxy PDU.

##### 7.2.3.1.1 Characteristic behavior

When the client sends a Proxy PDU to the server by executing a GATT Write Without Response procedure on this characteristic, the Attribute Value field of the ATT Write Command packet contains the Proxy PDU.

The Mesh Proxy Data In characteristic shall support Proxy PDU messages containing Network PDUs, mesh beacons, and proxy configuration messages and shall not support other Proxy PDU type messages.

#### 7.2.3.2 Mesh Proxy Data Out characteristic

The Mesh Proxy Data Out characteristic is used by the server to send Proxy PDUs to the client.

The Mesh Proxy Data Out characteristic is identified using the UUID «Mesh Proxy Data Out», as defined in [4]. The characteristic value has the same format as the Proxy PDU.

The Mesh Proxy Data Out characteristic shall support Proxy PDU messages containing Network PDUs, mesh beacons, and proxy configuration messages and shall not support other Proxy PDU type messages.

##### 7.2.3.2.1 Characteristic behavior

As a bonding relationship is not required between the client and the server, the client will enable notifications (write value 0x0001) to the Mesh Proxy Data Out Client Characteristic Configuration Descriptor after a connection is established.

The server can send a Proxy PDU to the client by executing a GATT Notification procedure on this characteristic. The Attribute Value field of the ATT Handle Value Notification packet contains the Proxy PDU.

The Mesh Proxy Data Out characteristic shall support Proxy PDU message containing Network PDUs, mesh beacons, and proxy configuration messages and shall not support other Proxy PDU type messages.

# 8 Sample data

This section provides informative examples of sample data that can be used to verify implementations.

## 8.1 Security sample data

In this section, all derivations are from the following keys:

```
AppKey              : 3216d1509884b533248541792b877f98
NetKey              : f7a2a44f8e8a8029064f173ddc1e2b00
DevKey (1201)       : 37c612c4a2d337cb7b98355531b3617f
```

### 8.1.1 s1 SALT generation function

The s1 function is used to generate a 128-bit value, typically known as a “salt” from a sequence of octets. In this case, the sequence of octets is the ASCII string ”test”.

```
s1 (test)           : b73cefbd641ef2ea598c2b6efb62f79c
```

### 8.1.2 k1 function

The k1 function is used to convert some input key material into some output key material that uses two inputs, known as salt and info.

```
k1 N                : 3216d1509884b533248541792b877f98
k1 SALT             : 2ba14ffa0df84a2831938d57d276cab4
k1 P                : 5a09d60797eeb4478aada59db3352a0d
k1 T                : c764bea25cf9738b08956ea3c712d5af
k1                  : f6ed15a8934afbe7d83e8dcb57fcf5d7
```

### 8.1.3 k2 function (master)

The k2 function is used to convert the master security credentials, NetKey as N, and generate the master security material, NID, EncryptionKey, and PrivacyKey.

```
k2 N                : f7a2a44f8e8a8029064f173ddc1e2b00
k2 P                : 00
k2 s1(smk2)         : 4f90480c1871bfbffd16971f4d8d10b1
k2 T                : 2ea6467aa3378c4c545eda62935b9b86
k2 T0               :
k2 T1               : 82bea685dc2f1deec255ac643741f5ff
k2 T2               : 9f589181a0f50de73c8070c7a6d27f46
k2 T3               : 4c715bd4a64b938f99b453351653124f
NID                 : 7f
EncryptionKey       : 9f589181a0f50de73c8070c7a6d27f46
PrivacyKey          : 4c715bd4a64b938f99b453351653124f
```

### 8.1.4 k2 function (friendship)

The k2 function is also used to convert the friendship security credentials, NetKey as N, LPNAddress, FriendAddress, LPNCounter, and FriendCounter, and generate the friendship security material NID, EncryptionKey, and PrivacyKey.

```
LPNAddress           : 0203
FriendAddress        : 0405
LPNCounter           : 0607
FriendCounter        : 0809
k2 N                 : f7a2a44f8e8a8029064f173ddc1e2b00
k2 P                 : 010203040506070809
k2 s1(smk2)          : 4f90480c1871bfbffd16971f4d8d10b1
k2 T                 : 2ea6467aa3378c4c545eda62935b9b86
k2 T0                :
k2 T1                : 3a6b950f56718c1eab2c600a92d4e9f3
k2 T2                : 11efec0642774992510fb5929646df49
k2 T3                : d4d7cc0dfa772d836a8df9df5510d7a7
NID                  : 73
EncryptionKey        : 11efec0642774992510fb5929646df49
PrivacyKey           : d4d7cc0dfa772d836a8df9df5510d7a7
```

### 8.1.5 k3 function

The k3 function is used to generate a 64-bit Network ID used to identify the mesh network in public messages such as a Secure Network beacon.

```
k3 N                 : f7a2a44f8e8a8029064f173ddc1e2b00
k3 SALT              : 0036443503f195cc8a716e136291c302
k3 T                 : 6da9698c95f500e4edce3bb47f92754f
k3 CMAC(id64|0x01)   : 3527c5985f0c05ccff046958233db014
Network ID           : ff046958233db014
```

### 8.1.6 k4 function

The k4 function is used to generate an AID from an application key.

```
k4 N                 : 3216d1509884b533248541792b877f98
k4 SALT              : 0e9ac1b7cefa66874c97ee54ac5f49be
k4 T                 : 62e5d9240cdb3bb0b2743207ea2d6276
k4 CMAC(id6|0x01)    : 1431ea1afeb05224ab892a0217ccab38
AID                  : 38
```

## 8.2 Mesh key derivation sample data

In this section, all derivations are from the following keys. These are the same keys used in the mesh message sample data.

```
AppKey              : 63964771734fbd76e3b40519d1d94a48
NetKey              : 7dd7364cd842ad18c17c2b820c84c3d6
DevKey (1201)       : 9d6dd0e96eb25dc19a40ed9914f8f03f
```

### 8.2.1 Application key AID

The application keys AID is calculated using the k4 function.

```
k4 N                : 63964771734fbd76e3b40519d1d94a48
k4 SALT             : 0e9ac1b7cefa66874c97ee54ac5f49be
k4 T                : 921cb4f908cc5932e1d7b059fc163ce6
k4 CMAC(id6|0x01)   : 5f79cf09bbdab560e7f1ee404fd341a6
AID                 : 26
```

### 8.2.2 Encryption and privacy keys (Master)

The NID, EncryptionKey, and PrivacyKey for a normal mesh message sent using the master security credentials.

```
k2 N                : 7dd7364cd842ad18c17c2b820c84c3d6
k2 P                : 00
k2 s1(smk2)         : 4f90480c1871bfbffd16971f4d8d10b1
k2 T                : 39885e0463bafd54ca6e495b1001515a
k2 T0               :
k2 T1               : 88dad4892e81fecbe061ebd3fb093268
k2 T2               : 0953fa93e7caac9638f58820220a398e
k2 T3               : 8b84eedec100067d670971dd2aa700cf
NID                 : 68
EncryptionKey       : 0953fa93e7caac9638f58820220a398e
PrivacyKey          : 8b84eedec100067d670971dd2aa700cf
```

### 8.2.3 Encryption and privacy keys (Friendship)

The NID, EncryptionKey, and PrivacyKey for a mesh message between a Friend node and a Low Power node sent using the friendship security credentials.

```
LPNAddress          : 1201
FriendAddress       : 2345
LPNCounter          : 0000
FriendCounter       : 072f
k2 N                : 7dd7364cd842ad18c17c2b820c84c3d6
k2 P                : 01120123450000072f
```

```
k2 s1(smk2)           : 4f90480c1871bfbffd16971f4d8d10b1
k2 T                  : 39885e0463bafd54ca6e495b1001515a
k2 T0                 :
k2 T1                 : d91a3b3c63b5c50a98c838e52a4bc0de
k2 T2                 : be635105434859f484fc798e043ce40e
k2 T3                 : 5d396d4b54d3cbafe943e051fe9a4eb8
NID                   : 5e
EncryptionKey         : be635105434859f484fc798e043ce40e
PrivacyKey            : 5d396d4b54d3cbafe943e051fe9a4eb8
```

### 8.2.4 Network ID

The Network ID used to identify this NetKey, for example, in a Secure Network beacon.

```
k3 N                  : 7dd7364cd842ad18c17c2b820c84c3d6
k3 SALT               : 0036443503f195cc8a716e136291c302
k3 T                  : 36b82fd0fc400e797977bd12d08a4782
k3 CMAC(id64|0x01)    : ca296bcee3ccc2d33ecaff672f673370
Network ID            : 3ecaff672f673370
```

### 8.2.5 IdentityKey

The Identify key used to help identify the device in the Mesh Proxy Service's Service Data using Node Identity.

```
k1 N                  : 7dd7364cd842ad18c17c2b820c84c3d6
k1 SALT               : f8795a1aabf182e4f163d86e245e19f4
k1 P                  : 696431323801
k1 T                  : 55efb6c898c2a38bc9bd0a6097bff966
IdentityKey           : 84396c435ac48560b5965385253e210c
```

### 8.2.6 BeaconKey

The Beacon key is used to help secure the Secure Network beacon.

```
k1 N                  : 7dd7364cd842ad18c17c2b820c84c3d6
k1 SALT               : 2c24619ab793c1233f6e226738393dec
k1 P                  : 696431323801
k1 T                  : 829816cd429fde7d238b56d8bf771efb
BeaconKey             : 5423d967da639a99cb02231a83f7d254
```

## 8.3 Mesh message sample data

In this section, all messages are secured using the following keys:

```
AppKey                : 63964771734fbd76e3b40519d1d94a48
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
```

```
DevKey (1201)          : 9d6dd0e96eb25dc19a40ed9914f8f03f
```

The messages below show a typical set of transactions used by a Low Power node that are intended to show multiple examples of mesh messages.

### 8.3.1 Message #1

This shows an example of a new node that supports the Low Power feature attempting to find a Friend node. It does this by sending a Friend Request with its requirements to the all-friends address using a TTL of zero.

```
Transport Control Message

Opcode                 : 03 (Friend Request)
  Criteria             : 4b
  ReceiveDelay         : 50
  PollTimeout          : 057e40
  PreviousAddress      : 0000
  NumElements          : 01
  LPNCounter           : 0000

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 000001
SRC                    : 1201
DST                    : fffd
UpperTransportPDU      : 4b50057e400000010000

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 000001
SRC                    : 1201
DST                    : fffd
SEG                    : 00
Opcode                 : 03
Header                 : 03
UpperTransportPDU      :   4b50057e400000010000
LowerTransportPDU      : 034b50057e400000010000

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
```

```
CTL                   : 01
TTL                   : 00
SEQ                   : 000001
SRC                   : 1201
DST                   : fffd
LowerTransportPDU     : 034b50057e400000010000
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 00800000011201000012345678
IVI NID               : 68
CTL TTL               :   80
SEQ                   :     000001
SRC                   :           1201
DST                   :               fffd
TransportPDU          :                   034b50057e400000010000
NetMIC Size           : 64 bits
EncTransportPDU       :               b5e5bfdacbaf6cb7fb6bff871f
NetMIC                :                                         035444ce83a670df

Obfuscation

Privacy Random        : 000000000012345678b5e5bfdacbaf6c
PECB                  : 6ca487507564
preObfuscation        : 800000011201

NetworkPDU            : 68eca487516765b5e5bfdacbaf6cb7fb6bff871f035444ce83a670df
```

### 8.3.2 Message #2

A node that supports the Friend feature responds to the Friend Request with a Friend Offer.

```
Transport Control Message

Opcode                : 04 (Friend Offer)
  ReceiveWindow       : 32
  QueueSize           : 03
  SubListSize         : 08
  RSSI                : ba (-70)
  FriendCounter       : 072f

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 014820
```

```
SRC                    : 2345
DST                    : 1201
UpperTransportPDU      : 320308ba072f

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 014820
SRC                    : 2345
DST                    : 1201
SEG                    : 00
Opcode                 : 04
Header                 : 04
UpperTransportPDU      :   320308ba072f
LowerTransportPDU      : 04320308ba072f

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 00
SEQ                    : 014820
SRC                    : 2345
DST                    : 1201
LowerTransportPDU      : 04320308ba072f
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 00800148202345000012345678
IVI NID                : 68
CTL TTL                :   80
SEQ                    :     014820
SRC                    :           2345
DST                    :               1201
TransportPDU           :                   04320308ba072f
NetMIC Size            : 64 bits
EncTransportPDU        :               79d7dbc0c9b4d43eeb
NetMIC                 :                                 ec129d20a620d01e

Obfuscation

Privacy Random         : 0000000001234567879d7dbc0c9b4d4
PECB                   : 54c96e094e3c
```

```
preObfuscation          : 800148202345

NetworkPDU              : 68d4c826296d7979d7dbc0c9b4d43eebec129d20a620d01e
```

### 8.3.3 Message #3

Another node supporting the Friend feature responds with another Friend Offer. This Friend Offer has a smaller queue size and a significantly worse RSSI figure than the other Friend Offer message.

```
Transport Control Message

Opcode                  : 04 (Friend Offer)
  ReceiveWindow         : fa
  QueueSize             : 02
  SubListSize           : 05
  RSSI                  : a6 (-90)
  FriendCounter         : 000a

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 2b3832
SRC                     : 2fe3
DST                     : 1201
UpperTransportPDU       : fa0205a6000a

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 2b3832
SRC                     : 2fe3
DST                     : 1201
SEG                     : 00
Opcode                  : 04
Header                  : 04
UpperTransportPDU       :   fa0205a6000a
LowerTransportPDU       : 04fa0205a6000a

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
```

```
SEQ                     : 2b3832
SRC                     : 2fe3
DST                     : 1201
LowerTransportPDU       : 04fa0205a6000a
NID                     : 68
EncryptionKey           : 0953fa93e7caac9638f58820220a398e
PrivacyKey              : 8b84eedec100067d670971dd2aa700cf
Network Nonce           : 00802b38322fe3000012345678
IVI NID                 : 68
CTL TTL                 :   80
SEQ                     :     2b3832
SRC                     :           2fe3
DST                     :               1201
TransportPDU            :                   04fa0205a6000a
NetMIC Size             : 64 bits
EncTransportPDU         :               53273086b8c5ee00bd
NetMIC                  :                                 d9cfcc62a2ddf572

Obfuscation

Privacy Random          : 0000000000123456785327308 6b8c5ee
PECB                    : 5a2d13fb4211
preObfuscation          : 802b38322fe3

NetworkPDU              : 68da062bc96df253273086b8c5ee00bdd9cfcc62a2ddf572
```

### 8.3.4 Message #4

The node sends a Friend Poll to confirm the friendship with one of the Friend nodes.

```
Transport Control Message

Opcode                  : 01 (Friend Poll)
  FSN                   : 0

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 000002
SRC                     : 1201
DST                     : 2345
UpperTransportPDU       : 00

LowerTransportUnsegmentedControlPDU
```

```
CTL                    : 01
TTL                    : 00
SEQ                    : 000002
SRC                    : 1201
DST                    : 2345
SEG                    : 00
Opcode                 : 01
Header                 : 01
UpperTransportPDU      :   00
LowerTransportPDU      : 0100

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
LPNAddress             : 1201
FriendAddress          : 2345
LPNCounter             : 0000
FriendCounter          : 072f
CTL                    : 01
TTL                    : 00
SEQ                    : 000002
SRC                    : 1201
DST                    : 2345
LowerTransportPDU      : 0100
NID                    : 5e
EncryptionKey          : be635105434859f484fc798e043ce40e
PrivacyKey             : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce          : 00800000021201000012345678
IVI NID                : 5e
CTL TTL                :   80
SEQ                    :     000002
SRC                    :           1201
DST                    :               2345
TransportPDU           :                   0100
NetMIC Size            : 64 bits
EncTransportPDU        :               b0e5d0ad
NetMIC                 :                       970d579a4e88051c

Obfuscation

Privacy Random         : 0000000000012345678b0e5d0ad970d57
PECB                   : 04eba0902a0e
preObfuscation         : 800000021201
```

```
NetworkPDU             : 5e84eba092380fb0e5d0ad970d579a4e88051c
```

### 8.3.5 Message #5

The Friend node confirms this friendship with a Friend Update message. The two nodes are now friends.

```
Transport Control Message

Opcode                 : 02 (Friend Update)
  Flags                : 00
  IV Index             : 12345678
  MD                   : 00

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 014834
SRC                    : 2345
DST                    : 1201
UpperTransportPDU      : 001234567800

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 014834
SRC                    : 2345
DST                    : 1201
SEG                    : 00
Opcode                 : 02
Header                 : 02
UpperTransportPDU      :   001234567800
LowerTransportPDU      : 02001234567800

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
LPNAddress             : 1201
FriendAddress          : 2345
LPNCounter             : 0000
FriendCounter          : 072f
CTL                    : 01
TTL                    : 00
SEQ                    : 014834
```

```
SRC                    : 2345
DST                    : 1201
LowerTransportPDU      : 02001234567800
NID                    : 5e
EncryptionKey          : be635105434859f484fc798e043ce40e
PrivacyKey             : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce          : 00800148342345000012345678
IVI NID                : 5e
CTL TTL                :   80
SEQ                    :     014834
SRC                    :           2345
DST                    :               1201
TransportPDU           :                   02001234567800
NetMIC Size            : 64 bits
EncTransportPDU        :               5c39da1792b1fee9ec
NetMIC                 :                                 74b786c56d3a9dee

Obfuscation

Privacy Random         : 0000000000123456785c39da1792b1fe
PECB                   : 2fd7bd08609e
preObfuscation         : 800148342345

NetworkPDU             : 5eafd6f53c43db5c39da1792b1fee9ec74b786c56d3a9dee
```

### 8.3.6 Message #6

A Configuration Client sends a Config AppKey Add message to the Low Power node. This message is sent in two segments.

```
Access Payload

Opcode                 : 00 (Config AppKey Add)
NetKeyIndex            : 123
AppKeyIndex            : 456
AppKey                 : 63964771734fbd76e3b40519d1d94a48
Access Payload         : 0056341263964771734fbd76e3b40519d1d94a48

UpperTransportAccessPDU

IV Index               : 12345678
DevKey                 : 9d6dd0e96eb25dc19a40ed9914f8f03f
TTL                    : 04
SEQ                    : 3129ab
SRC                    : 0003
```

```
DST                     : 1201
Application Nonce       : 02003129ab0003120112345678
EncAccessPayload        : ee9dddfd2169326d23f3afdfcfdc18c52fdef772
TransMIC Size           : 32 bits
TransMIC                : e0e17308
UpperTransportPDU       : ee9dddfd2169326d23f3afdfcfdc18c52fdef772e0e17308
Segment#0               : ee9dddfd2169326d23f3afdf
Segment#1               : cfdc18c52fdef772e0e17308

LowerTransportSegmentedAccessPDU

CTL                     : 00
TTL                     : 04
SEQ                     : 3129ab
SRC                     : 0003
DST                     : 1201
SEG                     : 01
AKF                     : 00
AID                     : 00
SZMIC                   : 00
SeqZero                 : 9ab
SegO                    : 00
SegN                    : 01
Header                  : 8026ac01
Segment#0               :         ee9dddfd2169326d23f3afdf
LowerTransportPDU       : 8026ac01ee9dddfd2169326d23f3afdf

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 00
TTL                     : 04
SEQ                     : 3129ab
SRC                     : 0003
DST                     : 1201
LowerTransportPDU       : 8026ac01ee9dddfd2169326d23f3afdf
NID                     : 68
EncryptionKey           : 0953fa93e7caac9638f58820220a398e
PrivacyKey              : 8b84eedec100067d670971dd2aa700cf
Network Nonce           : 00043129ab0003000012345678
IVI NID                 : 68
CTL TTL                 :   04
SEQ                     :     3129ab
SRC                     :           0003
```

```
DST                      :               1201
TransportPDU             :                   8026ac01ee9dddfd2169326d23f3afdf
NetMIC Size              : 32 bits
EncTransportPDU          :               0afba8c63d4e686364979deaf4fd40961145
NetMIC                   :                                                    939cda0e

Obfuscation

Privacy Random           : 0000000000123456780afba8c63d4e68
PECB                     : ce84ec9f8a20
preObfuscation           : 043129ab0003

NetworkPDU               : 68cab5c5348a230afba8c63d4e686364979deaf4fd40961145939cda0e

LowerTransportSegmentedAccessPDU

CTL                      : 00
TTL                      : 04
SEQ                      : 3129ac
SRC                      : 0003
DST                      : 1201
SEG                      : 01
AKF                      : 00
AID                      : 00
SZMIC                    : 00
SeqZero                  : 3129ab
SegO                     : 01
SegN                     : 01
Header                   : 8026ac21
Segment#1                :         cfdc18c52fdef772e0e17308
LowerTransportPDU        : 8026ac21cfdc18c52fdef772e0e17308

NetworkPDU

IVindex                  : 12345678
NetKey                   : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                      : 00
TTL                      : 04
SEQ                      : 3129ac
SRC                      : 0003
DST                      : 1201
LowerTransportPDU        : 8026ac21cfdc18c52fdef772e0e17308
NID                      : 68
EncryptionKey            : 0953fa93e7caac9638f58820220a398e
PrivacyKey               : 8b84eedec100067d670971dd2aa700cf
```

```
Network Nonce          : 00043129ac0003000012345678
IVI NID                : 68
CTL TTL                :   04
SEQ                    :     3129ac
SRC                    :           0003
DST                    :               1201
TransportPDU           :                   8026ac21cfdc18c52fdef772e0e17308
NetMIC Size            : 32 bits
EncTransportPDU        :               6cae0c032bf0746f44f1b8cc8ce5edc57e55
NetMIC                 :                                                   beed49c0

Obfuscation

Privacy Random         : 0000000000123456786cae0c032bf074
PECB                   : 12249c714a87
preObfuscation         : 043129ac0003

NetworkPDU             : 681615b5dd4a846cae0c032bf0746f44f1b8cc8ce5edc57e55beed49c0
```

### 8.3.7 Message #7

A friend of the destination acknowledges only one of the segments.

```
Transport Control Message

Opcode                 : 00 (Segment Acknowledgment)
  OBO                  : 01
  SeqZero              : 09ab
  BlockAck             : 00000002

UpperTransportControlPDU

TTL                    : 0b
SEQ                    : 014835
SRC                    : 2345
DST                    : 0003
UpperTransportPDU      : a6ac00000002

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 0b
SEQ                    : 014835
SRC                    : 2345
DST                    : 0003
```

```
SEG                   : 00
Opcode                : 00
Header                : 00
UpperTransportPDU     :   a6ac00000002
LowerTransportPDU     : 00a6ac00000002

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 0b
SEQ                   : 014835
SRC                   : 2345
DST                   : 0003
LowerTransportPDU     : 00a6ac00000002
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 008b0148352345000012345678
IVI NID               : 68
CTL TTL               :   8b
SEQ                   :     014835
SRC                   :           2345
DST                   :               0003
TransportPDU          :                   00a6ac00000002
NetMIC Size           : 64 bits
EncTransportPDU       :               0d0d730f94d7f3509d
NetMIC                :                                 f987bb417eb7c05f

Obfuscation

Privacy Random        : 0000000000123456780d0d730f94d7f3
PECB                  : 6f77fd62bfdd
preObfuscation        : 8b0148352345

NetworkPDU            : 68e476b5579c980d0d730f94d7f3509df987bb417eb7c05f
```

### 8.3.8 Message #8

The Configuration Client receives the segment acknowledgment and retransmits the unacknowledged segment.

```
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
```

```
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 00043129ad0003000012345678
IVI NID                : 68
CTL TTL                :   04
SEQ                    :     3129ad
SRC                    :           0003
DST                    :               1201
TransportPDU           :                   8026ac01ee9dddfd2169326d23f3afdf
NetMIC Size            : 32 bits
EncTransportPDU        :               0e2f91add6f06e66006844cec97f973105ae
NetMIC                 :                                                   2534f958

Obfuscation

Privacy Random         : 0000000000123456780e2f91add6f06e
PECB                   : 499b4bcac2cc
preObfuscation         : 043129ad0003
NetworkPDU             : 684daa6267c2cf0e2f91add6f06e66006844cec97f973105ae2534f958
```

### 8.3.9 Message #9

The Friend node receives this last segment and sends an acknowledgment of this last segment.

```
Transport Control Message

Opcode                 : 00 (Segment Acknowledgment)
  OBO                  : 01
  SeqZero              : 09ab
  BlockAck             : 00000003

UpperTransportControlPDU

TTL                    : 0b
SEQ                    : 014836
SRC                    : 2345
DST                    : 0003
UpperTransportPDU      : a6ac00000003

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 0b
SEQ                    : 014836
SRC                    : 2345
DST                    : 0003
```

```
SEG                   : 00
Opcode                : 00
Header                : 00
UpperTransportPDU     :   a6ac00000003
LowerTransportPDU     : 00a6ac00000003

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 0b
SEQ                   : 014836
SRC                   : 2345
DST                   : 0003
LowerTransportPDU     : 00a6ac00000003
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 008b0148362345000012345678
IVI NID               : 68
CTL TTL               :   8b
SEQ                   :     014836
SRC                   :           2345
DST                   :               0003
TransportPDU          :                   00a6ac00000003
NetMIC Size           : 64 bits
EncTransportPDU       :               d85d806bbed248614f
NetMIC                :                                  938067b0d983bb7b

Obfuscation

Privacy Random        : 0000000000012345678d85d806bbed248
PECB                  : 25c52fdb6a44
preObfuscation        : 8b0148362345

NetworkPDU            : 68aec467ed4901d85d806bbed248614f938067b0d983bb7b
```

### 8.3.10 Message #10

Sometime later, the Low Power node polls its Friend node to check for more stored messages.

```
Transport Control Message

Opcode                : 01 (Friend Poll)
```

```
  FSN                   : 1

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 000003
SRC                     : 1201
DST                     : 2345
UpperTransportPDU       : 01

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 000003
SRC                     : 1201
DST                     : 2345
SEG                     : 00
Opcode                  : 01
Header                  : 01
UpperTransportPDU       :   01
LowerTransportPDU       : 0101

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
LPNAddress              : 1201
FriendAddress           : 2345
LPNCounter              : 0000
FriendCounter           : 072f
CTL                     : 01
TTL                     : 00
SEQ                     : 000003
SRC                     : 1201
DST                     : 2345
LowerTransportPDU       : 0101
NID                     : 5e
EncryptionKey           : be635105434859f484fc798e043ce40e
PrivacyKey              : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce           : 00800000031201000012345678
IVI NID                 : 5e
CTL TTL                 :   80
SEQ                     :     000003
SRC                     :           1201
```

```
DST                   :              2345
TransportPDU          :                  0101
NetMIC Size           : 64 bits
EncTransportPDU       :              7777ed35
NetMIC                :                      5afaf66d899c1e3d

Obfuscation

Privacy Random        : 0000000000123456787777ed355afaf6
PECB                  : fb78656b679e
preObfuscation        : 800000031201

NetworkPDU            : 5e7b786568759f7777ed355afaf66d899c1e3d
```

### 8.3.11 Message #11

The Friend node responds to this poll with the first segment of the stored message. It also indicates that it has more data.

```
NID                   : 5e
EncryptionKey         : be635105434859f484fc798e043ce40e
PrivacyKey            : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce         : 00033129ad0003000012345678
IVI NID               : 5e
CTL TTL               :   03
SEQ                   :     3129ad
SRC                   :           0003
DST                   :              1201
TransportPDU          :                  c026ac01ee9dddfd2169326d23f3afdf
NetMIC Size           : 32 bits
EncTransportPDU       :              d5e748a20ecfd98ddfd32de80befb400213d
NetMIC                :                                          113813b5

Obfuscation

Privacy Random        : 0000000000012345678d5e748a20ecfd9
PECB                  : 6d8ee98cedf6
preObfuscation        : 033129ad0003
NetworkPDU            : 5e6ebfc021edf5d5e748a20ecfd98ddfd32de80befb400213d113813b5
```

### 8.3.12 Message #12

The Low Power node didn't receive that message, so polls again for the same message with the FSN value having the same value as last time.

```
Transport Control Message
```

```
Opcode                  : 01 (Friend Poll)
  FSN                   : 1

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 000004
SRC                     : 1201
DST                     : 2345
UpperTransportPDU       : 01

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 000004
SRC                     : 1201
DST                     : 2345
SEG                     : 00
Opcode                  : 01
Header                  : 01
UpperTransportPDU       :   01
LowerTransportPDU       : 0101

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
LPNAddress              : 1201
FriendAddress           : 2345
LPNCounter              : 0000
FriendCounter           : 072f
CTL                     : 01
TTL                     : 00
SEQ                     : 000004
SRC                     : 1201
DST                     : 2345
LowerTransportPDU       : 0101
NID                     : 5e
EncryptionKey           : be635105434859f484fc798e043ce40e
PrivacyKey              : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce           : 00800000041201000012345678
IVI NID                 : 5e
CTL TTL                 :   80
```

```
SEQ                    :     000004
SRC                    :           1201
DST                    :               2345
TransportPDU           :                   0101
NetMIC Size            : 64 bits
EncTransportPDU        :               ae214660
NetMIC                 :                       87599c2426ce9a35

Obfuscation

Privacy Random         : 000000000012345678ae21466087599c
PECB                   : 0a18fc6a5f04
preObfuscation         : 800000041201

NetworkPDU             : 5e8a18fc6e4d05ae21466087599c2426ce9a35
```

### 8.3.13 Message #13

The Friend node responds with the same message as last time.

```
NID                    : 5e
EncryptionKey          : be635105434859f484fc798e043ce40e
PrivacyKey             : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce          : 00033129ad0003000012345678
IVI NID                : 5e
CTL TTL                :   03
SEQ                    :     3129ad
SRC                    :           0003
DST                    :               1201
TransportPDU           :                   c026ac01ee9dddfd2169326d23f3afdf
NetMIC Size            : 32 bits
EncTransportPDU        :               d5e748a20ecfd98ddfd32de80befb400213d
NetMIC                 :                                                   113813b5

Obfuscation

Privacy Random         : 000000000012345678d5e748a20ecfd9
PECB                   : 6d8ee98cedf6
preObfuscation         : 033129ad0003

NetworkPDU             : 5e6ebfc021edf5d5e748a20ecfd98ddfd32de80befb400213d113813b5
```

### 8.3.14 Message #14

The Low Power node received the retransmitted stored message. As that message has the MD bit set, it sends another Friend Poll to obtain the next message.

```
Transport Control Message

Opcode                : 01 (Friend Poll)
  FSN                 : 0

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 000005
SRC                   : 1201
DST                   : 2345
UpperTransportPDU     : 00

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 000005
SRC                   : 1201
DST                   : 2345
SEG                   : 00
Opcode                : 01
Header                : 01
UpperTransportPDU     :   00
LowerTransportPDU     : 0100

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
LPNAddress            : 1201
FriendAddress         : 2345
LPNCounter            : 0000
FriendCounter         : 072f
CTL                   : 01
TTL                   : 00
SEQ                   : 000005
SRC                   : 1201
DST                   : 2345
LowerTransportPDU     : 0100
NID                   : 5e
EncryptionKey         : be635105434859f484fc798e043ce40e
PrivacyKey            : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce         : 00800000051201000012345678
IVI NID               : 5e
```

```
CTL TTL              :   80
SEQ                  :     000005
SRC                  :           1201
DST                  :               2345
TransportPDU         :                   0100
NetMIC Size          : 64 bits
EncTransportPDU      :               7d3ae62a
NetMIC               :                       3c75dff683dce24e

Obfuscation

Privacy Random       : 0000000000123456787d3ae62a3c75df
PECB                 : 8bbaf92e4e8e
preObfuscation       : 800000051201

NetworkPDU           : 5e0bbaf92b5c8f7d3ae62a3c75dff683dce24e
```

### 8.3.15 Message #15

The Friend node responds, with the next message in the friend queue. The Friend node has no more data, so it sets the MD to 0.

```
NID                  : 5e
EncryptionKey        : be635105434859f484fc798e043ce40e
PrivacyKey           : 5d396d4b54d3cbafe943e051fe9a4eb8
Network Nonce        : 00033129ac0003000012345678
IVI NID              : 5e
CTL TTL              :   03
SEQ                  :     3129ac
SRC                  :           0003
DST                  :               1201
TransportPDU         :                   8026ac21cfdc18c52fdef772e0e17308
NetMIC Size          : 32 bits
EncTransportPDU      :               f1d29805664d235eacd707217dedfe78497f
NetMIC               :                                                   efec7391

Obfuscation

Privacy Random       : 000000000012345678f1d29805664d23
PECB                 : abeb9ca27ee4
preObfuscation       : 033129ac0003
NetworkPDU           : 5ea8dab50e7ee7f1d29805664d235eacd707217dedfe78497fefec7391
```

### 8.3.16 Message #16

The Low Power node has now received the complete Config AppKey Add message, so it responds to the segmented message with a status message. This is sent directly to the Configuration Client.

```
Access Payload

Opcode                 : 8003 (Config AppKey Status)
Status                 : 00
NetKeyIndex            : 123
AppKeyIndex            : 456
Access Payload         : 800300563412

UpperTransportAccessPDU

IV Index               : 12345678
DevKey                 : 9d6dd0e96eb25dc19a40ed9914f8f03f
TTL                    : 0b
SEQ                    : 000006
SRC                    : 1201
DST                    : 0003
Application Nonce      : 02000000061201000312345678
EncAccessPayload       : 89511bf1d1a8
TransMIC Size          : 32 bits
TransMIC               : 1c11dcef
UpperTransportPDU      : 89511bf1d1a81c11dcef

LowerTransportUnsegmentedAccessPDU

CTL                    : 00
TTL                    : 0b
SEQ                    : 000006
SRC                    : 1201
DST                    : 0003
SEG                    : 00
AKF                    : 00
AID                    : 00
Header                 : 00
UpperTransportPDU      :   89511bf1d1a81c11dcef
LowerTransportPDU      : 0089511bf1d1a81c11dcef

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
```

```
CTL                    : 00
TTL                    : 0b
SEQ                    : 000006
SRC                    : 1201
DST                    : 0003
LowerTransportPDU      : 0089511bf1d1a81c11dcef
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 000b0000061201000012345678
IVI NID                : 68
CTL TTL                :   0b
SEQ                    :     000006
SRC                    :           1201
DST                    :               0003
TransportPDU           :                   0089511bf1d1a81c11dcef
NetMIC Size            : 32 bits
EncTransportPDU        :               6b9be7f5a642f2f98680e61c3a
NetMIC                 :                                       8b47f228

Obfuscation

Privacy Random         : 0000000000123456786b9be7f5a64287
PECB                   : b037280f9de9
preObfuscation         : 0b0000061201

NetworkPDU             : 68e80e5da5af0e6b9be7f5a642f2f98680e61c3a8b47f228
```

### 8.3.17 Message #17

A Relay node receives the message from the Low Power node and relays it, decrementing the TTL value.

```
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 000a0000061201000012345678
IVI NID                : 68
CTL TTL                :   0a
SEQ                    :     000006
SRC                    :           1201
DST                    :               0003
TransportPDU           :                   0089511bf1d1a81c11dcef
NetMIC Size            : 32 bits
EncTransportPDU        :               6b9be7f5a642f2f98680e61c3a
NetMIC                 :                                       8b47f228
```

```
Obfuscation

Privacy Random         : 0000000000123456786b9be7f5a642f2
PECB                   : e30e5da3bd0f
preObfuscation         : 0a0000061201
NetworkPDU             : 68e80e5da5af0e6b9be7f5a642f2f98680e61c3a8b47f228
```

### 8.3.18 Message #18

The Low Power node sends a Health Current Status message indicating that there are no faults.

```
Access Payload

Opcode                 : 04 (Health Current Status)
TestID:                : 00
CompanyID              : 0000
Faults                 : 00
Access Payload         : 0400000000

UpperTransportAccessPDU

IV Index               : 12345678
AppKey                 : 63964771734fbd76e3b40519d1d94a48
TTL                    : 03
SEQ                    : 000007
SRC                    : 1201
DST                    : ffff
Application Nonce      : 01000000071201ffff12345678
EncAccessPayload       : 5a8bde6d91
TransMIC Size          : 32 bits
TransMIC               : 06ea078a
UpperTransportPDU      : 5a8bde6d9106ea078a

LowerTransportUnsegmentedAccessPDU

CTL                    : 00
TTL                    : 03
SEQ                    : 000007
SRC                    : 1201
DST                    : ffff
SEG                    : 00
AKF                    : 01
AID                    : 26
Header                 : 66
```

```
UpperTransportPDU     :   5a8bde6d9106ea078a
LowerTransportPDU     : 665a8bde6d9106ea078a

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 00
TTL                   : 03
SEQ                   : 000007
SRC                   : 1201
DST                   : ffff
LowerTransportPDU     : 665a8bde6d9106ea078a
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 00030000071201000012345678
IVI NID               : 68
CTL TTL               :   03
SEQ                   :     000007
SRC                   :           1201
DST                   :               ffff
TransportPDU          :                   665a8bde6d9106ea078a
NetMIC Size           : 32 bits
EncTransportPDU       :               5673728a627fb938535508e2
NetMIC                :                                         1a6baf57

Obfuscation

Privacy Random        : 0000000000123456785673728a627fb9
PECB                  : 4bcba430940f
preObfuscation        : 030000071201

NetworkPDU            : 6848cba437860e5673728a627fb938535508e21a6baf57
```

### 8.3.19 Message #19

The Low Power node sends another Health Current Status message indicating that there are three faults: Battery Low Warning, Power Supply Interrupted Warning, and Supply Voltage Too Low Warning.

```
Access Payload

Opcode                : 04 (Health Current Status)
TestID                : 00
CompanyID             : 0000
```

```
Faults                  : 010703
Access Payload          : 04000000010703

UpperTransportAccessPDU

IV Index                : 12345678
AppKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 03
SEQ                     : 000009
SRC                     : 1201
DST                     : ffff
Application Nonce       : 01000000091201ffff12345678
EncAccessPayload        : ca6cd88e698d12
TransMIC Size           : 32 bits
TransMIC                : 65f43fc5
UpperTransportPDU       : ca6cd88e698d1265f43fc5

LowerTransportSegmentedAccessPDU

CTL                     : 00
TTL                     : 03
SEQ                     : 000009
SRC                     : 1201
DST                     : ffff
SEG                     : 01
AKF                     : 01
AID                     : 26
UpperTransportPDU       :   ca6cd88e698d1265f43fc5
LowerTransportPDU       : 66ca6cd88e698d1265f43fc5

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 00
TTL                     : 03
SEQ                     : 000009
SRC                     : 1201
DST                     : ffff
LowerTransportPDU       : 66ca6cd88e698d1265f43fc5
NID                     : 68
EncryptionKey           : 0953fa93e7caac9638f58820220a398e
PrivacyKey              : 8b84eedec100067d670971dd2aa700cf
Network Nonce           : 0003000009120100001234567 8
IVI NID                 : 68
```

```
CTL TTL                 :   03
SEQ                     :     000009
SRC                     :           1201
DST                     :               ffff
TransportPDU            :                   66ca6cd88e698d1265f43fc5
NetMIC Size             : 32 bits
EncTransportPDU         :               3010a05e1b23a926023da75d25ba
NetMIC                  :                                   91793736

Obfuscation

Privacy Random          : 0000000000123456783010a05e1b23a9
PECB                    : 120edee5ca3d
preObfuscation          : 030000091201

NetworkPDU              : 68110edeecd83c3010a05e1b23a926023da75d25ba91793736
```

### 8.3.20 Message #20

The Low Power node sends the Health Current Status message with the same three faults.

```
Access Payload

Opcode                  : 04 (Health Current Status)
TestID                  : 00
CompanyID               : 0000
Faults                  : 010703
Access Payload          : 04000000010703

UpperTransportAccessPDU

IV Index                : 12345677
AppKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 03
SEQ                     : 070809
SRC                     : 1234
DST                     : ffff
Application Nonce       : 01000708091234ffff12345677
EncAccessPayload        : 9c9803e110fea9
TransMIC Size           : 32 bits
TransMIC                : 29e9542d
UpperTransportPDU       : 9c9803e110fea929e9542d

LowerTransportSegmentedAccessPDU
```

```
CTL                    : 00
TTL                    : 03
SEQ                    : 070809
SRC                    : 1234
DST                    : ffff
SEG                    : 01
AKF                    : 01
AID                    : 26
Header                 : 66
Segment#0              :   9c9803e110fea929e9542d
LowerTransportPDU      : 669c9803e110fea929e9542d

NetworkPDU

IVindex                : 12345677
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 00
TTL                    : 03
SEQ                    : 070809
SRC                    : 1234
DST                    : ffff
LowerTransportPDU      : 669c9803e110fea929e9542d
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 00030708091234000012345677
IVI NID                : e8
CTL TTL                :   03
SEQ                    :     070809
SRC                    :           1234
DST                    :               ffff
TransportPDU           :                   669c9803e110fea929e9542d
NetMIC Size            : 32 bits
EncTransportPDU        :               8c3dc87344a16c787f6b08cc897c
NetMIC                 :                                           941a5368

Obfuscation

Privacy Random         : 0000000000123456778c3dc87344a16c
PECB                   : 5fcd59ebfaad
preObfuscation         : 030708091234

NetworkPDU             : e85cca51e2e8998c3dc87344a16c787f6b08cc897c941a5368
```

### 8.3.21 Message #21

The Low Power node sends a vendor command to a group address.

```
Access Payload

Opcode                  : 15 : 000a (Vendor 15 : 000a)
Params                  : 48656c6c6f
Access Payload          : d50a0048656c6c6f

UpperTransportAccessPDU

IV Index                : 12345677
AppKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 03
SEQ                     : 07080a
SRC                     : 1234
DST                     : 8105
Application Nonce       : 010007080a1234810512345677
EncAccessPayload        : 2fa730fd98f6e4bd
TransMIC Size           : 32 bits
TransMIC                : 120ea9d6
UpperTransportPDU       : 2fa730fd98f6e4bd120ea9d6

LowerTransportUnsegmentedAccessPDU

CTL                     : 00
TTL                     : 03
SEQ                     : 07080a
SRC                     : 1234
DST                     : 8105
SEG                     : 00
AKF                     : 01
AID                     : 26
Header                  : 66
UpperTransportPDU       :   2fa730fd98f6e4bd120ea9d6
LowerTransportPDU       : 662fa730fd98f6e4bd120ea9d6

NetworkPDU

IVindex                 : 12345677
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 00
TTL                     : 03
SEQ                     : 07080a
```

```
SRC                   : 1234
DST                   : 8105
LowerTransportPDU     : 662fa730fd98f6e4bd120ea9d6
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 000307080a1234000012345677
IVI NID               : e8
CTL TTL               :   03
SEQ                   :     07080a
SRC                   :           1234
DST                   :               8105
TransportPDU          :                   662fa730fd98f6e4bd120ea9d6
NetMIC Size           : 32 bits
EncTransportPDU       :               e4d611358eaf17796a6c98977f69e5
NetMIC                :                                             872c4620

Obfuscation

Privacy Random        : 0000000000012345677e4d611358eaf17
PECB                  : b2021754866e
preObfuscation        : 0307080a1234

NetworkPDU            : e8b1051f5e945ae4d611358eaf17796a6c98977f69e5872c4620
```

### 8.3.22 Message #22

The Low Power node sends a vendor command to a virtual address.

```
Access Payload

Opcode                : 15 : 000a (Vendor 15 : 000a)
Params                : 48656c6c6f
Access Payload        : d50a0048656c6c6f

UpperTransportAccessPDU

IV Index              : 12345677
AppKey                : 63964771734fbd76e3b40519d1d94a48
TTL                   : 03
SEQ                   : 07080b
SRC                   : 1234
Label UUID            : 0073e7e4d8b9440faf8415df4c56c0e1
DST (Virtual Address) : b529
Application Nonce     : 010007080b1234b52912345677
```

```
EncAccessPayload       : 3871b904d4315263
TransMIC Size          : 32 bits
TransMIC               : 16ca48a0
UpperTransportPDU      : 3871b904d431526316ca48a0

LowerTransportUnsegmentedAccessPDU

CTL                    : 00
TTL                    : 03
SEQ                    : 07080b
SRC                    : 1234
DST                    : b529
SEG                    : 00
AKF                    : 01
AID                    : 26
Header                 : 66
UpperTransportPDU      :   3871b904d431526316ca48a0
LowerTransportPDU      : 663871b904d431526316ca48a0

NetworkPDU

IVindex                : 12345677
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 00
TTL                    : 03
SEQ                    : 07080b
SRC                    : 1234
DST                    : b529
LowerTransportPDU      : 343871b904d431526316ca48a0
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 000307080b1234000012345677
IVI NID                : e8
CTL TTL                :   03
SEQ                    :     07080b
SRC                    :           1234
DST                    :               b529
TransportPDU           :                   663871b904d431526316ca48a0
NetMIC Size            : 32 bits
EncTransportPDU        :               ed31f3fdcf88a411135fea55df730b
NetMIC                 :                                             6b28e255

Obfuscation
```

```
Privacy Random         : 0000000000012345677ed31f3fdcf88a4
PECB                   : db5ba6c5e3d7
preObfuscation         : 0307080b1234

NetworkPDU             : e8d85caecef1e3ed31f3fdcf88a411135fea55df730b6b28e255
```

### 8.3.23 Message #23

The Low Power node sends a vendor command to a different virtual address.

```
Access Payload

Opcode                 : 15 : 000a (Vendor 15 : 000a)
Params                 : 48656c6c6f
Access Payload         : d50a0048656c6c6f

UpperTransportAccessPDU

IV Index               : 12345677
AppKey                 : 63964771734fbd76e3b40519d1d94a48
TTL                    : 03
SEQ                    : 07080c
SRC                    : 1234
Label UUID             : f4a002c7fb1e4ca0a469a021de0db875
DST (Virtual Address)  : 9736
Application Nonce      : 010007080c1234973612345677
EncAccessPayload       : 2456db5e3100eef6
TransMIC Size          : 32 bits
TransMIC               : 5daa7a38
UpperTransportPDU      : 2456db5e3100eef65daa7a38

LowerTransportUnsegmentedAccessPDU

CTL                    : 00
TTL                    : 03
SEQ                    : 07080c
SRC                    : 1234
DST                    : 9736
SEG                    : 00
AKF                    : 01
AID                    : 26
Header                 : 66
UpperTransportPDU      :   2456db5e3100eef65daa7a38
LowerTransportPDU      : 662456db5e3100eef65daa7a38
```

```
NetworkPDU

IVindex               : 12345677
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 00
TTL                   : 03
SEQ                   : 07080c
SRC                   : 1234
DST                   : 9736
LowerTransportPDU     : 342456db5e3100eef65daa7a38
NID                   : 68
EncryptionKey         : 0953fa93e7caac9638f58820220a398e
PrivacyKey            : 8b84eedec100067d670971dd2aa700cf
Network Nonce         : 000307080c1234000012345677
IVI NID               : e8
CTL TTL               :   03
SEQ                   :     07080c
SRC                   :           1234
DST                   :               9736
TransportPDU          :                   662456db5e3100eef65daa7a38
NetMIC Size           : 32 bits
EncTransportPDU       :               7a9d696d3dd16a75489696f0b70c71
NetMIC                :                                             1b881385

Obfuscation

Privacy Random        : 0000000000123456777a9d696d3dd16a
PECB                  : 74a385d9ec19
preObfuscation        : 0307080c1234

NetworkPDU            : e877a48dd5fe2d7a9d696d3dd16a75489696f0b70c711b881385
```

### 8.3.24 Message #24

The Low Power node sends a vendor command to a virtual address using a 64-bit TransMIC.

```
Access Payload

Opcode                : 2A : 000a (Vendor 2a : 000a)
Params                : 576f726c64
Access Payload        : ea0a00576f726c64

UpperTransportAccessPDU

IV Index              : 12345677
```

```
AppKey                   : 63964771734fbd76e3b40519d1d94a48
TTL                      : 03
SEQ                      : 07080d
SRC                      : 1234
Label UUID               : f4a002c7fb1e4ca0a469a021de0db875
DST (Virtual Address)    : 9736
Application Nonce        : 010007080d1234973612345677
EncAccessPayload         : de1547118463123e
TransMIC Size            : 64 bits
TransMIC                 : 5f6a17b99dbca387
UpperTransportPDU        : de1547118463123e5f6a17b99dbca387
Segment#0                : de1547118463123e5f6a17b9
Segment#1                : 9dbca387

LowerTransportSegmentedAccessPDU

CTL                      : 00
TTL                      : 03
SEQ                      : 07080d
SRC                      : 1234
DST                      : 9736
SEG                      : 01
AKF                      : 01
AID                      : 26
SZMIC                    : 01
SeqZero                  : 80d
SegO                     : 00
SegN                     : 01
Header                   : e6a03401
Segment#0                :         de1547118463123e5f6a17b9
LowerTransportPDU        : e6a03401de1547118463123e5f6a17b9

NetworkPDU

IVindex                  : 12345677
NetKey                   : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                      : 00
TTL                      : 03
SEQ                      : 07080d
SRC                      : 1234
DST                      : 9736
LowerTransportPDU        : e6a03401de1547118463123e5f6a17b9
NID                      : 68
EncryptionKey            : 0953fa93e7caac9638f58820220a398e
PrivacyKey               : 8b84eedec100067d670971dd2aa700cf
```

```
Network Nonce          : 000307080d1234000012345677
IVI NID                : e8
CTL TTL                :   03
SEQ                    :     07080d
SRC                    :           1234
DST                    :               9736
TransportPDU           :                   e6a03401de1547118463123e5f6a17b9
NetMIC Size            : 32 bits
EncTransportPDU        :               94e998b4081f5a7308ce3edbb3b06cdecd02
NetMIC                 :                                                   8e307f1c

Obfuscation

Privacy Random         : 0000000000123456779 4e998b4081f5a
PECB                   : 375f63a6ccc7
preObfuscation         : 0307080d1234

NetworkPDU             : e834586babdef394e998b4081f5a7308ce3edbb3b06cdecd028e307f1c

LowerTransportSegmentedAccessPDU

CTL                    : 00
TTL                    : 03
SEQ                    : 07080e
SRC                    : 1234
DST                    : 9736
SEG                    : 01
AKF                    : 01
AID                    : 26
SZMIC                  : 01
SeqZero                : 7080d
SegO                   : 01
SegN                   : 01
Header                 : e6a03421
Segment#1              :         9dbca387
LowerTransportPDU      : e6a034219dbca387

NetworkPDU

IVindex                : 12345677
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 00
TTL                    : 03
SEQ                    : 07080e
SRC                    : 1234
```

```
DST                    : 9736
LowerTransportPDU      : e6a034219dbca387
NID                    : 68
EncryptionKey          : 0953fa93e7caac9638f58820220a398e
PrivacyKey             : 8b84eedec100067d670971dd2aa700cf
Network Nonce          : 000307080e1234000012345677
IVI NID                : e8
CTL TTL                :   03
SEQ                    :     07080e
SRC                    :           1234
DST                    :               9736
TransportPDU           :                   e6a034219dbca387
NetMIC Size            : 32 bits
EncTransportPDU        :               dc2f4dd6fb4d32870129
NetMIC                 :                                   1be4aafe

Obfuscation

Privacy Random         : 00000000012345677dc2f4dd6fb4d32
PECB                   : 5212a77dcec9
preObfuscation         : 0307080e1234

NetworkPDU             : e85115af73dcfddc2f4dd6fb4d328701291be4aafe
```

## 8.4 Beacon sample data

The following beacon sample data shows examples of beacons.

### 8.4.1 Unprovisioned device beacon (without URI)

This shows an example of an unprovisioned device. This device has some OOB information that is an ASCII-string written on a piece of paper and on the device itself.

```
Device UUID            : 70cf7c9732a345b691494810d2e9cbf4
OOB                    : String, on piece of paper, on device
OOB Information        : a040

Beacon                 : 0070cf7c9732a345b691494810d2e9cbf4a040
```

### 8.4.2 Unprovisioned device beacon (with URI)

This shows an example of an unprovisioned device that also includes some OOB information as a number on the inside of the manual, and also a URI-hash that can be used to help identify the device.

```
Device UUID            : 70cf7c9732a345b691494810d2e9cbf4
OOB                    : Number, Inside Manual
```

```
OOB Information       : 4020
URI                   : https://www.example.com/mesh/products/light-switch-v3
URI Data              : 172f2f7777772e6578616d706c652e636f6d2f6d6573682f70726f6475637473
                        2f6c696768742d7377697463682d7633
URI Hash              : d97478b3667f4839487469c72b8e5e9e

Beacon                : 0070cf7c9732a345b691494810d2e9cbf44020d97478b3
```

### 8.4.3 Secure Network beacon

This shows an example of a Secure Network beacon that includes the IV Index for a given network key. There are now key refresh or IV updates occurring at this time.

```
IV Update Flag        : 0
Key Refresh Flag      : 0
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
IV Index              : 12345678
Flags                 : 00
Network ID            : 3ecaff672f673370
Message               : 003ecaff672f67337012345678
BeaconKey             : 5423d967da639a99cb02231a83f7d254
Authentication Value  : 8ea261582f364f6f3c74ef80336ca17e

Beacon                : 01003ecaff672f673370123456788ea261582f364f6f
```

### 8.4.4 Secure Network beacon (IV update in progress)

This shows an example Secure Network beacon that would be sent when the IV Index is being updated. In this example, the IV Index has been incremented and the IV Update Flag is set.

```
IV Update Flag        : 1
Key Refresh Flag      : 0
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
IV Index              : 12345679
Flags                 : 02
Network ID            : 3ecaff672f673370
Message               : 023ecaff672f67337012345679
BeaconKey             : 5423d967da639a99cb02231a83f7d254
Authentication Value  : c2af80ad072a135c28cf843369887039

Beacon                : 01023ecaff672f67337012345679c2af80ad072a135c
```

### 8.4.5 Secure Network beacon (IV update complete)

This shows an example of a Secure Network beacon after the IV Index has been updated. The IV Index has the same value as the previous Secure Network beacon, but the IV Update Flag is now cleared.

```
IV Update Flag         : 0
Key Refresh Flag       : 0
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
IV Index               : 12345679
Flags                  : 00
Network ID             : 3ecaff672f673370
Message                : 003ecaff672f67337012345679
BeaconKey              : 5423d967da639a99cb02231a83f7d254
Authentication Value   : c62f09e4c957f59d96f506f64604bfc1

Beacon                 : 01003ecaff672f67337012345679c62f09e4c957f59d
```

## 8.5 Provisioning Service sample data

The following sample data shows the advertising data for a Provisioning Service advertisement.

### 8.5.1 Mesh Provisioning Service advertising service data

This sample data shows that the device has some OOB information as a number that is printed inside of the manual.

```
Device UUID            : 70cf7c9732a345b691494810d2e9cbf4
OOB                    : Number, Inside Manual
OOB Information        : 4020

Adv Len                : 15
Adv (Service Data)     :   16
Mesh Provisioning UUID:      1827
Device UUID            :           70cf7c9732a345b691494810d2e9cbf4
OOB Information        :                                               4020
ADV Data               : 1516271870cf7c9732a345b691494810d2e9cbf44020
```

## 8.6 Mesh Proxy Service sample data

The following sample data shows Mesh Proxy service sample data.

### 8.6.1 Service data using Network ID

This shows the advertising data used to broadcast the Network ID of a network that can be accessed through a Mesh Proxy service. This would be used to allow a device to connect to a specific mesh network.

```
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
Adv Len                : 0c
Adv (Service Data)     :   16
Mesh Proxy UUID        :      1828
Type                   :           00
```

```
Network ID             :          3ecaff672f673370
ADV Data               : 0c162818003ecaff672f673370
```

### 8.6.2 Service data using Node Identity

This shows the advertising data used to broadcast the identity of a node in a private manner that can be accessed through a Mesh Proxy service. This would be used to allow a device to connect to a specific node.

```
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
Source Address         : 1201
Random                 : 34ae608fbbc1f2c6
Nonce                  : 000000000034ae608fbbc1f2c61201
ResolvingKey           : 84396c435ac48560b5965385253e210c
Adv Len                : 14
Adv (Service Data)     :   16
Mesh Proxy UUID        :     1828
Type                   :         01
Hash                   :           00861765aefcc57b
Random                 :                           34ae608fbbc1f2c6
ADV Data               : 141628180100861765aefcc57b34ae608fbbc1f2c6
```

## 8.7 PB-ADV provisioning sample data

The following sample data shows a complete set of provisioning transactions over PB-ADV. The following sample data is based on the provisioning and device private keys documented below.

```
Prov Private Key       : 06a516693c9aa31a6084545d0c5db641b48572b97203ddffb7ac73f7d0457663
Prov Public Key        : 2c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78dd
                         919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d4f
Device Private Key     : 529aa0670d72cd6497502ed473502b037e8803b5c60829a5a3caa219505530ba
Device Public Key      : f465e43ff23d3f1b9dc7dfc04da8758184dbc966204796eccf0d6cf5e16500cc
                         0201d048bcbbd899eeefc424164e33c201c2b010ca6b4d43a8a155cad8ecb279
Prov ECDH              : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
Device ECDH            : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
Prov Random            : 8b19ac31d58b124c946209b5db1021b9
Device Random          : 55a2a2bca04cd32ff6f346bd0a0c1a3a
```

### 8.7.1 PB-ADV Link Open

The Provisioner, after receiving the Mesh Provisioning Service advertising data, will send a Link Open message including the Device UUID of the new device.

```
Link Open

Device UUID            : 70cf7c9732a345b691494810d2e9cbf4
```

```
Provisioning Control

LinkID              : 23af5850
Transaction         :        00
Opcode              : 00 (Link Open)
Message             : 23af5850000370cf7c9732a345b691494810d2e9cbf4
```

### 8.7.2 PB-ADV Link Ack

The new device will respond with a Link Ack to confirm that it is accepting this Link Open. All other messages after this point will use this LinkID to identify this session until the Link Close message is received.

```
Link Ack

Provisioning Control

LinkID              : 23af5850
Transaction         :        00
Opcode              : 01 (Link Ack)
Message             : 23af58500007
```

### 8.7.3 PB-ADV Provisioning Invite

The Provisioner will invite the new device to join the mesh network. This includes a duration for which the new device will attract attention of the user such that the user knows which new device is currently being provisioned. In this case, the duration is 0 seconds, meaning that the Provisioner didn't need the new device to make itself known to the user.

```
Provisioning Invite

Duration            : 00 (0 seconds)
Message             : 0000

LinkID              : 23af5850
Transaction         :        00

Segment0            : 0000
SegN                : 00
PDU Length          : 0002
FCS                 : 14
Message0            : 23af5850000000002140000
```

#### 8.7.3.1 PB-ADV Transaction Ack

The Provisioner acknowledges the invite transaction.

```
PBADV Transaction Acknowledge

LinkID              : 23af5850
Transaction         :         00
Message             : 23af58500001
```

### 8.7.4 PB-ADV Provisioning Capabilities

The new device responds to the invite by sending its capabilities. The new device has only one element, supports the mandatory P-256 algorithm, and has no OOB information or input or output capabilities.

```
Provisioning Capabilities

Num Elements        : 01
Algorithms          : 0001
Pub Key Type        : 00
Static OOB          : 00
Output OOB Size     : 00
Output OOB Action   : 0000
Input OOB Size      : 00
Input Action        : 0000
Message             : 0101000100000000000000

LinkID              : 23af5850
Transaction         :         80

Segment0            : 0101000100000000000000
SegN                : 00
PDU Length          : 000c
FCS                 : d6
Message0            : 23af58508000000cd6010100010000000000000000
```

#### 8.7.4.1 PB-ADV Transaction Ack

The Provisioner acknowledges the capabilities transaction.

```
PBADV Transaction Acknowledge

LinkID              : 23af5850
Transaction         :         80
Message             : 23af58508001
```

### 8.7.5 PB-ADV Provisioning Start

The Provisioner upon receiving the capabilities, sends a provisioning start message stating the algorithm and authentication method to use.

```
Provisioning Start

Algorithm             : 00
Pub Key Type          : 0000
Authentication Method : 00
Authentication Action : 00
Authentication Size   : 00
Message               : 020000000000

LinkID                : 23af5850
Transaction           :         01

Segment0              : 020000000000
SegN                  : 00
PDU Length            : 0006
FCS                   : 64
Message0              : 23af58500100000664020000000000
```

#### 8.7.5.1 PB-ADV Transaction Ack

The new device acknowledges this start transaction.

```
PBADV Transaction Acknowledge

LinkID                : 23af5850
Transaction           :         01
Message               : 23af58500101
```

### 8.7.6 PB-ADV Provisioning Public Key (Provisioner)

The Provisioner sends its public key to the new device. This message contains the 512-bit public key, and is therefore transmitted using three segments in three messages.

```
Provisioning Start

Public Key            : 2c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78dd
                        919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d4f
Message               : 032c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78
                        dd919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d
                        4f

LinkID                : 23af5850
Transaction           :         02

Segment0              : 032c31a47b5779809ef44cb5eaaf5c3e43d5f8fa
Segment1              : ad4a8794cb987e9b03745c78dd919512183898dfbecd52
```

```
Segment2              : e2408e43871fd021109117bd3ed4eaf8437743715d4f
SegN                  : 02
PDU Length            : 0041
FCS                   : d1
Message0              : 23af585002080041d1032c31a47b5779809ef44cb5eaaf5c3e43d5f8fa
Message1              : 23af58500206ad4a8794cb987e9b03745c78dd919512183898dfbecd52
Message2              : 23af5850020ae2408e43871fd021109117bd3ed4eaf8437743715d4f
```

#### 8.7.6.1 PB-ADV Transaction Ack

Once the new device receives all the segments of the provisioning public key transaction, it will send the acknowledgment for that transaction.

```
PBADV Transaction Acknowledge

LinkID                : 23af5850
Transaction           :         02
Message               : 23af58500201
```

### 8.7.7 PB-ADV Provisioning Public Key (Device)

The device sends its public key to the Provisioner. This is again a multi-segment transaction.

```
Provisioning Start

Public Key            : f465e43ff23d3f1b9dc7dfc04da8758184dbc966204796eccf0d6cf5e16500cc
                        0201d048bcbbd899eeefc424164e33c201c2b010ca6b4d43a8a155cad8ecb279
Message               : 03f465e43ff23d3f1b9dc7dfc04da8758184dbc966204796eccf0d6cf5e16500
                        cc0201d048bcbbd899eeefc424164e33c201c2b010ca6b4d43a8a155cad8ecb2
                        79

LinkID                : 23af5850
Transaction           :         81

Segment0              : 03f465e43ff23d3f1b9dc7dfc04da8758184dbc9
Segment1              : 66204796eccf0d6cf5e16500cc0201d048bcbbd899eeef
Segment2              : c424164e33c201c2b010ca6b4d43a8a155cad8ecb279
SegN                  : 02
PDU Length            : 0041
FCS                   : 10
Message0              : 23af5850810800411003f465e43ff23d3f1b9dc7dfc04da8758184dbc9
Message1              : 23af5850810666204796eccf0d6cf5e16500cc0201d048bcbbd899eeef
Message2              : 23af5850810ac424164e33c201c2b010ca6b4d43a8a155cad8ecb279
```

#### 8.7.7.1 PB-ADV Transaction Ack

The Provisioner sends a transaction acknowledgment when it receives all the segments of the public key transaction.

```
PBADV Transaction Acknowledge

LinkID                : 23af5850
Transaction           :         81
Message               : 23af58508101
```

### 8.7.8 PB-ADV Provisioning Confirmation (Provisioner)

The Provisioner will calculate a confirmation value that is based off of all the information already exchanged, a random number that has not been exchanged yet, and an authentication value that is communicated OOB. As this provisioning is not using any authentication, the AuthValue is set to 0.

```
InvitePDUValue        : 00
CapabilitiesPDUValue  : 0100010000000000000000
StartPDUValue         : 0000000000
ProvisionerPublicKey  : 2c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78dd
                        919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d4f
DevicePublicKey       : f465e43ff23d3f1b9dc7dfc04da8758184dbc966204796eccf0d6cf5e16500cc
                        0201d048bcbbd899eeefc424164e33c201c2b010ca6b4d43a8a155cad8ecb279
ConfirmationInputs    : 00010001000000000000000000000000002c31a47b5779809ef44cb5eaaf5c3e
                        43d5f8faad4a8794cb987e9b03745c78dd919512183898dfbecd52e2408e4387
                        1fd021109117bd3ed4eaf8437743715d4ff465e43ff23d3f1b9dc7dfc04da875
                        8184dbc966204796eccf0d6cf5e16500cc0201d048bcbbd899eeefc424164e33
                        c201c2b010ca6b4d43a8a155cad8ecb279
ConfirmationSalt      : 5faabe187337c71cc6c973369dcaa79a
ECDHSecret            : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
k1 N                  : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
k1 SALT               : 5faabe187337c71cc6c973369dcaa79a
k1 P                  : 7072636b
k1 T                  : ace84c6f002e0b4c23467e75687bae8f
ConfirmationKey       : e31fe046c68ec339c425fc6629f0336f
RandomProvisioner     : 8b19ac31d58b124c946209b5db1021b9
AuthValue             : 00000000000000000000000000000000

Provisioning Confirmation

Confirmation          : b38a114dfdca1fe153bd2c1e0dc46ac2
Message               : 05b38a114dfdca1fe153bd2c1e0dc46ac2
LinkID                : 23af5850
Transaction           :         03
```

```
Segment0              : 05b38a114dfdca1fe153bd2c1e0dc46ac2
SegN                  : 00
PDU Length            : 0011
FCS                   : d1
Message0              : 23af585003000011d105b38a114dfdca1fe153bd2c1e0dc46ac2
```

### 8.7.8.1 PB-ADV Transaction Ack

The new device will acknowledge the confirmation transaction.

```
PBADV Transaction Acknowledge

LinkID                : 23af5850
Transaction           :         03
Message               : 23af58500301
```

## 8.7.9 PB-ADV Provisioning Confirmation (Device)

The new device will send its confirmation value using all the information that it has exchanged so far, the authentication value communicated OOB, and a random number that has not been disclosed yet.

```
InvitePDUValue        : 00
CapabilitiesPDUValue  : 0100010000000000000000
StartPDUValue         : 0000000000
ProvisionerPublicKey  : 2c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78dd
                        919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d4f
DevicePublicKey       : f465e43ff23d3f1b9dc7dfc04da8758184dbc966204796eccf0d6cf5e16500cc
                        0201d048bcbbd899eeefc424164e33c201c2b010ca6b4d43a8a155cad8ecb279
ConfirmationInputs    : 00010001000000000000000000000000002c31a47b5779809ef44cb5eaaf5c3e
                        43d5f8faad4a8794cb987e9b03745c78dd919512183898dfbecd52e2408e4387
                        1fd021109117bd3ed4eaf8437743715d4ff465e43ff23d3f1b9dc7dfc04da875
                        8184dbc966204796eccf0d6cf5e16500cc0201d048bcbbd899eeefc424164e33
                        c201c2b010ca6b4d43a8a155cad8ecb279
ConfirmationSalt      : 5faabe187337c71cc6c973369dcaa79a
ECDHSecret            : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
k1 N                  : ab85843a2f6d883f62e5684b38e307335fe6e1945ecd19604105c6f23221eb69
k1 SALT               : 5faabe187337c71cc6c973369dcaa79a
k1 P                  : 7072636b
k1 T                  : ace84c6f002e0b4c23467e75687bae8f
ConfirmationKey       : e31fe046c68ec339c425fc6629f0336f
RandomDevice          : 55a2a2bca04cd32ff6f346bd0a0c1a3a
AuthValue             : 00000000000000000000000000000000

Provisioning Confirmation

Confirmation          : eeba521c196b52cc2e37aa40329f554e
```

```
Message                 : 05eeba521c196b52cc2e37aa40329f554e

LinkID                  : 23af5850
Transaction             :         82

Segment0                : 05eeba521c196b52cc2e37aa40329f554e
SegN                    : 00
PDU Length              : 0011
FCS                     : ec
Message0                : 23af585082000011ec05eeba521c196b52cc2e37aa40329f554e
```

#### 8.7.9.1 PB-ADV Transaction Ack

The Provisioner will acknowledge the confirmation transaction from the new device.

```
PBADV Transaction Acknowledge

LinkID                  : 23af5850
Transaction             :         82
Message                 : 23af58508201
```

### 8.7.10 PB-ADV Provisioning Random (Provisioner)

The Provisioner will now expose its random number used to generate its confirmation value that it has previously committed to.

```
Provisioning Random

Random                  : 8b19ac31d58b124c946209b5db1021b9
Message                 : 068b19ac31d58b124c946209b5db1021b9

LinkID                  : 23af5850
Transaction             :         04

Segment0                : 068b19ac31d58b124c946209b5db1021b9
SegN                    : 00
PDU Length              : 0011
FCS                     : d3
Message0                : 23af585004000011d3068b19ac31d58b124c946209b5db1021b9
```

#### 8.7.10.1 PB-ADV Transaction Ack

The new device acknowledges this random number transaction.

```
PBADV Transaction Acknowledge

LinkID                  : 23af5850
```

```
Transaction            :         04
Message                : 23af58500401
```

### 8.7.11 PB-ADV Provisioning Random (Device)

The new device now sends its random number to the Provisioner.

```
Provisioning Random

Random                 : 55a2a2bca04cd32ff6f346bd0a0c1a3a
Message                : 0655a2a2bca04cd32ff6f346bd0a0c1a3a

LinkID                 : 23af5850
Transaction            :         83

Segment0               : 0655a2a2bca04cd32ff6f346bd0a0c1a3a
SegN                   : 00
PDU Length             : 0011
FCS                    : 59
Message0               : 23af585083000011590655a2a2bca04cd32ff6f346bd0a0c1a3a
```

#### 8.7.11.1 PB-ADV Transaction Ack

The Provisioner acknowledges receiving this random number transaction from the new device.

```
PBADV Transaction Acknowledge

LinkID                 : 23af5850
Transaction            :         83
Message                : 23af58508301
```

### 8.7.12 PB-ADV Provisioning Data

The Provisioner can now provide the provisioning data required by the new device to become a node in a mesh network. This includes a NetKey along with a network key index, the current IV Index of this network key, and the unicast address of the first element of this node, and therefore all the subsequent addresses of additional elements. This data is encrypted and authenticated using a session key derived from the ECDH shared secret. This data requires two segments to communicate.

```
ConfirmationSalt       : 5faabe187337c71cc6c973369dcaa79a
Random Provisioner     : 8b19ac31d58b124c946209b5db1021b9
Random Device          : 55a2a2bca04cd32ff6f346bd0a0c1a3a
ProvisioningInputs     : 5faabe187337c71cc6c973369dcaa79a8b19ac31d58b124c946209b5db1021b9
                         55a2a2bca04cd32ff6f346bd0a0c1a3a
ProvisioningSalt       : a21c7d45f201cf9489a2fb57145015b4
DeviceKey              : 0520adad5e0142aa3e325087b4ec16d8
SessionKey             : c80253af86b33dfa450bbdb2a191fea3
```

```
Nonce                     :    da7ddbe78b5f62b81d6847487e
NetKey                    : efb2255e6422d330088e09bb015ed707
NetKeyIndex               : 0567
Flags                     : 00
IVIndex                   : 01020304
UnicastAddress            : 0b0c
ProvisioningData          : efb2255e6422d330088e09bb015ed707056700010203040b0c
EncProvisioningData       : d0bd7f4a89a2ff6222af59a90a60ad58acfe3123356f5cec29
ProvisioningDataMIC       : 73e0ec50783b10c7

Provisioning Data

EncProvisioningData       : d0bd7f4a89a2ff6222af59a90a60ad58acfe3123356f5cec29
ProvisioningDataMIC       : 73e0ec50783b10c7
Message                   : 07d0bd7f4a89a2ff6222af59a90a60ad58acfe3123356f5cec2973e0ec50783b
                            10c7

LinkID                    : 23af5850
Transaction               :        05

Segment0                  : 07d0bd7f4a89a2ff6222af59a90a60ad58acfe31
Segment1                  : 23356f5cec2973e0ec50783b10c7
SegN                      : 01
PDU Length                : 0022
FCS                       : 8b
Message0                  : 23af5850050400228b07d0bd7f4a89a2ff6222af59a90a60ad58acfe31
Message1                  : 23af5850050623356f5cec2973e0ec50783b10c7
```

### 8.7.12.1 PB-ADV Transaction Ack

The new device acknowledges the reception of the provisioning data transaction.

```
PBADV Transaction Acknowledge

LinkID                    : 23af5850
Transaction               :        05
Message                   : 23af58500501
```

## 8.7.13 PB-ADV Provisioning Complete

The new device now indicates that it has successfully received and processed the provisioning data.

```
Provisioning Complete

Message                   : 08
```

```
LinkID              : 23af5850
Transaction         :         84
Segment0            : 08
SegN                : 00
PDU Length          : 0001
FCS                 : 3e
Message0            : 23af5850840001003e08
```

#### 8.7.13.1 PB-ADV Transaction Ack

The Provisioner acknowledges receiving the provisioning complete transaction from the new device.

```
PBADV Transaction Acknowledge

LinkID              : 23af5850
Transaction         :         84
Message             : 23af58508401
```

### 8.7.14 PB-ADV Link Close

Finally, the Provisioner closes the PB-ADV session by using the Link Close message.

```
Link Close

Reason              : 00

Provisioning Control

LinkID              : 23af5850
Transaction         :         00
Opcode              : 02 (Link Close)
Message             : 23af5850000b00
```

## 8.8 PB-GATT SAR sample data

The Provisioner is using PB-GATT to transport Provisioning PDU. The ATT_MTU is 23. The Type is Provisioning Public Key (0x03) and the value of the Parameters is:

```
Provisioning PDU Type : 03 (Provisioning Public Key)
Public Key X          : 2c31a47b5779809ef44cb5eaaf5c3e43d5f8faad4a8794cb987e9b03745c78dd
Public Key Y          : 919512183898dfbecd52e2408e43871fd021109117bd3ed4eaf8437743715d4f
```

### 8.8.1 1st segment

```
ATT Opcode            : 52 (Write Command)
ATT Handle            : 0003
ATT Value             : 43032c31a47b5779809ef44cb5eaaf5c3e43d5f8fa
```

### 8.8.2 2nd segment

```
ATT Opcode            : 52 (Write Command)
ATT Handle            : 0003
ATT Value             : 83ad4a8794cb987e9b03745c78dd919512183898
```

### 8.8.3 3rd segment

```
ATT Opcode            : 52 (Write Command)
ATT Handle            : 0003
ATT Value             : 83dfbecd52e2408e43871fd021109117bd3ed4ea
```

### 8.8.4 4th segment

```
ATT Opcode            : 52 (Write Command)
ATT Handle            : 0003
ATT Value             : c3f8437743715d4f
```

## 8.9 Proxy Configuration Message sample data

The Proxy client is configuring Proxy to use whitelist filtering.

```
CTL                   : 01
ProxyFilterPkt        : 0000

NetKey                : d1aafb2a1a3c281cbdb0e960edfad852
NID                   : 10
EncryptionKey         : 3a4fe84a6cc2c6a766ea93f1084d4039
PrivacyKey            : f695fcce709ccface4d8b7a1e6e39d25
IV Index              : 12345678
ProxyNonce            : 0300000001000100001234 5678
IVI NID               : 10
CTL TTL               :   80
SEQ                   :     000001
SRC                   :           0001
DST                   :               0000
TransportPDU          :                   0000
NetMIC Size           : 64 bits
EncTransportPDU       :               8b8c2851
NetMIC                :                       2e792d3711f4b526

Obfuscation

PrivacyRandom         : 0000000000123456788b8c28512e792d
PECB                  : b86bd60ffbba6ca41e7109226f247a16
preObfuscation        : 800000010001

ProxyMessage          : 10386bd60efbbb8b8c28512e792d3711f4b526
```

```
ProxyPDU               : 0210386bd60efbbb8b8c28512e792d3711f4b526
```

## 8.10 Composition Data sample data

This sample represents an example of Composition Data Page 0 (see Section 4.2.1). The data below is a sequence of octets.

```
0C001A0001000800030000010501000000800100001003103F002A00
```

To help with parsing of this sequence of octets, it has been formatted with appropriate spacing characters.

```
0C00 1A00 0100 0800 0300 0001 05 01 0000 0080 0100 0010 0310 3F002A00
```

Note: The composition data is little-endian.

The above Composition Data Page 0 can be described as follows:

- CID is 0x000C
- PID is 0x001A
- VID is 0x0001
- CRPL is 0x0008
- Features is 0x0003 – Relay and Friend features.
- Loc is “front” – 0x0100
- NumS is 5
- NumV is 1
- The Bluetooth SIG Models supported are:
    - 0x0000, 0x8000, 0x0001, 0x1000, 0x1003
- The Vendor Models supported are:
    - Company Identifier 0x003F and Model Identifier 0x002A

## 8.11 Directed Forwarding sample data

This section provides informative examples of sample data that can be used to verify implementations.

### 8.11.1 PREQ with all unset options

A Directed Forwarding node sends a Path Request (PREQ) message as a Path Originator to discover a path. No PREQ options are set.

```
Transport Control Message

Opcode                 : 30 (Path Request)
  OPT.OBOO             : 00
  OPT.EA               : 00
  OPT.WP               : 00
  OPT                  : 00
```

```
  PD                    : 0607
  PO                    : 0405
  OFN                   : 0c
  OPM                   : 00

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
UpperTransportPDU       : 00060704050c00

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
SEG                     : 00
Opcode                  : 30
Header                  : 30
UpperTransportPDU       :   00060704050c00
LowerTransportPDU       : 3000060704050c00

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
LowerTransportPDU       : 3000060704050c00
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 0080123456040500001234 5678
IVI NID                 : 0d
CTL TTL                 :   80
SEQ                     :     123456
SRC                     :           0405
DST                     :               fffb
```

```
TransportPDU          :                   3000060704050c00
NetMIC Size           : 64 bits
EncTransportPDU       :               8ead6e19d63a97a45336
NetMIC                :                                   1a93820ac3aeeed6

Obfuscation

Privacy Random        : 8ead6e19d63a97
PECB                  : 5e5df0b31cc6af8a478b34a050207cd9
preObfuscation        : 801234560405

NetworkPDU            : 0dde4fc4e518c38ead6e19d63a97a453361a93820ac3aeeed6
```

### 8.11.2 PREQ with all set options

A Directed Forwarding node sends a Path Request (PREQ) message as a Delegate Path Originator to discover a path. All PREQ options are set.

```
Transport Control Message

Opcode                : 30 (Path Request)
  OPT.OBOO            : 01
  OPT.EA              : 01
  OPT.WP              : 01
  OPT                 : e0
  PD                  : 0607
  PO                  : 0405
  OFN                 : 0c
  OPM                 : 00
  PTX                 : 00
  DPO                 : 0777
  OAR                 : 05

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : fffb
UpperTransportPDU     : e0060704050c0000077705

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
```

```
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
SEG                     : 00
Opcode                  : 30
Header                  : 30
UpperTransportPDU       :   e0060704050c0000077705
LowerTransportPDU       : 30e0060704050c0000077705

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
LowerTransportPDU       : 30e0060704050c0000077705
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00801234560405000012345678
IVI NID                 : 0d
CTL TTL                 :   80
SEQ                     :     123456
SRC                     :           0405
DST                     :               fffb
TransportPDU            :                   30e0060704050c0000077705
NetMIC Size             : 64 bits
EncTransportPDU         :               8ead6ef9d63a97a45336eecf8189
NetMIC                  :                                           a34c669044bd3f1a

Obfuscation

Privacy Random          : 8ead6ef9d63a97
PECB                    : dfc9f48ce613291748658c3f40ea9b6e
preObfuscation          : 801234560405

NetworkPDU              : 0d5fdbc0dae2168ead6ef9d63a97a45336eecf8189a34c669044bd3f1a
```

### 8.11.3 PREP with all unset options

A Directed Forwarding node sends a Path Reply (PREP) message as a Path Destination to reply to a Path Request (PREQ) message. No PREP options are set.

```
Transport Control Message

Opcode                  : 31 (Path Reply)
  OPT.OBOD              : 00
  OPT.OBOO              : 00
  OPT.EA                : 00
  OPT.CRQ               : 00
  OPT                   : 00
  PD                    : 0607
  PO                    : 0405
  OFN                   : 0c
  OPM                   : 02

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 123456
SRC                     : 0607
DST                     : 1234
UpperTransportPDU       : 00060704050c02

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0607
DST                     : 1234
SEG                     : 00
Opcode                  : 31
Header                  : 31
UpperTransportPDU       :   00060704050c02
LowerTransportPDU       : 3100060704050c02

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0607
DST                     : 1234
LowerTransportPDU       : 3100060704050c02
NID                     : 0d
```

```
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00801234560607000012345678
IVI NID                : 0d
CTL TTL                :   80
SEQ                    :     123456
SRC                    :           0607
DST                    :               1234
TransportPDU           :                   3100060704050c02
NetMIC Size            : 64 bits
EncTransportPDU        :               85e26de309546a5ef9e7
NetMIC                 :                                   23a36a32b5d61ac8

Obfuscation

Privacy Random         : 85e26de309546a
PECB                   : cc6c0d8387b8dcc1da39966757ed3cd9
preObfuscation         : 801234560607

NetworkPDU             : 0d4c7e39d581bf85e26de309546a5ef9e723a36a32b5d61ac8
```

### 8.11.4 PREP with all set options

A Directed Forwarding node sends a Path Reply (PREP) message as a Delegate Path Destination to reply to a Path Request (PREQ) message. All PREP options are set.

```
Transport Control Message

Opcode                 : 31 (Path Reply)
  OPT.OBOD             : 01
  OPT.OBOO             : 01
  OPT.EA               : 01
  OPT.CRQ              : 01
  OPT                  : f0
  PD                   : 0607
  PO                   : 0405
  OFN                  : 0c
  OPM                  : 02
  DPD                  : 0777
  DAR                  : 05

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 123456
```

```
SRC                    : 0777
DST                    : 1234
UpperTransportPDU      : f0060704050c02077705

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 123456
SRC                    : 0777
DST                    : 1234
SEG                    : 00
Opcode                 : 31
Header                 : 31
UpperTransportPDU      :   f0060704050c02077705
LowerTransportPDU      : 31f0060704050c02077705

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 00
SEQ                    : 123456
SRC                    : 0777
DST                    : 1234
LowerTransportPDU      : 31f0060704050c02077705
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00801234560777000012345678
IVI NID                : 0d
CTL TTL                :   80
SEQ                    :     123456
SRC                    :           0777
DST                    :               1234
TransportPDU           :                   31f0060704050c02077705
NetMIC Size            : 64 bits
EncTransportPDU        :               310bb0affb3260a0eeb880f2c5
NetMIC                 :                                         2a4c9ecea3d75cf5

Obfuscation

Privacy Random         : 310bb0affb3260
PECB                   : 150d7efb6e4a91c5d1f53c1140bc5821
```

```
preObfuscation          : 801234560777

NetworkPDU              : 0d951f4aad693d310bb0affb3260a0eeb880f2c52a4c9ecea3d75cf5
```

### 8.11.5 PCONF

A Directed Forwarding node sends a Path Confirmation (PCONF) message as a Path Originator to confirm that a Path Request (PREP) message is received and the associated path can be used in both directions.

```
Transport Control Message

Opcode                  : 32 (Path Confirmation)
  OPT.OBOO              : 00
  OPT                   : 00
  PD                    : 0607
  PO                    : 0405
  OFN                   : 0c

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : 1234
UpperTransportPDU       : 00060704050c

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : 1234
SEG                     : 00
Opcode                  : 32
Header                  : 32
UpperTransportPDU       :   00060704050c
LowerTransportPDU       : 3200060704050c

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
```

```
TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : 1234
LowerTransportPDU     : 3200060704050c
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00801234560405000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123456
SRC                   :           0405
DST                   :               1234
TransportPDU          :                   3200060704050c
NetMIC Size           : 64 bits
EncTransportPDU       :               63626c19d63a97a453
NetMIC                :                                 0b1ac45439c38e90

Obfuscation

Privacy Random        : 63626c19d63a97
PECB                  : 2dd8f528b747cc51202cdcda6d668051
preObfuscation        : 801234560405

NetworkPDU            : 0dadcac17eb34263626c19d63a97a4530b1ac45439c38e90
```

### 8.11.6 PTREQ with all unset options

A Directed Forwarding node sends a Path Tracing Request (PTREQ) message as a Path Originator to validate a path. No PTREQ options are set.

```
Transport Control Message

Opcode                : 33 (Path Tracing Request)
  OPT.OBOO            : 00
  OPT                 : 00
  PD                  : 0607
  PO                  : 0405
  OPM                 : 00

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
```

```
SRC                     : 0405
DST                     : fffb
UpperTransportPDU       : 000607040500

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
SEG                     : 00
Opcode                  : 33
Header                  : 33
UpperTransportPDU       :   000607040500
LowerTransportPDU       : 33000607040500

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
LowerTransportPDU       : 33000607040500
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00801234560405000012345678
IVI NID                 : 0d
CTL TTL                 :   80
SEQ                     :     123456
SRC                     :           0405
DST                     :               fffb
TransportPDU            :                   33000607040500
NetMIC Size             : 64 bits
EncTransportPDU         :               8ead6d19d63a97a45f
NetMIC                  :                                 dee573a569c18fc6

Obfuscation

Privacy Random          : 8ead6d19d63a97
PECB                    : b1707d7134d0c75d2b00d9a66d3a5c7e
```

```
preObfuscation          : 801234560405

NetworkPDU              : 0d3162492730d58ead6d19d63a97a45fdee573a569c18fc6
```

### 8.11.7 PTREQ with all set options

A Directed Forwarding node sends a Path Tracing Request (PTREQ) message as a Delegate Path Originator to validate a path. All PREQ options are set.

```
Transport Control Message

Opcode                  : 33 (Path Tracing Request)
  OPT.OBOO              : 01
  OPT                   : 80
  PD                    : 0607
  PO                    : 0405
  OPM                   : 00
  DPO                   : 0777

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
UpperTransportPDU       : 8006070405000777

LowerTransportUnsegmentedControlPDU

CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
SEG                     : 00
Opcode                  : 33
Header                  : 33
UpperTransportPDU       :  8006070405000777
LowerTransportPDU       : 338006070405000777

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
```

```
TTL                     : 00
SEQ                     : 123456
SRC                     : 0405
DST                     : fffb
LowerTransportPDU       : 338006070405000777
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00801234560405000012345678
IVI NID                 : 0d
CTL TTL                 :   80
SEQ                     :     123456
SRC                     :           0405
DST                     :               fffb
TransportPDU            :                   338006070405000777
NetMIC Size             : 64 bits
EncTransportPDU         :               8ead6d99d63a97a45f3199
NetMIC                  :                                     eef6eb0b202065be

Obfuscation

Privacy Random          : 8ead6d99d63a97
PECB                    : 6dcecb48c3bf9eabe6eba36d866c1fb8
preObfuscation          : 801234560405

NetworkPDU              : 0deddcff1ec7ba8ead6d99d63a97a45f3199eef6eb0b202065be
```

### 8.11.8 PTREP

A Directed Forwarding node sends a Path Tracing Reply (PTREP) message as a Path Destination to reply to a Path Tracking Request (PTREQ) message.

```
Transport Control Message

Opcode                  : 34 (Path Tracing Reply)
  OPT.OBOO              : 00
  OPT                   : 00
  PD                    : 0607
  PO                    : 0405
  OPM                   : 02

UpperTransportControlPDU

TTL                     : 00
SEQ                     : 123456
```

```
SRC                   : 0607
DST                   : 1234
UpperTransportPDU     : 000607040502

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : 1234
SEG                   : 00
Opcode                : 34
Header                : 34
UpperTransportPDU     :   000607040502
LowerTransportPDU     : 34000607040502

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : 1234
LowerTransportPDU     : 34000607040502
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00801234560607000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123456
SRC                   :           0607
DST                   :               1234
TransportPDU          :                   34000607040502
NetMIC Size           : 64 bits
EncTransportPDU       :               85e268e309546a5ef7
NetMIC                :                                 07a11371d2bce964

Obfuscation

Privacy Random        : 85e268e309546a
PECB                  : 9d3b42d169eea1fec07e294dd7f342b1
```

```
preObfuscation         : 801234560607

NetworkPDU             : 0d1d2976876fe985e268e309546a5ef707a11371d2bce964
```

### 8.11.9 PAREQ with all unset options

A Directed Forwarding node sends a Path Assistant Requst (PAREQ) message to one of its neighbors to request assistance for path forwarding. No PAREQ options are set except the DIR option.

```
Transport Control Message

Opcode                 : 35 (Path Assistant Request)
  OPT.DIR              : 01
  OPT.OBOO             : 00
  OPT.OBOD             : 00
  OPT.OEA              : 00
  OPT.DEA              : 00
  OPT.SL               : 00
  OPT.BPV              : 00
  OPT                  : 80
  PD                   : 0002
  PO                   : 0001
  FN                   : a0
  PN                   : 0809
  BID                  : 01
  Path Remaining Time  : c003

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
UpperTransportPDU      : 8000020001a0080901c003

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
SEG                    : 00
Opcode                 : 35
Header                 : 35
```

```
UpperTransportPDU      :   8000020001a0080901c003
LowerTransportPDU      : 358000020001a0080901c003

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 358000020001a0080901c003
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00801234560405000012345678
IVI NID                : 0d
CTL TTL                :   80
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   358000020001a0080901c003
NetMIC Size            : 64 bits
EncTransportPDU        :               77516b99d03f93a0ff3ee7c9368f
NetMIC                 :                                           ae8dc951200fbada

Obfuscation

Privacy Random         : 77516b99d03f93
PECB                   : 0376bb1ff8f9309b41ad988f34eb321d
preObfuscation         : 801234560405

NetworkPDU             : 0d83648f49fcfc77516b99d03f93a0ff3ee7c9368fae8dc951200fbada
```

### 8.11.10 PAREQ with all set options

A Directed Forwarding node sends a Path Assistant Request (PAREQ) message to one of its neighbors to request assistance for path forwarding. PAREQ options are all set. The message is sent in three segments.

```
Transport Control Message

Opcode                 : 35 (Path Assistant Request)
  OPT.DIR              : 01
```

```
  OPT.OBOO            : 01
  OPT.OBOD            : 01
  OPT.OEA             : 01
  OPT.DEA             : 01
  OPT.SL              : 01
  OPT.BPV             : 01
  OPT                 : fe
  PD                  : 0020
  PO                  : 0010
  FN                  : a0
  PN                  : 0809
  BID                 : 01
  Path Remaining Time : c003
  DPO                 : 00a0
  DPD                 : 00b0
  OAR                 : 01
  DAR                 : 02
  SubscriptionList    : c000c001

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
UpperTransportPDU     : fe00200010a0080901c00300a000b00102c000c001
Segment#0             : fe00200010a00809
Segment#1             : 01c00300a000b001
Segment#2             : 02c000c001

LowerTransportSegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 01
Opcode                : 35
SeqZero               : 1456
SegO                  : 00
SegN                  : 02
Header                : b5515802
Segment#0             :         fe00200010a00809
LowerTransportPDU     : b5515802fe00200010a00809
```

```
NetworkPDU

IVindex              : 12345678
NetKey               : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                  : 01
TTL                  : 00
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
LowerTransportPDU    : b5515802fe00200010a00809
NID                  : 0d
EncryptionKey        : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey           : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce        : 00801234560405000012345678
IVI NID              : 0d
CTL TTL              :   80
SEQ                  :     123456
SRC                  :           0405
DST                  :               0607
TransportPDU         :                   b5515802fe00200010a00809
NetMIC Size          : 64 bits
EncTransportPDU      :               7751eb48883f6da17f36fe68fe85
NetMIC               :                                           ff193a7387be7252

Obfuscation

Privacy Random       : 7751eb48883f6d
PECB                 : 0623a0347f25afbcf32ab266e9286760
preObfuscation       : 801234560405

NetworkPDU           : 0d863194627b207751eb48883f6da17f36fe68fe85ff193a7387be7252

LowerTransportSegmentedControlPDU

CTL                  : 01
TTL                  : 00
SEQ                  : 123457
SRC                  : 0405
DST                  : 0607
SEG                  : 01
Opcode               : 35
SeqZero              : 1456
SegO                 : 01
SegN                 : 02
```

```
Header                   : b5515822
Segment#1                :         01c00300a000b001
LowerTransportPDU        : b551582201c00300a000b001

NetworkPDU

IVindex                  : 12345678
NetKey                   : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                      : 01
TTL                      : 00
SEQ                      : 123457
SRC                      : 0405
DST                      : 0607
LowerTransportPDU        : b551582201c00300a000b001
NID                      : 0d
EncryptionKey            : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey               : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce            : 00801234570405000012345678
IVI NID                  : 0d
CTL TTL                  :   80
SEQ                      :     123457
SRC                      :           0405
DST                      :               0607
TransportPDU             :                   b551582201c00300a000b001
NetMIC Size              : 64 bits
EncTransportPDU          :               c61f0b0185d31516cfb0f271a884
NetMIC                   :                                           58a7b9d296e39d7c

Obfuscation

Privacy Random           : c61f0b0185d315
PECB                     : 9e6687083f81777a2e24929161128f55
preObfuscation           : 801234570405

NetworkPDU               : 0d1e74b35f3b84c61f0b0185d31516cfb0f271a88458a7b9d296e39d7c

LowerTransportSegmentedControlPDU

CTL                      : 01
TTL                      : 00
SEQ                      : 123458
SRC                      : 0405
DST                      : 0607
SEG                      : 01
Opcode                   : 35
```

```
SeqZero               : 1456
SegO                  : 02
SegN                  : 02
Header                : b5515842
Segment#2             :         02c000c001
LowerTransportPDU     : b551584202c000c001

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123458
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : b551584202c000c001
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00801234580405000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123458
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   b551584202c000c001
NetMIC Size           : 64 bits
EncTransportPDU       :               ee3c046a86a4328ceffec4
NetMIC                :                                     d51aee9ad83c271b

Obfuscation

Privacy Random        : ee3c046a86a432
PECB                  : d5a7d073d636ca655cef4c599a7f4518
preObfuscation        : 801234580405

NetworkPDU            : 0d55b5e42bd233ee3c046a86a4328ceffec4d51aee9ad83c271b
```

### 8.11.11 PAREP

A Directed Forwarding node sends a Path Assistant Reply (PAREP) message to reply to a Path Assistant Request (PAREQ) and confirm that it can provide assistance for path forwarding.

```
Transport Control Message
```

```
Opcode                : 36 (Path Assistant Reply)
  Status              : 00
  OPT.OBOO            : 00
  OPT                 : 00
  PD                  : 0002
  PO                  : 0001
  FN                  : a0
  PAI                 : 0405
  PN                  : 0809

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : fffb
UpperTransportPDU     : 000000020001a004050809

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : fffb
SEG                   : 00
Opcode                : 36
Header                : 36
UpperTransportPDU     :   000000020001a004050809
LowerTransportPDU     : 36000000020001a004050809

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : fffb
LowerTransportPDU     : 36000000020001a004050809
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
```

```
Network Nonce         : 00801234560607000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123456
SRC                   :           0607
DST                   :               fffb
TransportPDU          :                   36000000020001a004050809
NetMIC Size           : 64 bits
EncTransportPDU       :               682d6ae30f536c5bf4459448ae4f
NetMIC                :                                           670aa28316c25e9a

Obfuscation

Privacy Random        : 682d6ae30f536c
PECB                  : a597a3cef56c235b55b6c2cf73549960
preObfuscation        : 801234560607

NetworkPDU            : 0d25859798f36b682d6ae30f536c5bf4459448ae4f670aa28316c25e9a
```

### 8.11.12 PERR with all unset options

A Directed Forwarding node sends a Path Error (PERR) message as an intermediate node of a path to notify other nodes that the path is invalid. No PERR options are set.

```
Transport Control Message

Opcode                : 37 (Path Error)
  OPT.WPO             : 00
  OPT.OBOO            : 00
  OPT                 : 00
  PD                  : 0607
  FN                  : 0c

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 1234
DST                   : fffb
UpperTransportPDU     : 0006070c

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
```

```
SEQ                     : 123456
SRC                     : 1234
DST                     : fffb
SEG                     : 00
Opcode                  : 37
Header                  : 37
UpperTransportPDU       :   0006070c
LowerTransportPDU       : 370006070c

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 00
SEQ                     : 123456
SRC                     : 1234
DST                     : fffb
LowerTransportPDU       : 370006070c
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00801234561234000012345678
IVI NID                 : 0d
CTL TTL                 :   80
SEQ                     :     123456
SRC                     :           1234
DST                     :               fffb
TransportPDU            :                   370006070c
NetMIC Size             : 64 bits
EncTransportPDU         :               c2f257199f1477
NetMIC                  :                             015a9a2433012282

Obfuscation

Privacy Random          : c2f257199f1477
PECB                    : 4f1201f111bdffdf1d065921d77cf29f
preObfuscation          : 801234561234

NetworkPDU              : 0dcf0035a70389c2f257199f1477015a9a2433012282
```

### 8.11.13 PERR with all set options

A Directed Forwarding node sends a Path Error (PERR) message as an intermediate node of a path to notify other nodes that the path is invalid. All PERR options are set.

```
Transport Control Message

Opcode                : 37 (Path Error)
  OPT.WPO             : 01
  OPT.OBOO            : 01
  OPT                 : c0
  PD                  : 0607
  FN                  : 0c
  PO                  : 0405
  DPO                 : 0777

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 1234
DST                   : fffb
UpperTransportPDU     : c006070c04050777

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 1234
DST                   : fffb
SEG                   : 00
Opcode                : 37
Header                : 37
UpperTransportPDU     :   c006070c04050777
LowerTransportPDU     : 37c006070c04050777

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 1234
DST                   : fffb
LowerTransportPDU     : 37c006070c04050777
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
```

```
Network Nonce          : 00801234561234000012345678
IVI NID                : 0d
CTL TTL                :   80
SEQ                    :     123456
SRC                    :           1234
DST                    :               fffb
TransportPDU           :                   37c006070c04050777
NetMIC Size            : 64 bits
EncTransportPDU        :               c2f257d99f147704c70c8a
NetMIC                 :                                     9460b0e06aaece97

Obfuscation

Privacy Random         : c2f257d99f1477
PECB                   : 9a48562b889dce428a0ba5fa4f27c52a
preObfuscation         : 801234561234

NetworkPDU             : 0d1a5a627d9aa9c2f257d99f147704c70c8a9460b0e06aaece97
```

### 8.11.14 NINFO with all unset options and no neighbor information

A Directed Forwarding node sends a Neighbor Information (NINFO) message to announce only its presence. No NINFO options are set.

```
Transport Control Message

Opcode                 : 38 (Neighbor Information)
  OPT.NEA              : 00
  OPT.NDEA             : 00
  OPT                  : 00
  NFN                  : 05
  Wanted RSSI          : 00

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : fffb
UpperTransportPDU      : 000500

LowerTransportUnsegmentedControlPDU

CTL                    : 01
TTL                    : 00
```

```
SEQ                   : 123456
SRC                   : 0405
DST                   : fffb
SEG                   : 00
Opcode                : 38
Header                : 38
UpperTransportPDU     :   000500
LowerTransportPDU     : 38000500

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : fffb
LowerTransportPDU     : 38000500
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00801234560405000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123456
SRC                   :           0405
DST                   :               fffb
TransportPDU          :                   38000500
NetMIC Size           : 64 bits
EncTransportPDU       :               8ead6619d53d
NetMIC                :                           07c2e7c985eafc4b

Obfuscation

Privacy Random        : 8ead6619d53d07
PECB                  : 8b71ede873b83817b534d8320ed0261f
preObfuscation        : 801234560405

NetworkPDU            : 0d0b63d9be77bd8ead6619d53d07c2e7c985eafc4b
```

### 8.11.15 NINFO with all set options and with neighbor information

A Directed Forwarding node sends a Neighbor Information (NINFO) message to announce its presence and its neighbors. All NINFO options are set. The message is sent in two segments.

```
Transport Control Message

Opcode                 : 38 (Neighbor Information)
  OPT.NEA              : 01
  OPT.NDEA             : 01
  OPT                  : 82
  NFN                  : 05
  Wanted RSSI          : 00
  NAR                  : 04
  Neighbor#0:
  PD                   : 1111
  DFN                  : 15
  DAR                  : 03
  Neighbor#1:
  PD                   : 1112
  DFN                  : 16
  Neighbor#2:
  PD                   : 1113
  DFN                  : 17

UpperTransportControlPDU

TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : fffb
UpperTransportPDU      : 8205000411111503111216111317
Segment#0              : 8205000411111503
Segment#1              : 111216111317

LowerTransportSegmentedControlPDU

CTL                    : 01
TTL                    : 00
SEQ                    : 123456
SRC                    : 0405
DST                    : fffb
SEG                    : 01
Opcode                 : 38
SeqZero                : 1456
SegO                   : 00
SegN                   : 01
Header                 : b8515801
Segment#0              :         8205000411111503
LowerTransportPDU      : b85158018205000411111503
```

```
NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0405
DST                   : fffb
LowerTransportPDU     : b8515801820500041111503
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00801234560405000012345678
IVI NID               : 0d
CTL TTL               :   80
SEQ                   :     123456
SRC                   :           0405
DST                   :               fffb
TransportPDU          :                   b8515801820500041111503
NetMIC Size           : 64 bits
EncTransportPDU       :               8eade648883c11a45f32ffd9e38f
NetMIC                :                                           05fcbc29e296760d

Obfuscation

Privacy Random        : 8eade648883c11
PECB                  : 24d44d9fab4553f9f28c265db6003910
preObfuscation        : 801234560405

NetworkPDU            : 0da4c679c9af408eade648883c11a45f32ffd9e38f05fcbc29e296760d

LowerTransportSegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123457
SRC                   : 0405
DST                   : fffb
SEG                   : 01
Opcode                : 38
SeqZero               : 1456
SegO                  : 01
SegN                  : 01
```

```
Header               : b8515821
Segment#1            :         111216111317
LowerTransportPDU    : b8515821111216111317

NetworkPDU

IVindex              : 12345678
NetKey               : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                  : 01
TTL                  : 00
SEQ                  : 123457
SRC                  : 0405
DST                  : fffb
LowerTransportPDU    : b8515821111216111317
NID                  : 0d
EncryptionKey        : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey           : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce        : 0080123457040500001234567 8
IVI NID              : 0d
CTL TTL              :   80
SEQ                  :     123457
SRC                  :           0405
DST                  :               fffb
TransportPDU         :                   b8515821111216111317
NetMIC Size          : 64 bits
EncTransportPDU      :               3fe3060185d005c4daa14166
NetMIC               :                                       7458f7d487f6a274

Obfuscation

Privacy Random       : 3fe3060185d005
PECB                 : cf2445112252cbed72de6bd48dfb21e0
preObfuscation       : 801234570405

NetworkPDU           : 0d4f36714626573fe3060185d005c4daa141667458f7d487f6a274
```

## 8.11.16 LREQ

A Directed Forwarding node sends a Link Quality Measurement Request (LREQ) message to one of its neighbors to perform link quality measurement.

```
Transport Control Message

Opcode               : 39 (Link Quality Measurement Request)
  Session ID         : ab
```

```
  PTX                    : 08
  Wanted RSSI            : b0

UpperTransportControlPDU

TTL                      : 00
SEQ                      : 123456
SRC                      : 0405
DST                      : 0607
UpperTransportPDU        : ab08b0

LowerTransportUnsegmentedControlPDU

CTL                      : 01
TTL                      : 00
SEQ                      : 123456
SRC                      : 0405
DST                      : 0607
SEG                      : 00
Opcode                   : 39
Header                   : 39
UpperTransportPDU        :   ab08b0
LowerTransportPDU        : 39ab08b0

NetworkPDU

IVindex                  : 12345678
NetKey                   : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                      : 01
TTL                      : 00
SEQ                      : 123456
SRC                      : 0405
DST                      : 0607
LowerTransportPDU        : 39ab08b0
NID                      : 0d
EncryptionKey            : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey               : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce            : 00801234560405000012345678
IVI NID                  : 0d
CTL TTL                  :   80
SEQ                      :     123456
SRC                      :           0405
DST                      :               0607
TransportPDU             :                   39ab08b0
NetMIC Size              : 64 bits
```

```
EncTransportPDU       :              775167b2d88d
NetMIC                :                          cf7acac7e96aa521

Obfuscation

Privacy Random        : 775167b2d88dcf
PECB                  : 50e56ce06a3ee0dd3f92e25b2b170c3e
preObfuscation        : 801234560405

NetworkPDU            : 0dd0f758b66e3b775167b2d88dcf7acac7e96aa521
```

### 8.11.17 LREP

A Directed Forwarding node sends a Link Quality Measurement Reply (LREP) message to reply to a Link Quality Measurement Request (LREQ).

```
Transport Control Message

Opcode                : 3a (Link Quality Measurement Reply)
  Session ID          : ab
  PTX                 : 08
  Wanted RSSI         : b0
  Measured RSSI       : b0

UpperTransportControlPDU

TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : 0405
UpperTransportPDU     : ab08b0b0

LowerTransportUnsegmentedControlPDU

CTL                   : 01
TTL                   : 00
SEQ                   : 123456
SRC                   : 0607
DST                   : 0405
SEG                   : 00
Opcode                : 3a
Header                : 3a
UpperTransportPDU     :   ab08b0b0
LowerTransportPDU     : 3aab08b0b0
```

```
NetworkPDU

IVindex              : 12345678
NetKey               : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                  : 01
TTL                  : 00
SEQ                  : 123456
SRC                  : 0607
DST                  : 0405
LowerTransportPDU    : 3aab08b0b0
NID                  : 0d
EncryptionKey        : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey           : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce        : 00801234560607000012345678
IVI NID              : 0d
CTL TTL              :   80
SEQ                  :     123456
SRC                  :           0607
DST                  :               0405
TransportPDU         :                   3aab08b0b0
NetMIC Size          : 64 bits
EncTransportPDU      :               93d3664807e3de
NetMIC               :                             6e2042acb5e0e91b

Obfuscation

Privacy Random       : 93d3664807e3de
PECB                 : 3ecdf97d2e2464735ccc23d89a4acd26
preObfuscation       : 801234560607

NetworkPDU           : 0dbedfcd2b282393d3664807e3de6e2042acb5e0e91b
```

### 8.11.18 DF Directed Forwarding Get

A Directed Forwarding Client sends a DF Directed Forwarding Get message to a Directed Forwarding Server to read its Directed Forwarding state.

```
Access Payload

Opcode               : bf30 (DF Directed Forwarding Get)
Access Payload       : bf30

UpperTransportAccessPDU

IVindex              : 12345678
```

```
DevKey                    : 63964771734fbd76e3b40519d1d94a48
TTL                       : 07
SEQ                       : 123456
SRC                       : 0405
DST                       : 0607
Device Nonce              : 02001234560405060712345678
EncAccessPayload          : 232c
TransMIC Size             : 32 bits
TransMIC                  : a53feafb
UpperTransportPDU         : 232ca53feafb

LowerTransportUnsegmentedAccessPDU

CTL                       : 01
TTL                       : 07
SEQ                       : 123456
SRC                       : 0405
DST                       : 0607
SEG                       : 00
AKF                       : 00
AID                       : 00
Header                    : 00
UpperTransportPDU         :   232ca53feafb
LowerTransportPDU         : 00232ca53feafb

NetworkPDU

IVindex                   : 12345678
NetKey                    : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                       : 01
TTL                       : 07
SEQ                       : 123456
SRC                       : 0405
DST                       : 0607
LowerTransportPDU         : 00232ca53feafb
NID                       : 0d
EncryptionKey             : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey                : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce             : 00871234560405000012345678
IVI NID                   : 0d
CTL TTL                   :   87
SEQ                       :     123456
SRC                       :           0405
DST                       :               0607
TransportPDU              :                   00232ca53feafb
```

```
NetMIC Size           : 32 bits
EncTransportPDU       :              ee9639026b103d680d
NetMIC                :                                  1abf0e07

Obfuscation

Privacy Random        : ee9639026b103d
PECB                  : ddc1340eafdd19d80477ffe3b78ecf7f
preObfuscation        : 871234560405

NetworkPDU            : 0d5ad30058abd8ee9639026b103d680d1abf0e07
```

### 8.11.19 DF Directed Forwarding Set

A Directed Forwarding Client sends a DF Directed Forwarding Set message to a Directed Forwarding Server to enable the Directed Forwarding feature.

```
Access Payload

Opcode                : bf31 (DF Directed Forwarding Set)
Directed Forwarding   : 01 (enabled)
Access Payload        : bf3101

UpperTransportAccessPDU

IVindex               : 12345678
DevKey                : 63964771734fbd76e3b40519d1d94a48
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
Device Nonce          : 02001234560405060712345678
EncAccessPayload      : 232df1
TransMIC Size         : 32 bits
TransMIC              : 547138a2
UpperTransportPDU     : 232df1547138a2

LowerTransportUnsegmentedAccessPDU

CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 00
```

```
AKF                    : 00
AID                    : 00
Header                 : 00
UpperTransportPDU      :   232df1547138a2
LowerTransportPDU      : 00232df1547138a2

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 00232df1547138a2
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   00232df1547138a2
NetMIC Size            : 32 bits
EncTransportPDU        :               ee9639026a4456f3cefe
NetMIC                 :                                   3a552a8d

Obfuscation

Privacy Random         : ee9639026a4456
PECB                   : c1748f0fcc11898f663d7e424f84b64f
preObfuscation         : 871234560405

NetworkPDU             : 0d4666bb59c814ee9639026a4456f3cefe3a552a8d
```

### 8.11.20 DF Directed Forwarding Status

A Directed Forwarding Server receives a DF Directed Forwarding Get message or a DF Directed Forwarding Set message from a Directed Forwarding Client and responds with a DF Directed Forwarding Status message.

```
Access Payload
```

```
Opcode                  : bf32 (DF Directed Forwarding Status)
Directed Forwarding     : 01 (enabled)
Access Payload          : bf3201

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02001234560405060712345678
EncAccessPayload        : 232ef1
TransMIC Size           : 32 bits
TransMIC                : 8e1a7801
UpperTransportPDU       : 232ef18e1a7801

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   232ef18e1a7801
LowerTransportPDU       : 00232ef18e1a7801

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
LowerTransportPDU       : 00232ef18e1a7801
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
```

```
PrivacyKey           : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce        : 00871234560405000012345678
IVI NID              : 0d
CTL TTL              :   87
SEQ                  :     123456
SRC                  :           0405
DST                  :               0607
TransportPDU         :                   00232ef18e1a7801
NetMIC Size          : 32 bits
EncTransportPDU      :               ee96390269448c988e5d
NetMIC               :                                   c45465a3

Obfuscation

Privacy Random       : ee96390269448c
PECB                 : a0ab9c89b957d9b1932a064706f587a3
preObfuscation       : 871234560405

NetworkPDU           : 0d27b9a8dfbd52ee96390269448c988e5dc45465a3
```

### 8.11.21 DF Path Metric Get

A Directed Forwarding Client sends a DF Path Metric Get message to a Directed Forwarding Server.

```
Access Payload

Opcode               : bf33 (DF Path Metric Get)
Access Payload       : bf33

UpperTransportAccessPDU

IVindex              : 12345678
DevKey               : 63964771734fbd76e3b40519d1d94a48
TTL                  : 07
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
Device Nonce         : 02001234560405060712345678
EncAccessPayload     : 232f
TransMIC Size        : 32 bits
TransMIC             : aabc8c70
UpperTransportPDU    : 232faabc8c70

LowerTransportUnsegmentedAccessPDU
```

```
CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 00
AKF                   : 00
AID                   : 00
Header                : 00
UpperTransportPDU     :   232faabc8c70
LowerTransportPDU     : 00232faabc8c70

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : 00232faabc8c70
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00871234560405000012345678
IVI NID               : 0d
CTL TTL               :   87
SEQ                   :     123456
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   00232faabc8c70
NetMIC Size           : 32 bits
EncTransportPDU       :               ee963902681fbe0e86
NetMIC                :                                 5f3b84c0

Obfuscation

Privacy Random        : ee963902681fbe
PECB                  : 5738949897c0ad7182109b0b24f8d1a0
preObfuscation        : 871234560405

NetworkPDU            : 0dd02aa0ce93c5ee963902681fbe0e865f3b84c0
```

### 8.11.22 DF Path Metric Set

A Directed Forwarding Client sends a DF Path Metric Set message to a Directed Forwarding Server to set the hop-count path metric with average RSSI-based path selection and set the path lifetime to 12 hours.

```
Access Payload

Opcode                  : bf34 (DF Path Metric Set)
Path Metric Type        : 01 (hop-count with average RSSI-based path selection)
Path Lifetime           : 800c (12 hours)
Access Payload          : bf34010c80

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02001234560405060712345678
EncAccessPayload        : 2328f1340b
TransMIC Size           : 32 bits
TransMIC                : c811f911
UpperTransportPDU       : 2328f1340bc811f911

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   2328f1340bc811f911
LowerTransportPDU       : 002328f1340bc811f911

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
```

```
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : 002328f1340bc811f911
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00871234560405000012345678
IVI NID               : 0d
CTL TTL               :   87
SEQ                   :     123456
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   002328f1340bc811f911
NetMIC Size           : 32 bits
EncTransportPDU       :               ee9639026f4436893e4da4b1
NetMIC                :                                       933c285a

Obfuscation

Privacy Random        : ee9639026f4436
PECB                  : db7672e671a0bd6b7fcefae6c2f3b2dd
preObfuscation        : 871234560405

NetworkPDU            : 0d5c6446b075a5ee9639026f4436893e4da4b1933c285a
```

### 8.11.23 DF Path Metric Status

A Directed Forwarding Server receives a DF Path Metric Get message or a DF Path Metric Set message from a Directed Forwarding Client and responds with a DF Path Metric Status message.

```
Access Payload

Opcode                : bf35 (DF Path Metric Status)
Path Metric Type      : 01 (hop-count with average RSSI-based path selection)
Path Lifetime         : 800c (12 hours)
Access Payload        : bf35010c80

UpperTransportAccessPDU

IVindex               : 12345678
DevKey                : 63964771734fbd76e3b40519d1d94a48
TTL                   : 07
SEQ                   : 123456
```

```
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02001234560405060712345678
EncAccessPayload        : 2329f1340b
TransMIC Size           : 32 bits
TransMIC                : 3f016f64
UpperTransportPDU       : 2329f1340b3f016f64

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   2329f1340b3f016f64
LowerTransportPDU       : 002329f1340b3f016f64

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
LowerTransportPDU       : 002329f1340b3f016f64
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00871234560405000012345678
IVI NID                 : 0d
CTL TTL                 :   87
SEQ                     :     123456
SRC                     :           0405
DST                     :               0607
TransportPDU            :                   002329f1340b3f016f64
NetMIC Size             : 32 bits
EncTransportPDU         :               ee9639026e443689c95d32c4
NetMIC                  :                                       3f6192bc
```

```
Obfuscation

Privacy Random          : ee9639026e4436
PECB                    : 257c8474f3a6a4909d778515c472d2a8
preObfuscation          : 871234560405

NetworkPDU              : 0da26eb022f7a3ee9639026e443689c95d32c43f6192bc
```

### 8.11.24 DF Neighbor Information Get

A Directed Forwarding Client sends a DF Neighbor Information Get message to a Directed Forwarding Server.

```
Access Payload

Opcode                  : bf36 (DF Neighbor Information Get)
Access Payload          : bf36

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 0200123456040506071234567 8
EncAccessPayload        : 232a
TransMIC Size           : 32 bits
TransMIC                : 303fc435
UpperTransportPDU       : 232a303fc435

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   232a303fc435
```

```
LowerTransportPDU      : 00232a303fc435

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 00232a303fc435
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   00232a303fc435
NetMIC Size            : 32 bits
EncTransportPDU        :               ee9639026d853d46c3
NetMIC                 :                                 c47dea2b

Obfuscation

Privacy Random         : ee9639026d853d
PECB                   : f8967cd1e4bbff22bfb06908205c3b5d
preObfuscation         : 871234560405

NetworkPDU             : 0d7f844887e0beee9639026d853d46c3c47dea2b
```

### 8.11.25 DF Neighbor Information Set

A Directed Forwarding Client sends a DF Neighbor Information Set message to a Directed Forwarding Server to extend the neighborhood by as many asto 2 hops and to allow transmission and processing of NINFO for proactive path establishment.

```
Access Payload

Opcode                 : bf37 (DF Neighbor Information Set)
Neighborhood           : 02 (hops)
Neighbor Info. Options: 03 (NINFO tx enabled and NINFO processing at rx enabled)
```

```
Access Payload          : bf370203

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02000123456040506071234567
EncAccessPayload        : 232bf23b
TransMIC Size           : 32 bits
TransMIC                : 3f5b44cf
UpperTransportPDU       : 232bf23b3f5b44cf

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :  232bf23b3f5b44cf
LowerTransportPDU       : 00232bf23b3f5b44cf

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
LowerTransportPDU       : 00232bf23b3f5b44cf
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00871234560405000012345678
IVI NID                 : 0d
```

```
CTL TTL                : 87
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
TransportPDU           : 00232bf23b3f5b44cf
NetMIC Size            : 32 bits
EncTransportPDU        : ee9639026c4739bdad1892
NetMIC                 : c1c09c43

Obfuscation

Privacy Random         : ee9639026c4739
PECB                   : fa596a370c423bbe5fe564598d011cfa
preObfuscation         : 871234560405

NetworkPDU             : 0d7d4b5e610847ee9639026c4739bdad1892c1c09c43
```

### 8.11.26 DF Neighbor Information Status

A Directed Forwarding Server receives a DF Neighbor Information Get message or a DF Neighbor Information Set message from a Directed Forwarding Client and responds with a DF Neighbor Information Status message.

```
Access Payload

Opcode                 : bf38 (DF Neighbor Information Status)
Neighborhood           : 02
Neighbor Info. Options: 03 (NINFO tx enabled and NINFO processing at rx enabled)
Access Payload         : bf380203

UpperTransportAccessPDU

IVindex                : 12345678
DevKey                 : 63964771734fbd76e3b40519d1d94a48
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
Device Nonce           : 02001234560405060712345678
EncAccessPayload       : 2324f23b
TransMIC Size          : 32 bits
TransMIC               : 0276d204
UpperTransportPDU      : 2324f23b0276d204

LowerTransportUnsegmentedAccessPDU
```

```
CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 00
AKF                   : 00
AID                   : 00
Header                : 00
UpperTransportPDU     :   2324f23b0276d204
LowerTransportPDU     : 002324f23b0276d204

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : 002324f23b0276d204
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00871234560405000012345678
IVI NID               : 0d
CTL TTL               :   87
SEQ                   :     123456
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   002324f23b0276d204
NetMIC Size           : 32 bits
EncTransportPDU       :               ee96390263473980808e59
NetMIC                :                                     a4271ce4

Obfuscation

Privacy Random        : ee963902634739
PECB                  : 9a4852f2d2f404647c5871082547feed
preObfuscation        : 871234560405

NetworkPDU            : 0d1d5a66a4d6f1ee96390263473980808e59a4271ce4
```

### 8.11.27 DF Forwarding Table Add

A Directed Forwarding Client sends a DF Forwarding Table Add message with an empty Subscription List field to a Directed Forwarding Server.

```
Access Payload

Opcode                  : bf39 (DF Forwarding Table Add)
PO                      : 1234
OAR                     : 02
PD                      : 5678
DAR                     : 04
BTO                     : 01
BTD                     : 01
Subscription List       :
Access Payload          : bf393412027856040101

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02001234560405060712345678
EncAccessPayload        : 2325c42a89468a1d7e1c
TransMIC Size           : 32 bits
TransMIC                : 85509aa7
UpperTransportPDU       : 2325c42a89468a1d7e1c85509aa7

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   2325c42a89468a1d7e1c85509aa7
LowerTransportPDU       : 002325c42a89468a1d7e1c85509aa7
```

```
NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 002325c42a89468a1d7e1c85509aa7
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   002325c42a89468a1d7e1c85509aa7
NetMIC Size            : 32 bits
EncTransportPDU        :               ee9639026271280bb0d640de49535bdda6
NetMIC                 :                                                 1a7799b9

Obfuscation

Privacy Random         : ee963902627128
PECB                   : 0bb7aa5cdca28dadb752145f83c8bb01
preObfuscation         : 871234560405

NetworkPDU             : 0d8ca59e0ad8a7ee9639026271280bb0d640de49535bdda61a7799b9
```

### 8.11.28 DF Forwarding Table Delete

A Directed Forwarding Client sends a DF Forwarding Table Delete message to a Directed Forwarding Server.

```
Access Payload

Opcode                 : bf3b (DF Forwarding Table Delete)
PO                     : 1234
DPO                    : 0000
PD                     : 5678
DPD                    : 0000
Access Payload         : bf3b3412000078560000
```

```
UpperTransportAccessPDU

IVindex              : 12345678
DevKey               : 63964771734fbd76e3b40519d1d94a48
TTL                  : 07
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
Device Nonce         : 02001234560405060712345678
EncAccessPayload     : 2327c42a8b3ea44f7f1d
TransMIC Size        : 32 bits
TransMIC             : c0e22088
UpperTransportPDU    : 2327c42a8b3ea44f7f1dc0e22088

LowerTransportUnsegmentedAccessPDU

CTL                  : 01
TTL                  : 07
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
SEG                  : 00
AKF                  : 00
AID                  : 00
Header               : 00
UpperTransportPDU    :   2327c42a8b3ea44f7f1dc0e22088
LowerTransportPDU    : 002327c42a8b3ea44f7f1dc0e22088

NetworkPDU

IVindex              : 12345678
NetKey               : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                  : 01
TTL                  : 07
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
LowerTransportPDU    : 002327c42a8b3ea44f7f1dc0e22088
NID                  : 0d
EncryptionKey        : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey           : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce        : 00871234560405000012345678
IVI NID              : 0d
CTL TTL              :   87
```

```
SEQ                 :     123456
SRC                 :           0405
DST                 :               0607
TransportPDU        :                   002327c42a8b3ea44f7f1dc0e22088
NetMIC Size         : 32 bits
EncTransportPDU     :               ee96390260712809c8f812df4816e96789
NetMIC              :                                               f2818309

Obfuscation

Privacy Random      : ee963902607128
PECB                : 01a935f5bbb458d29531e29c79637cf8
preObfuscation      : 871234560405

NetworkPDU          : 0d86bb01a3bfb1ee96390260712809c8f812df4816e96789f2818309
```

### 8.11.29 DF Forwarding Table Status

A Directed Forwarding Server receives a DF Forwarding Table Add message or a DF Forwarding Table Delete message from a Directed Forwarding Client and responds with a DF Forwarding Table Status message with Status equal to Success. The message is sent in two segments.

```
Access Payload

Opcode              : bf3c (DF Forwarding Table Status)
Status              : 00 (Success)
PO                  : 1234
OAR                 : 02
DPO                 : 0000
PD                  : 5678
DAR                 : 04
DPD                 : 0000
Access Payload      : bf3c00341202000078560400000

UpperTransportAccessPDU

IVindex             : 12345678
DevKey              : 63964771734fbd76e3b40519d1d94a48
TTL                 : 07
SEQ                 : 123456
SRC                 : 0405
DST                 : 0607
Device Nonce        : 02001234560405060712345678
EncAccessPayload    : 2320f00c993cdc19074ba70c2f
TransMIC Size       : 32 bits
```

```
TransMIC              : 7fa2c80b
UpperTransportPDU     : 2320f00c993cdc19074ba70c2f7fa2c80b
Segment#0             : 2320f00c993cdc19074ba70c
Segment#1             : 2f7fa2c80b

LowerTransportSegmentedAccessPDU

CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 01
AKF                   : 00
AID                   : 00
SZMIC                 : 00
SeqZero               : 1456
SegO                  : 00
SegN                  : 01
Header                : 80515801
Segment#0             :         2320f00c993cdc19074ba70c
LowerTransportPDU     : 805158012320f00c993cdc19074ba70c

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : 805158012320f00c993cdc19074ba70c
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00871234560405000012345678
IVI NID               : 0d
CTL TTL               :   87
SEQ                   :     123456
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   805158012320f00c993cdc19074ba70c
NetMIC Size           : 64 bits
EncTransportPDU       :               ee96b9701fb421a20650c49c89cf0c0ca60c
```

```
NetMIC                  :                                                   0f521b1a

Obfuscation

Privacy Random          : ee96b9701fb421
PECB                    : a6dce667c7f5765202841542b1e4cdc0
preObfuscation          : 871234560405

NetworkPDU              : 0d21ced231c3f0ee96b9701fb421a20650c49c89cf0c0ca60c0f521b1a

LowerTransportSegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123457
SRC                     : 0405
DST                     : 0607
SEG                     : 01
AKF                     : 00
AID                     : 00
SZMIC                   : 00
SeqZero                 : 1456
SegO                    : 01
SegN                    : 01
Header                  : 80515821
Segment#1               :         2f7fa2c80b
LowerTransportPDU       : 805158212f7fa2c80b

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123457
SRC                     : 0405
DST                     : 0607
LowerTransportPDU       : 805158212f7fa2c80b
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00871234570405000012345678
IVI NID                 : 0d
CTL TTL                 :   87
SEQ                     :     123457
```

```
SRC                  :          0405
DST                  :              0607
TransportPDU         :                  805158212f7fa2c80b
NetMIC Size          : 64 bits
EncTransportPDU      :              ee4074612546aaf4644df0
NetMIC               :                                  fa0f187e

Obfuscation

Privacy Random       : ee4074612546aa
PECB                 : 5b9cfa53a50d2299e8506560e7ac36ac
preObfuscation       : 871234570405

NetworkPDU           : 0ddc8ece04a108ee4074612546aaf4644df0fa0f187e
```

### 8.11.30 DF Forwarding Table Get

A Directed Forwarding Client sends a DF Forwarding Table Get message to a Directed Forwarding Server.

```
Access Payload

Opcode               : bf3d (DF Forwarding Table Get)
FirstIndex           : 0000
IndexRange           : ffff
Access Payload       : bf3d0000ffff

UpperTransportAccessPDU

IVindex              : 12345678
DevKey               : 63964771734fbd76e3b40519d1d94a48
TTL                  : 07
SEQ                  : 123456
SRC                  : 0405
DST                  : 0607
Device Nonce         : 02001234560405060712345678
EncAccessPayload     : 2321f03874c1
TransMIC Size        : 32 bits
TransMIC             : 8b49e314
UpperTransportPDU    : 2321f03874c18b49e314

LowerTransportUnsegmentedAccessPDU

CTL                  : 01
TTL                  : 07
```

```
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
SEG                    : 00
AKF                    : 00
AID                    : 00
Header                 : 00
UpperTransportPDU      :   2321f03874c18b49e314
LowerTransportPDU      : 002321f03874c18b49e314

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 002321f03874c18b49e314
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   002321f03874c18b49e314
NetMIC Size            : 32 bits
EncTransportPDU        :               ee96390266453af637d7144341
NetMIC                 :                                         6d510cde

Obfuscation

Privacy Random         : ee96390266453a
PECB                   : 0d7646450c5f2236d98df1a2d32d201f
preObfuscation         : 871234560405

NetworkPDU             : 0d8a647213085aee96390266453af637d71443416d510cde
```

### 8.11.31 DF Forwarding Table List

A Directed Forwarding Server receives a DF Forwarding Table Get message from a Directed Forwarding Client and responds with a DF Forwarding Table List message that reports a single fixed path entry. The message is sent in two segments.

```
Access Payload

Opcode                 : bf3e (DF Forwarding Table List)
FirstIndex             : 0000
OptionalFieldMap       : 081a
Flags                  : 01 (fixed path entry)
PD                     : 5678
PO                     : 1234
DAR                    : 04
OAR                    : 02
Subscription List Size: 01
Subscription List      : c000
Access Payload         : bf3e00001a0801785634120402010 0c0

UpperTransportAccessPDU

IVindex                : 12345678
DevKey                 : 63964771734fbd76e3b40519d1d94a48
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
Device Nonce           : 02001234560405060712345678
EncAccessPayload       : 2322f0389136dd61291da3383d9ea10d
TransMIC Size          : 32 bits
TransMIC               : b0017012
UpperTransportPDU      : 2322f0389136dd61291da3383d9ea10db0017012
Segment#0              : 2322f0389136dd61291da338
Segment#1              : 3d9ea10db0017012

LowerTransportSegmentedAccessPDU

CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
SEG                    : 01
AKF                    : 00
```

```
AID                    : 00
SZMIC                  : 00
SeqZero                : 1456
SegO                   : 00
SegN                   : 01
Header                 : 80515801
Segment#0              :         2322f0389136dd61291da338
LowerTransportPDU      : 805158012322f0389136dd61291da338

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 805158012322f0389136dd61291da338
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   805158012322f0389136dd61291da338
NetMIC Size            : 64 bits
EncTransportPDU        :               ee96b9701fb421a00664cc9688b7225aa238
NetMIC                 :                                                   9a7b8f28

Obfuscation

Privacy Random         : ee96b9701fb421
PECB                   : a6dce667c7f5765202841542b1e4cdc0
preObfuscation         : 871234560405

NetworkPDU             : 0d21ced231c3f0ee96b9701fb421a00664cc9688b7225aa2389a7b8f28

LowerTransportSegmentedAccessPDU

CTL                    : 01
TTL                    : 07
```

```
SEQ                   : 123457
SRC                   : 0405
DST                   : 0607
SEG                   : 01
AKF                   : 00
AID                   : 00
SZMIC                 : 00
SeqZero               : 1456
SegO                  : 01
SegN                  : 01
Header                : 80515821
Segment#1             :         3d9ea10db0017012
LowerTransportPDU     : 805158213d9ea10db0017012

NetworkPDU

IVindex               : 12345678
NetKey                : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                   : 01
TTL                   : 07
SEQ                   : 123457
SRC                   : 0405
DST                   : 0607
LowerTransportPDU     : 805158213d9ea10db0017012
NID                   : 0d
EncryptionKey         : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey            : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce         : 00871234570405000012345678
IVI NID               : 0d
CTL TTL               :   87
SEQ                   :     123457
SRC                   :           0405
DST                   :               0607
TransportPDU          :                   805158213d9ea10db0017012
NetMIC Size           : 64 bits
EncTransportPDU       :               ee4074612546b81567884b6c8345
NetMIC                :                                           b91b6acc

Obfuscation

Privacy Random        : ee4074612546b8
PECB                  : 5fc320c2ecaa2fc9e27a84f72db04861
preObfuscation        : 871234570405

NetworkPDU            : 0dd8d11495e8afee4074612546b81567884b6c8345b91b6acc
```

### 8.11.32 DF Path Request Transmit Get

A Directed Forwarding Client sends a DF Path Request Transmit Get message to a Directed Forwarding Server.

```
Access Payload

Opcode                  : bf3f (DF Path Request Transmit Get)
Access Payload          : bf3f

UpperTransportAccessPDU

IVindex                 : 12345678
DevKey                  : 63964771734fbd76e3b40519d1d94a48
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
Device Nonce            : 02001234560405060712345678
EncAccessPayload        : 2323
TransMIC Size           : 32 bits
TransMIC                : dd320eb7
UpperTransportPDU       : 2323dd320eb7

LowerTransportUnsegmentedAccessPDU

CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
SEG                     : 00
AKF                     : 00
AID                     : 00
Header                  : 00
UpperTransportPDU       :   2323dd320eb7
LowerTransportPDU       : 002323dd320eb7

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123456
```

```
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 002323dd320eb7
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :       123456
SRC                    :             0405
DST                    :                 0607
TransportPDU           :                     002323dd320eb7
NetMIC Size            : 32 bits
EncTransportPDU        :                 ee9639026468308c41
NetMIC                 :                                   d6040dca

Obfuscation

Privacy Random         : ee963902646830
PECB                   : c5a68e1d625bd1f16f1921e559394619
preObfuscation         : 871234560405

NetworkPDU             : 0d42b4ba4b665eee9639026468308c41d6040dca
```

### 8.11.33 DF Path Request Transmit Set

A Directed Forwarding Client sends a DF Path Request Transmit Set message to a Directed Forwarding Server to set the maximum number of concurrent Directed Forwarding Initialization operations to 8 and to extend the minimum interval between two Directed Forwarding Initialization operations for the same Path Destination to 5 minutes.

```
Access Payload

Opcode                 : bf40 (DF Path Request Transmit Set)
Conc. Initialization   : 08
Path Req. Interval     : 01
Access Payload         : bf400801

UpperTransportAccessPDU

IVindex                : 12345678
DevKey                 : 63964771734fbd76e3b40519d1d94a48
TTL                    : 07
SEQ                    : 123456
```

```
SRC                    : 0405
DST                    : 0607
Device Nonce           : 02001234560405060712345678
EncAccessPayload       : 235cf839
TransMIC Size          : 32 bits
TransMIC               : 7d2636a0
UpperTransportPDU      : 235cf8397d2636a0

LowerTransportUnsegmentedAccessPDU

CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
SEG                    : 00
AKF                    : 00
AID                    : 00
Header                 : 00
UpperTransportPDU      :   235cf8397d2636a0
LowerTransportPDU      : 00235cf8397d2636a0

NetworkPDU

IVindex                : 12345678
NetKey                 : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                    : 01
TTL                    : 07
SEQ                    : 123456
SRC                    : 0405
DST                    : 0607
LowerTransportPDU      : 00235cf8397d2636a0
NID                    : 0d
EncryptionKey          : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey             : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce          : 00871234560405000012345678
IVI NID                : 0d
CTL TTL                :   87
SEQ                    :     123456
SRC                    :           0405
DST                    :               0607
TransportPDU           :                   00235cf8397d2636a0
NetMIC Size            : 32 bits
EncTransportPDU        :               ee9639021b4d3bffd06afd
NetMIC                 :                                     2462395d
```

```
Obfuscation

Privacy Random        : ee9639021b4d3b
PECB                  : 1e71cfb3ab0a3022dc0406c426bb4c4b
preObfuscation        : 871234560405

NetworkPDU            : 0d9963fbe5af0fee9639021b4d3bffd06afd2462395d
```

### 8.11.34 DF Path Request Transmit Status

A Directed Forwarding Server receives a DF Path Request Transmit Get message or a DF Path Request Transmit Set message from a Directed Forwarding Client and responds with a DF Path Request Transmit Status message.

```
Access Payload

Opcode                : bf41 (DF Path Request Transmit Status)
Conc. Initialization  : 08
Path Req. Interval    : 01
Access Payload        : bf410801

UpperTransportAccessPDU

IVindex               : 12345678
DevKey                : 63964771734fbd76e3b40519d1d94a48
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
Device Nonce          : 02001234560405060712345678
EncAccessPayload      : 235df839
TransMIC Size         : 32 bits
TransMIC              : f829fe80
UpperTransportPDU     : 235df839f829fe80

LowerTransportUnsegmentedAccessPDU

CTL                   : 01
TTL                   : 07
SEQ                   : 123456
SRC                   : 0405
DST                   : 0607
SEG                   : 00
AKF                   : 00
```

```
AID                     : 00
Header                  : 00
UpperTransportPDU       :   235df839f829fe80
LowerTransportPDU       : 00235df839f829fe80

NetworkPDU

IVindex                 : 12345678
NetKey                  : 7dd7364cd842ad18c17c2b820c84c3d6
CTL                     : 01
TTL                     : 07
SEQ                     : 123456
SRC                     : 0405
DST                     : 0607
LowerTransportPDU       : 00235df839f829fe80
NID                     : 0d
EncryptionKey           : b47a02c6cc9b4ac4cb9b88e765c9ade4
PrivacyKey              : 9bf7ab5a5ad415fbd77e07bb808f4865
Network Nonce           : 00871234560405000012345678
IVI NID                 : 0d
CTL TTL                 :   87
SEQ                     :     123456
SRC                     :           0405
DST                     :               0607
TransportPDU            :                   00235df839f829fe80
NetMIC Size             : 32 bits
EncTransportPDU         :               ee9639021a4d3b7adfa2dd
NetMIC                  :                                     836fd349

Obfuscation

Privacy Random          : ee9639021a4d3b
PECB                    : 40af98f6e7c36fae34f10ba2bb18a86b
preObfuscation          : 871234560405

NetworkPDU              : 0dc7bdaca0e3c6ee9639021a4d3b7adfa2dd836fd349
```

# 9 References

[1] Bluetooth Core Specification, Version 4.0 or later

[2] Bluetooth Core Specification, Version 5.0 or later

[3] Bluetooth Core Specification Addendum 6

[4] Bluetooth SIG Assigned Numbers (http://www.bluetooth.com/specifications/assigned-numbers)

[5] GATT Bluetooth Namespace Descriptors (https://www.bluetooth.com/specifications/assigned-numbers/gatt-namespace-descriptors)

[6] Bluetooth SIG Company Identifiers (https://www.bluetooth.com/specifications/assigned-numbers/company-Identifiers)

[7] Core Specification Supplement (CSS) v6 or later

[8] RFC4122 – A Universally Unique Identifier (UUID) URN Namespace (https://tools.ietf.org/html/rfc4122)

[9] RFC4493 – The AES-CMAC Algorithm (https://tools.ietf.org/html/rfc4493)

[10] RFC3610 – Counter with CBC-MAC (CCM) (https://tools.ietf.org/html/rfc3610)

[11] Bluetooth Mesh Model specification

[12] FIPS PUB 186-4 (http://dx.doi.org/10.6028/NIST.FIPS.186-4)