Chapter 7B: Bluetooth Mesh Details

This chapter covers lower level details of the Bluetooth Mesh Protocol, Stack and Packets.

7B.1 PROTOCOL DETAILS 2
7B.1.1 Elements 2
7B.1.2 States and Properties 2
7B.1.3 Scenes 3
7B.1.4 Messages 3
7B.1.5 Addressing 5
7B.1.6 Publish and Subscribe 7
7B.2 MODELS 8
7B.2.1 Overview 8
7B.2.2 Lighting Models 10
7B.2.3 Sensor Models 13
7B.2.4 Scene Models 14
7B.3 MESH STACK ARCHITECTURE 15
7B.3.1 Bearer Layer 15
7B.3.2 Network Layer 15
7B.3.3 Lower Transport Layer 16
7B.3.4 Upper Transport Layer 16
7B.3.5 Access Layer 16
7B.3.6 Foundation Model Layer 16
7B.3.7 Model Layer 16
7B.4 SECURITY 17
7B.4.1 Security Keys 17
7B.4.2 Preventing Replay Attacks 18
7B.4.3 Node Removal and Preventing Trashcan Attacks 18
7B.5 (ADVANCED) UPCOMING FEATURES 19
7B.5.1 Directed Forwarding 20
7B.5.2 Configuration Database 19
7B.5.3 Remote Provisioning 19
7B.5.4 Device Firmware Update (DFU) 19
7B.5.5 Subnet Bridge (SBR) 20
7B.6 (ADVANCED) PACKET DETAILS 21
7B.6.1 Access Messages 21
7B.6.2 Control Messages 23
7B.6.3 Packet Segmentation and Reassembly 25
7B.7 EXERCISES 26
Exercise 7B.1 Add more lights to the Network and Create/Modify Groups 26

7B.1 Protocol Details

This section defines some of the concepts needed to understand mesh networks. These will be tied together when we discuss Models in the next section.

7B.1.1 Elements

It is possible for a node to have more than one part that can be independently controlled. In Bluetooth mesh, these independent parts are called elements. For example, you may have a ceiling fan with a light that are part of the same physical fixture but are controlled separately. In that case, you would have one node (the fixture) with two elements – one to control the fan and the other to control the light.

Likewise, hardware that can be controlled in multiple ways may have more than one element. For example, an RGB LED can be controlled using Hue, Saturation, and Lightness. In that case, the RGB LED will have 3 elements – one for the top level HSL and Lightness, one for Hue, and one for Saturation. This will be discussed in more detail later when we talk about lighting models.

7B.1.2 States and Properties

States

Elements can be in various conditions and in Bluetooth mesh, these conditions are stored in values called states. Each state is a value of a certain type contained within an element. In addition to the values, states have behaviors that are associated with that state. States are defined by the Bluetooth SIG.

For example, there is a state called *Generic OnOff*' which can have two values – ON or OFF. This is useful for devices like light bulbs or fan motors, etc. The term Generic is used to indicate that this state and its behaviors may be useful in different kinds of device.

Properties

Properties also contain values relating to an element, but unlike states, properties provide context for interpreting states. For example, consider a device that wants to send a temperature state value. The temperature state may be "Present Indoor Ambient Temperature" or "Present Outdoor Ambient Temperature". In this case, a property would be used to provide context for the temperature state value.

Properties can be Manufacturer properties which are read-only or Admin properties which allow read-write access.

State Transitions

State transitions may be instantaneous or may execute over a period called the transition time.

State Binding

States may be bound together such that a change in one state causes a change in the other. One state may be bound to multiple other states. For example, a light controlled by a dimmer switch will have one state called Generic OnOff and another called Generic Level to specify the brightness. If the light is in the ON state but is dimmed to the point that the Level becomes zero, then the OnOff state will transition to OFF.

State binding is defined by the models that contain the states in question. These can be found in the Bluetooth Mesh specification.

7B.1.3 Scenes

A scene is a collection of states that is stored together and identified with a 16-bit Scene Number. A scene can be recalled by a scene message or can be recalled at a predetermined time. This allows a group of nodes to be set to a previously stored set of states using a single action. The scene information is stored by each element that is part of a scene. This is done using a scene model as we will discuss later.

7B.1.4 Messages

In Bluetooth Mesh, messages are broadcast by a sending device using advertising packets which can be received by any devices that are in range of the sender. Since advertising packets in BLE can be at most 31 octets, there is a limit to how much data can fit in a single packet. In cases where the data will not fit in a single packet, it will be segmented into multiple packets which will be described later.

In addition to the payload, the message contains a considerable amount of overhead information required for the mesh network to operate. The exact structure of the packets and the nature of this information will be discussed later, but for now, a summary of what the packet includes is:

1. Required BLE advertising packet fields (Size and Type). The Flags field is not included for mesh.
2. Message type and segmentation information
3. Network and Application security key information
4. Message sequence number
5. Message source address
6. Message destination address
7. Payload (including 1, 2, or 3 octets of message Opcode) – depending on the type, each packet can have a maximum of 4, 8, 11, or 12 octets of payload
8. Network and Transport layer message integrity check (MIC) information

There are several message types in mesh networks. These types can be classified as:

1. Control vs. Access
2. Acknowledged vs. Unacknowledged
3. GET, SET, STATUS
4. Segmented vs. Unsegmented

Control vs. Access

Control messages are internally generated by the stack and are sent between upper transport layers on different nodes. These include messages related to friend/low power nodes (friend poll, friend update, friend request, friend offer, etc.) and heartbeat messages. The user application does not need to handle control messages.

Heartbeat messages include the number of hops a message took to reach the destination and a list of the features supported by the node. This information can be used to optimize network performance.

Access messages are "normal" mesh messages that devices send and receive to convey information such as sensor readings, configuration settings, etc.

Acknowledged vs. Unacknowledged

As the name suggests, acknowledged messages require a response from the node that it is addressed to. The response confirms that the message was received it may also return data back to the sender (e.g. in response to a GET). If a sender does not receive the expected response from a message it may resend it. Messages must be idempotent so that a message received more than once is no different than if it had only been received once.

GET, SET, STATUS

All access messages are of the three broad types of GET, SET, and STATUS.

GET messages request a state value from one or more nodes. A GET message is always acknowledged.

SET messages are used to change the value of a state. A SET message can be either acknowledged or unacknowledged.

STATUS messages are sent in response to a GET, and acknowledged SET, or may also be sent by a node independently, for example, periodically using a timer. A STATUS message is always unacknowledged. Therefore, if a node sends a GET message but never receives a STATUS return message, it must resend the GET message.

Segmented vs. Unsegmented

Messages that can fit within one advertising packet (including mesh overhead) can be sent as unsegmented messages. For messages that won't fit in a single advertising packet, the message is segmented and sent using multiple packets.

Segmented messages can be broken into a maximum of 32 chunks that are delivered in up to 32 advertising packets.

The following table summarizes the maximum payload size for each message type:

Message Type	Max Payload Size (Octets)
Unsegmented Control or Access	11
Segmented Control	256
Segmented Access	376 or 380 *

* For Access messages, there is a transport layer message integrity check called the TransMIC which will be discussed later. The size of this value for a given message determines the maximum payload.

7B.1.5 Addressing

Mesh messages have a source address and a destination address. Both addresses are 16-bit values. There are three types of addresses defined for messages. They are:

1. Unicast
2. Group
3. Virtual

Note: there is also an unassigned address of 0x0000, but it is not used in messages.

The range and number of each type of address is:

Address Type	Address Range	Number of Addresses
Unassigned	0b0000000000000000	1
Unicast	0b0xxxxxxxxxxxxxx *	32767
Group	0b11xxxxxxxxxxxxx	16384
Virtual	0b10xxxxxxxxxxxxx	16384 hash values

*(excluding 0b0000000000000000)

Unicast

A unicast address is used to communicate with a single element in a single node. Each element in a network must have a unicast address that is unique to that network. During provisioning, the primary element in a node is assigned a unique unicast address and each additional element in the node uses the next address.

The source address in any message must be a unicast address. That is, the message must specify the specific element that message was sent by. Group and Virtual addresses are not allowed as the source address.

Group

As the name implies, a group address is used to communicate with one or more elements. Group addresses are either defined by the Bluetooth SIG (known as fixed group addresses) or are assigned dynamically for a given mesh network. There are 16K total group addresses available. The SIG has reserved space for 256 of them to be fixed while the rest can be dynamically chosen by the network.

Of the 256 group addresses that the SIG has reserved for fixed addresses, currently only 4 of them are assigned specific purposes. The rest are reserved for future use. They are:

Fixed Group Address	Name
0b1111111100000000 – 0b1111111111111011	Reserved
0b1111111111111100	all-proxies
0b1111111111111101	all-friends
0b1111111111111110	all-relays
0b1111111111111111	all-nodes

Other group addresses can be assigned for any logical group in the network. For example, room names such as kitchen, bedroom or living room could be identified as group names to control multiple elements at once. As another example, you can have one switch turn on/off multiple bulbs at the same time with a single message to a group address.

Virtual

A virtual address is assigned to one or more elements across one or more nodes. A virtual address takes the form of a 128-bit UUID that any element can be assigned to, like a label. This 128-bit address is used in calculating the message integrity check.

The 14 LSBs of the virtual address are set to a hash of the label UUID such that each hash represents many label UUIDs. When an access message is received for a virtual address with a matching hash, each corresponding label UUID is compared as part of the authentication to see if there is a matching element.

This may be used by a manufacturer to allow mesh networks including those devices to send messages to all similar devices at one time.

7B.1.6 Publish and Subscribe

Sending a message is known in Bluetooth mesh as "Publishing". While smart clients (such as smartphones) will know the destination of messages, a standard mesh device needs to be configured to know where and how messages need to be sent. For example, a light switch needs to know things like:

- The destination of its OnOff Set messages (this can be a unicast or a group address).
- What security information to use to protect the messages.
- How many times to retransmit the message and what interval to use between retransmissions.

In addition to that, a sensor may need to be configured to periodically publish the sensor data.

Configuration will be discussed in more detail later.

Models can be "Subscribed" to group addresses so that they will process messages sent to a specific group or groups. That is, a given model will receive messages sent to its element's unicast address and to all groups that the model is subscribed to.

7B.2 Models

7B.2.1 Overview

Models are used to define the functionality of a node – they bring together the concepts of states, transitions, bindings, and messages for an element. These are analogous to Services in regular Bluetooth devices.

Remember that a single device may have multiple elements and each element may have multiple models. For example, consider a ceiling fan with a light. There may be two elements: one for the fan and one for the light. The fan may have a GenericLevel Server model as well as the required configuration and node health models (discussed later) while the light may have a Light Lightness Server model (also discussed later). Each element might also have a Scene Server model so that sets of states can be stored and recalled.

There are three categories of models in mesh networks:

1. Server
2. Client
3. Control

Server

A server model defines a collection of states, transitions, bindings, and messages that control how an element behaves. A server is something that holds state values such as a light bulb (OnOff, Level, etc.). It will typically be subscribed to one or more group addresses to receive messages from the appropriate client(s). A configuration manager or mesh manager is responsible for subscribing server models to the appropriate group addresses. Again, configuration will be discussed in more detail later.

Since servers keep track of states, they need to have defined values at power up. Many of the server models make use the Generic OnPowerUp state to specify what the device does at power up. That state allows for three values:

Value	Description
Off (0x00)	The element is off at power up.
Default (0x01)	The element is on at power up. Other states are used to store the default power up state values. For example, a state may be at 10% power, 50% power, or 100% power depending on the stored default value.
Restore (0x02)	If a transition was in progress at power down, the target state is restored at power up. If not, the state that the element was in at power down is restored.

Client

A client model does not define any states. Rather, it defines the messages that it can use to interact with one or more servers by sending GET/SET messages and receiving STATUS messages. A client is something that controls a server. A light switch is an example of a client that interacts with a server on a light fixture.

Control

Control models contain both a server model and a client model. A control model may also contain control logic which is a set of rules that coordinate interactions to other models that the control model connects with.

Required Models

Every node must have at least 2 models associated with it (assigned to the primary element if the node contains more than one element):

1. Mesh Model to allow Configuration
2. Node Health Model to allow the network to check on the health of a node

Model Categories

There are generic models such as *Generic OnOff Server* and *GenericOnOff Client* which may be applicable to different types of devices like lights and fans. There are also models that are specific to one type of device such as lighting (e.g. *Light Lightness Server* and *Light Lightness Client*), sensors (e.g. *Sensor Server* and *Sensor Client*), and time (e.g. *Time Server and Time Client*). Finally there are models for scenes (e.g. *Scene Server* and *Scene Client*).

As discussed previously, each of the models describes the behavior, states and messages that are applicable to that model. For example, a *Generic OnOff Server* has a single state called *Generic OnOff*. The model has four possible messages: *Generic OnOff Get*, *Generic OnOff Set*, *Generic OnOff Set Unacknowledged* and *Generic OnOff Status*.

All of the Bluetooth SIG defined models can be found in the Mesh Model Spec. This spec also includes all the defined behavior, states, and messages for each of the models.

Model Hierarchy

Models can (and often do) extend the functionality of another model. That is, models can be hierarchical. For example, the Light Lightness model extends the Generic Power OnOff Server model and the Generic Level Server model. What that means is that if you implement the Light Lightness model in an application, you get all the Light Lightness functionality plus all the Generic Level and Generic Power OnOff functionality included for free. In addition, the Generic Power OnOff model extends other models, and so on. You will be able to see the full list later when we examine the firmware.

Models that do not extend other models are known as "root models".

7B.2.2 Lighting Models

As the name suggests, the Lighting models are used for lighting control. There are many different server, client, and control models which are detailed in the mesh model spec. A few of them are:

- Light Lightness Server – Allows control using On/Off, Level, and Lightness (allows control of LED lighting which appears linear to the human eye)
- Light CTL Server - Adds control of light color temperature
- Light HSL Server – Adds control of hue and saturation (i.e. color) along with lightness
- Light Lightness Client
- Light CTL Client
- Light HSL Client

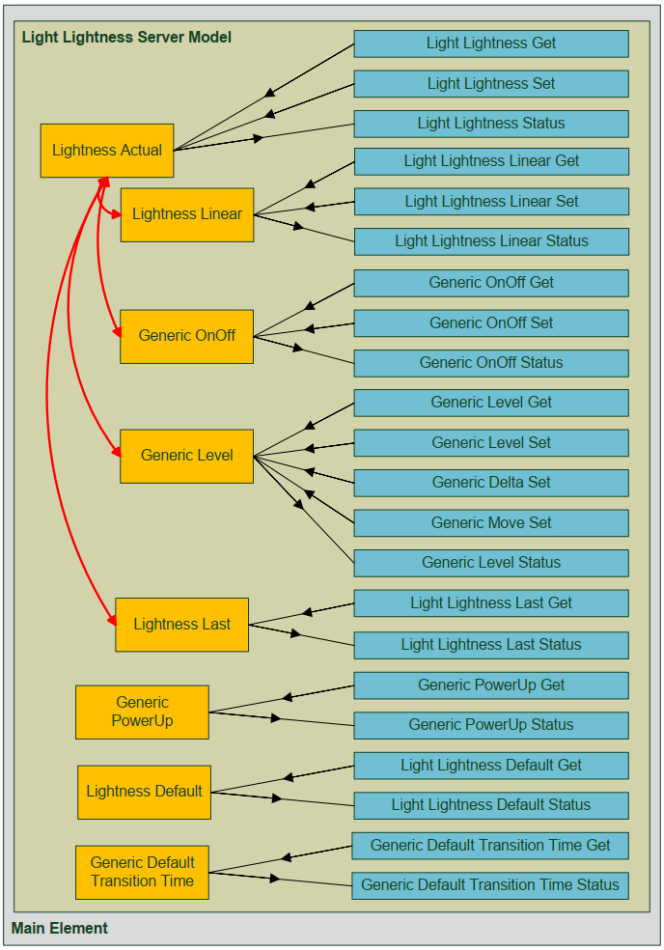
Light Lightness Server

Each of the Lighting models extend other models. As mentioned previously, the *Light Lightness Server* model extends the *Generic OnOff Server* model and the *Generic Level Server* model. This means that it contains everything from those models plus additional functionality. A benefit of extending server models is that you can use general clients to control various servers. For example, someone can buy an OnOff switch and then add it to a network to control an OnOff light, or a Dimmable light, or a Colored Light, or a music center since all those models will support *Generic OnOff* messages. Conversely, a "smart switch" can work with simple bulbs using Generic OnOff messages for the same reason.

The list of states and messages for each state for the Light Lightness Server model is shown in the table below.

State	Message
Generic OnOff	Generic OnOff Get
	Generic OnOff Set
	Generic OnOff Set Unacknowledged
	Generic OnOff Status
Generic OnPowerUp	Generic OnPowerUp Get
	Generic OnPowerUp Status
Generic Level	Generic Level Get
	Generic Level Set
	Generic Level Set Unacknowledged
	Generic Delta Set
	Generic Delta Set Unacknowledged
	Generic Move Set
	Generic Move Set Unacknowledged
	Generic Level Status
Light Lightness Actual	Light Lightness Get
	Light Lightness Set
	Light Lightness Set Unacknowledged
	Light Lightness Status
Light Lightness Linear	Light Lightness Linear Get
	Light Lightness Linear Set
	Light Lightness Linear Set Unacknowledged
	Light Lightness Linear Status
Light Lightness Last	Light Lightness Last Get
	Light Lightness Last Status
Light Lightness Default	Light Lightness Default Get
	Light Lightness Default Status
Light Lightness Range	Light Lightness Range Get
	Light Lightness Range Status

The states and messages can also be shown graphically along with binding between states:



(This figure is taken from the Bluetooth Mesh Model Specification)

Clients do not have states, but they have messages that they use to control states on servers. For example, the Light Lightness Client model has these messages:

Procedure	Message
Light Lightness Actual	Light Lightness Get
	Light Lightness Set
	Light Lightness Set Unacknowledged
	Light Lightness Status
Light Lightness Linear	Light Lightness Linear Get
	Light Lightness Linear Set
	Light Lightness Linear Set Unacknowledged
	Light Lightness Linear Status
Light Lightness Last	Light Lightness Last Get
	Light Lightness Last Status
Light Lightness Default	Light Lightness Default Get
	Light Lightness Default Status
Light Lightness Range	Light Lightness Range Get
	Light Lightness Range Status

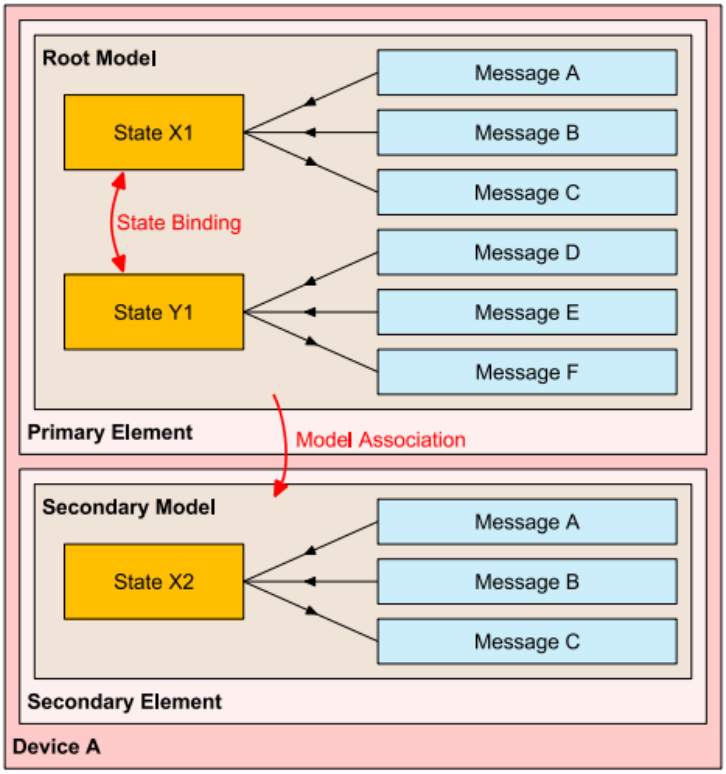
HSL (Hue, Saturation, Lightness) and CTL (Color Temperature and Lightness) Servers

For the HSL server model, the Lightness, Hue and Saturation models all extend the Generic Level model. You can imagine 3 sliders: one controlling lightness, another controlling hue, and the third controlling saturation. That means that when the server receives Generic Level Set commands (or any other messages defined by Generic Level), it needs to know if it needs to be applied to lightness, hue or saturation. To resolve this problem, an HSL server always occupies 3 elements: *Light HSL Server, Light Hue Server*, and *Light Saturation Server*.

Likewise, for the CTL server model, two elements are required: *Light CTL Server* and *Light CTL Temperature Server*.

Generically, the Mesh Core spec explains it this way:

For example, the figure below shows the element-model structure of a device that implements a root model with two bound states (X1 and Y1) 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 (X2). Messages are not capable of differentiating among multiple instances of the same state type on the same element (X1 vs. X2). Therefore, when more than one instance of a given state type is present on a device, each instance is required to be in a separate element. In this example, the second instance of State X (X2) is required to be located on the second element because it is the same type as a state (X1) in the primary element and thus has the same types of messages serving it.



(This figure is taken from the Bluetooth Mesh Specification)

7B.2.3 Sensor Models

There is a sensor server model and a sensor client model that define a standard way of interfacing with various sensors. The available states include:

Descriptor states

Sensor Descriptor states represent the attributes that describe the sensor data. This is data that does not change over the lifetime of an element (i.e. it isn't the actual sensor data). Descriptor states include:

- Property ID
- Negative and Positive Tolerance
- Sampling Function
- Measurement Period
- Update Interval

Setting states

Sensor setting states control various parameters of a sensor. These are settings that may be changed to adjust how a sensor operates. Setting states include:

- Setting Property ID
- Access
- Raw

Cadence states

The sensor cadence states control how often a sensor sends reports. It allows a sensor to be configured to send reports at a different cadence for a range of measured values. It also allows a sensor to be configured to send reports when a value changes up or down by a configured delta. The cadence states include:

- Fast Cadence Period Divider
- Trigger Type
- Trigger Type Delta Up and Down
- Minimum Interval
- Cadence Low and High

Data states

The sensor data states hold the actual measured sensor values. The data is a sequence of one or more pairs of Sensor Property IDs and Raw value fields. This allows a single model to contain more than one type of sensor data.

Series Column states

The sensor series column states allow sensor values to be organized as arrays (i.e. represented as a series of columns).

For additional details on using sensor models and their available states, see the Mesh Model specification.

7B.2.4 Scene Models

As mentioned previously, scenes store a collection of states so that they can be recalled with a single action. Scenes are managed by implementing a Scene Server Model in all the elements whose states need to be stored and a Scene Client model in the device responsible for requesting elements to store and recall scenes at the appropriate time.

Scene Store

A scene store operation stores values of the present state of all models within the element that contains the scene server model. Each scene is represented by a 16-bit network-wide Scene Number. Note that the destination address for scene store messages can be a group address so that a single scene store operation can affect multiple elements at once possibly across multiple nodes.

If a store operation is performed on a scene number that has not yet been used, then the current state values are stored. If the operation is performed on a Scene Number that already exists, then the existing scene is over-written with the new state values. The states to be stored are defined for each individual model and are provided in the mesh model spec.

Scene Recall

A scene recall operation will recall values previously stored in a scene number for an element and will set the current state to match the stored values. A scene state change can start multiple parallel model transitions. Each individual model handles the transition internally. Again, group addresses can be used so that a single scene recall operation can affect multiple elements across multiple nodes in the network.

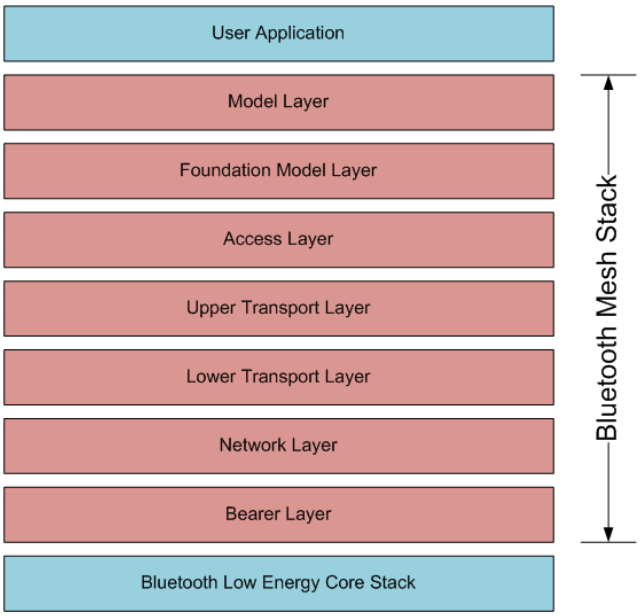
The scene allows individual control over each element based on what was stored in the scene for the element. For example, you could have a scene that turns on some lights fully, turns off other lights, and turns yet other lights on partially (i.e. dimmed). Once you have everything set the way you want it, you store it as a scene and can then recall it with a single scene recall command at a later time.

Scene Scheduler

Additional models are available to schedule scene transitions at predefined times. These are the *Scheduler Server* model, *Scheduler Setup Server* model and *Scheduler Client* model. These models are described in the mesh model spec.

7B.3 MESH Stack Architecture

The different parts of the Bluetooth mesh protocol are broken up using a layered system. In this case, the stack is broken into 7 layers as shown below. The lowest layer sits on top of the BLE stack.



7B.3.1 Bearer Layer

The lowest level defines how mesh messages get to/from the BLE stack. Presently, there are two bearer layers – the Advertising Bearer and the GATT Bearer. Most devices in a mesh network will use the Advertising Bearer to send messages using BLE advertising packets.

The GATT Bearer allows devices that don't support the Advertising Bearer (such as most smartphones) to communicate indirectly with the network via GATT Proxy nodes using the Proxy Protocol.

7B.3.2 Network Layer

The network layer defines message address types and network message format. It can support multiple bearers, each of which may have multiple network interfaces including the local interface which is used for communication between elements in the same node. Then network layer determines which network interface(s) to output messages over.

An input filter is used to determine if messages from the bearer layer should be delivered to the network layer for processing or not. An output filter is used to determine if messages should be delivered to the bearer layer or should be dropped.

The Relay and Proxy features are implemented in the network layer. Therefore, any messages not intended for this node are relayed to the appropriate network interface(s) but are not sent to the next layer in the stack.

7B.3.3 Lower Transport Layer

The lower transport layer is responsible for segmenting outgoing messages so that they will fit in the required transport PDU (Protocol Data Unit) and reassembling segmented incoming messages.

7B.3.4 Upper Transport Layer

The upper transport layer is responsible for encryption, decryption, and authentication of application data being passed to/from the access layer. It is also responsible for generating and dealing with transport control messages such as friendship and heartbeat messages.

7B.3.5 Access Layer

The access layer defines how higher-level applications can make use of the upper transport layer. It includes defining the format of the application data, defining and controlling the encryption and decryption process performed in the upper transport layer, and verifying data from the upper transport layer is intended for the right network and application before sending it further up the stack.

7B.3.6 Foundation Model Layer

The foundation model layer implements models required to configure and manage a mesh network. Specifically, the two required models: device configuration and device health.

7B.3.7 Model Layer

The model layer implements the behaviors, messages, states, etc. as defined in the mesh model specification. The user application interacts with devices by using the models. That is, this layer is the main interface between your application and the rest of the mesh stack.

7B.4 Security

Security is REQUIRED for Bluetooth mesh networks. The network, applications, and devices all have independent security that cannot be switched off. Furthermore, the provisioning process has required security built in.

In addition, mesh security protects against "replay attacks" and nodes can be removed from a network securely which prevents "trashcan attacks". These will both be discussed later.

7B.4.1 Security Keys

Three types of keys are used in mesh networks. They are:

1. Network Key (NetKey)
2. Application Key (AppKey)
3. Device Key (DevKey)

NetKey

All nodes in a mesh network must possess the network key. In fact, possession of the NetKey is what makes a node a member of a given mesh network. The NetKey allows a node to decrypt and authenticate messages at the network Layer. The mesh packet header and address are encrypted and authenticated with the network key. This allows a node to perform relay functions, but it does NOT allow the relay node to decrypt the application data that is stored in a message.

AppKey

The mesh packet payload is encrypted and authenticated with the application key. Therefore, data for a specific application can only be decrypted by nodes that have the AppKey for that application. The AppKeys are used by the upper transport layer to decrypt and authenticate messages before passing them to the access layer.

The existence of AppKeys allows multiple applications to share a mesh network (and therefore gain all the benefits of having more nodes such as increased reliability and range) without each node having access to all messages.

For example, consider a mesh network that has lights, HVAC, and home security devices on it. The light fixtures and light switches would share an AppKey for lighting; the thermostats, furnace, and air conditioner would share an AppKey for HVAC; the door locks and alarm system would share an AppKey for security. In this way, security messages can only be decrypted by devices that are part of the security system, etc.

DevKey

Each device has its own unique device key known only to itself and the provisioner device. This key is used for secure communication during configuration.

7B.4.2 Preventing Replay Attacks

A replay attack is a method in which the attacker records one or more messages and then sends them back some time later. For example, say a message is sent to unlock a door. If someone records that message, what prevents them from resending the message later to unlock the same door?

The answer in Bluetooth mesh is two message fields called the Sequence Number (SEQ) and IV Index.

Sequence Number

Each message sent by a node in a mesh network increments its 24-bit sequence number. When a node receives a message, it checks the sequence number against the sequence number from the last valid message from that node. If the number is not larger than the last sequence number, then the message is immediately discarded.

IV Index

The IV Index is used to handle overflow of the sequence number. If an element transmits a new message 10 times per second, the sequence number would wrap around after about 19 days of operation. (According to the spec, devices should not send more than 100 network PDUs in any 10 second window).

To enable longer periods of operation without overflow, a 32-bit IV index is used. Only the least significant bit of the IV index is transmitted with every message so that a node can know if the IV index of a message has been incremented. The complete IV index value is sent to each device once during provisioning and it is used in the derivation of the keys.

Each node can have an IV update procedure to signal to peer nodes that it is updating the IV index. That procedure takes at least 8 days to transition from the old to the new index.

The combination of sequence number and IV index result in messages that will not repeat on a given network for billions of years.

7B.4.3 Node Removal and Preventing Trashcan Attacks

If a device is physically removed from a network, it is important that it is also logically removed from the network so that any keys stored in the device cannot be used to mount an attack on the network. Such an attack is called a "Trashcan Attack" because it can happen if a device is disposed of and then recovered by someone from the trash.

There is a procedure for removing a node from an existing network that prevents trashcan attacks. The provisioner application is used to put the node being removed onto a black list and then it initiates a Key Refresh Procedure. The refresh results in providing all nodes in the network (except those on the black list) new network and application keys.

Note that removal of a node properly is the responsibility of the network administrator. It can be done at any time even if the node being removed is no longer on the network. For example, if a node is stolen or breaks, it can still be removed logically from the network.

7B.5 Advanced and Upcoming Features

The Bluetooth Mesh spec is still evolving. Here are a few features that are being discussed by the Bluetooth SIG and will be seen in future revisions. Several have pre-spec versions defined and will be supported in the next version of the Mesh spec that is expected mid-2020.

7B.5.1 Configuration Database (pre-spec version defined)

A device that creates the network and provisions other nodes has the information about the network security materials (network keys, application keys, device keys), addresses that have assigned to the nodes, nodes configuration, groups that have been created and other network properties. Currently there is no standard way to share this information with other devices that may want to manage and control the network. A standard format of a configuration database will be defined so that information for a network can be stored (e.g. on the cloud) and shared (e.g. with other phones and tablets) so that multiple devices will be able to provision, configure, and connect to a mesh network.

The configuration will be in JSON format and the schema will be defined in an upcoming revision of the mesh spec.

7B.5.2 Remote Provisioning (pre-spec version defined)

Today, provisioning is done via a direct connection from the provisioner to the unprovisioned device. This can be inconvenient if an unprovisioned device is in a physically difficult to reach location. There are plans to update the spec so that devices can be provisioned remotely across the existing mesh network that they are being provisioned to join.

Cypress Mesh apps can be compiled with the flag "REMOTE_PROVISION_SERVER_SUPPORTED" to enable this feature. See the mesh_onoff_server snip for an example of the required code changes.

7B.5.3 Device Firmware Update (DFU) (pre-spec version defined)

Updating firmware in mesh devices is possible today for nodes that can have a GATT connection while provisioned (e.g. GATT proxies). In this case, a custom service is included in the GATT database for OTA firmware update using a GATT connection. The update process uses the NetKey to ensure secure update of the firmware. This method is Cypress proprietary and it is not possible for a Cypress application to upgrade devices from other manufacturers.

The above described option does not work for nodes that can't have a GATT connection (e.g. Low Power Nodes) so there is no standard way to update firmware on those nodes over Bluetooth.

New models will be introduced in the mesh spec to include a way to update mesh devices over the mesh network itself. This will allow updating of low power nodes as well as remote DFU for nodes that are not easily accessible for a direct BLE GATT connection.

Cypress Mesh apps can be compiled with the flag "MESH_DFU" to enable this feature. See the mesh_onoff_server snip for an example of the required code changes.

7B.5.4 Directed Forwarding

The current version of the mesh network uses a managed-flood based scheme in which every relay node relays messages further out, so messages are propagated in all directions. In the next revision of the spec, relaying may alternately use a directed forwarding scheme, in which only relay nodes along a path toward the destination will relay the message until the message reaches the destination. The scheme is chosen by the node that originates the message. Nodes forming a path from a source to the destination may be explicitly configured by a Configuration Manager or a path may be discovered and maintained by a node that is originating messages. The path establishment and maintenance policy of an originator is configured by the Configuration Manager along with control aspects such as path lifetime, verification and re-establishment cadence, number of redundant lanes, and whether the paths are unidirectional or bidirectional.

7B.5.5 Subnet Bridge (SBR)

A mesh network can have one or more subnets that facilitate security domain 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. At the time of provisioning, a device is provisioned to one subnet and may be added to more subnets using the Configuration Model. A message transmitted in one subnet is only forwarded and processed by nodes that share the same network key and belong to the same subnet. The new protocol will add logic to configure a bridge device to be able to forward messages between subnets.

7B.6 (Advanced) Packet Details

Bluetooth mesh networks are accomplished using Advertising packets. The Advertising packet is 31 octets, but much of that space is used up by network overhead. That overhead includes information from the network layer, information from the lower transport layer, and information from the upper transport layer.

As was discussed in the prior chapter, the number of octets available for a message payload depends on whether it is a control or access messages and on whether the message is segmented or unsegmented.

The packets include message integrity checks for the network layer (for all messages) called the NetMIC and at the upper level transport layer (only for access messages) called the TransMIC. These are used to verify message integrity (i.e. the message was not modified - either intentionally or unintentionally) during transmission and to validate the message authenticity (i.e. it was sent by the stated sender). The two fields for integrity checks are as follows:

Name	Message Type	Size (Octets)
NetMIC	Control	8
NetMIC	Access	4
TransMIC	Unsegmented Access	4
TransMIC	Segmented Access	4 or 8

7B.6.1 Access Messages

Unsegmented Access Messages

Unsegmented access messages can contain up to 11 octets of payload (1, 2, or 3 octets of which are the opcode). The full 31 octets in the advertising packet (assuming a max size packet) are allocated as shown in the table below including the level in the stack that is responsible for each item.

Octet	# of Octets	Field Name	Level	Notes
0	1	Length	BLE ADV Packet	Advertising Packet Length
1	1	Type	BLE ADV Packet	Advertising Packet Type = Mesh Message
2	1	IVI | NID	Network	1 bit: LSB of IV Index 7 bits: value derived from NetKey to identify the encryption and privacy keys used for the packet
3	1	CTL | TTL	Network	1 bit: indicate an access message (0) 7 bits: Time to Live (TTL)
4 – 6	3	SEQ	Network	Sequence Number
7 – 8	2	SRC	Network	Source Address
9 - 10	2	DST	Network	Destination Address
11	1	SEG | AKF | AID	Lower Level Transport	1 bit: unsegmented message (0) 1 bit: application key flag 6 bits: application key identifier
12 – 22	11	Payload	Payload	Message Payload including 1, 2, or 3 bytes of Opcode
23 - 26	4	TransMIC	Upper Level Transport	Message Integrity Check for Upper Level Transport Layer
27 - 30	4	NetMIC	Network	Message Integrity Check for Network layer (4 octets for Access messages)

Segmented Access Messages

Segmented access messages can contain 12 octets of payload for each packet except the last one which can contain up to either 4 or 8. The first packet uses 1, 2, or 3 octest of its payload for the opcode. The maximum payload for the last packet depends on the size of the Transport Message Integrity Check (TransMIC) which can be either 4 or 8 octets depending on the message. This allows a maximum of 380 or 376 octets of payload in a single access message depending on the last packet. One, two, or three of those octets is an opcode for the message.

Number of Packets	Max Payload Size (Octets)	
	4 octet TransMIC	8 octet TransMIC
1 (unsegmented)	11	n/a
1 (segmented)	8	4
2	20	16
n	(nx12)-4	(nx12)-8
32	380	376

The full 31 octets in each advertising packet (assuming all max size packets) are allocated as shown in the table below including the level in the stack that is responsible for each item.

Octet	# of Octets	Field Name	Level	Notes
0	1	Length	BLE ADV Packet	Advertising Packet Length
1	1	Type	BLE ADV Packet	Advertising Packet Type = Mesh Message
2	1	IVI | NID	Network	1 bit: LSB of IV Index 7 bits: Value derived from NetKey to identify the encryption and privacy keys used for the packet
3	1	CTL | TTL	Network	1 bit: Indicates an access message (0) 7 bits: Time to Live (TTL)
4 – 6	3	SEQ	Network	Sequence Number
7 – 8	2	SRC	Network	Source Address
9 - 10	2	DST	Network	Destination Address
11	1	SEG | AKF | AID	Lower Level Transport	1 bit: segmented message (1) 1 bit: application key flag 6 bits: application key identifier
12 – 14	3	SZMIC | SeqZero | SegO | SegN	Lower Level Transport	1 bit: Size of TransMIC 13 bits: Least significant bits of SeqAuth 5 bits: Segment offset number 5 bits: Last segment number
15 – 18 15 – 22 15 – 26	4 8 12	Payload	Payload	Message Payload including 1, 2, or 3 bytes of Opcode. The last packet is a max of either 4 or 8 octets depending on the size of TransMic. The other packets are 12 octets each since TransMic is sent only at the end of the last packet.
19 – 26 23 – 26	8 4	TransMic	Upper Level Transport	Either 4 or 8 octets depending on value of SZMIC. This is only included in the last packet.
27 – 30	8	NetMIC	Network	Message Integrity Check for Network layer

7B.6.2 Control Messages

Unsegmented Control Messages

Like unsegmented access messages, unsegmented control messages can contain up to 11 octets of payload, but the overall contents of the packet are slightly different. The full 31 octets in the advertising packet (assuming a max size packet) are allocated as shown in the table below including the level in the stack that is responsible for each item.

Octet	# of Octets	Field Name	Level	Notes
0	1	Length	BLE ADV Packet	Advertising Packet Length
1	1	Type	BLE ADV Packet	Advertising Packet Type = Mesh Message
2	1	IVI | NID	Network	1 bit: LSB of IV Index 7 bits: Value derived from NetKey to identify the encryption and privacy keys used for the packet
3	1	CTL | TTL	Network	1 bit: Indicates a control message (1) 7 bits: Time to Live (TTL)
4 – 6	3	SEQ	Network	Sequence Number
7 – 8	2	SRC	Network	Source Address
9 – 10	2	DST	Network	Destination Address
11	1	SEG | Opcode	Lower Level Transport	1 bit: Unsegmented message (0) 7 bits: Opcode
12 – 22	11	Payload	Payload	Message Payload
23 – 30	8	NetMIC	Network	Message Integrity Check for Network layer

Segmented Control Messages

Segmented control messages can contain up to 8 octets of payload each. Therefore, a control message can be at most 256 octets long.

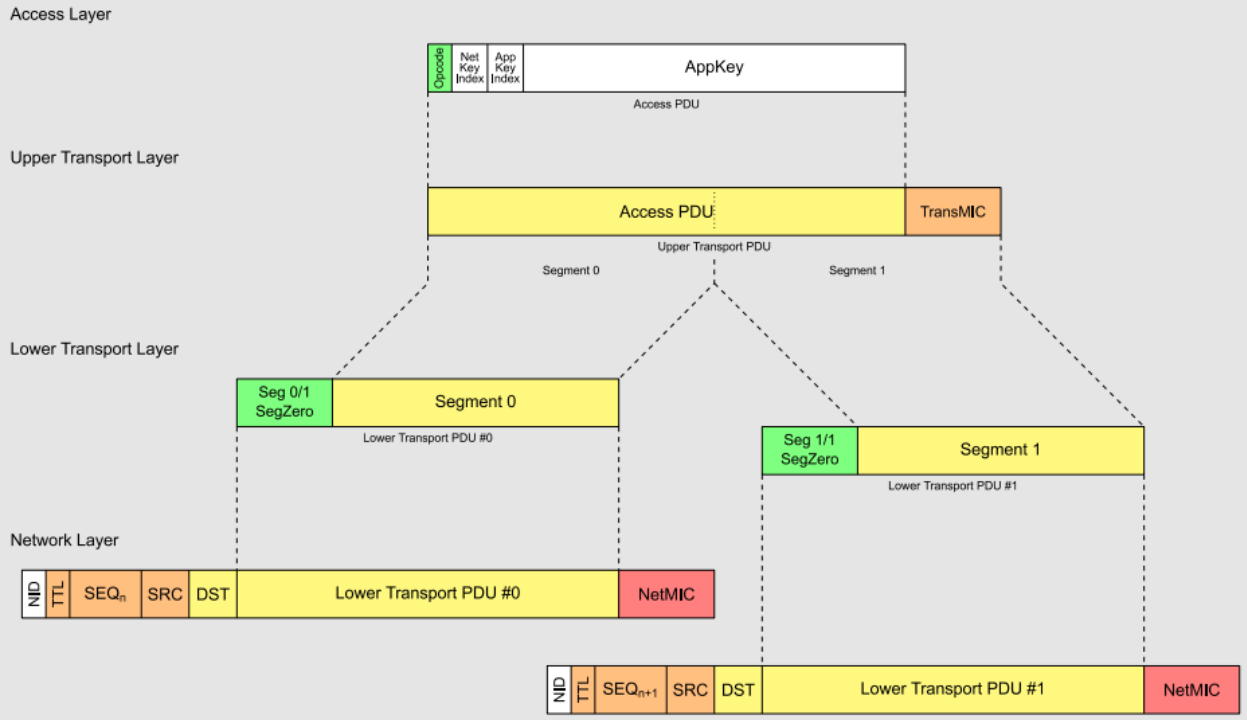
Number of Packets	Max Payload Size (Octets)
1 (unsegmented)	11
1 (segmented)	8
2	16
n	n*8
32	256

The full 31 octets in each of the advertising packets (assuming all max size packets) are allocated as shown in the table below including the level in the stack that is responsible for each item.

Octet	# of Octets	Field Name	Level	Notes
0	1	Length	BLE ADV Packet	Advertising Packet Length
1	1	Type	BLE ADV Packet	Advertising Packet Type = Mesh Message
2	1	IVI | NID	Network	1 bit: LSB of IV Index 7 bits: Value derived from NetKey to identify the encryption and privacy keys used for the packet
3	1	CTL | TTL	Network	1 bit: Indicates a control message (1) 7 bits: Time to Live (TTL)
4 – 6	3	SEQ	Network	Sequence Number
7 – 8	2	SRC	Network	Source Address
9 – 10	2	DST	Network	Destination Address
11	1	SEG | Opcode	Lower Level Transport	1 bit: Segmented message (1) 7 bits: Opcode
12 – 14	3	RFU | SeqZero | SegO | SegN	Lower Level Transport	1 bit: Unused 13 bits: Least significant bits of SeqAuth 5 bits: Segment offset number 5 bits: Last segment number
15 – 22	8	Payload	Payload	Message Payload
23 – 30	8	NetMIC	Network	Message Integrity Check for Network layer

7B.6.3 Packet Segmentation and Reassembly

As mentioned previously, the segmentation and reassembly of packets are handled by the lower transport layer. An example of how segmentation is done can be seen in the figure below.



(This figure is taken from the Bluetooth Mesh Profile Specification)

7B.7 Exercises

Exercise 7B.1 Add more lights to the Network and Create/Modify Groups

In this exercise you will add additional lights to the network you created in the previous chapter. You will experiment with associating devices to different groups.

1. Program your LightDimmable application into 3 more mesh kits.
    a. Hint: Unplug the other kits while programming each one or move them to an alternate power source that isn't connected to your computer.
2. Provision the new LightDimmable kits.
3. Optional: Rename the new kits in the app so that you can distinguish them.
4. Create two Rooms (i.e. Groups) and add 2 of the lights to each Room.
    a. Hint: Leave all the lights in the group "All". That is, just add them to the new rooms, don't move them.
5. Experiment with controlling all lights at once (All), one room at a time, and individually.
6. Optional: experiment with other room configurations. For example:
    a. Light 1 is in Room 1 and All
    b. Light 2 is in Room 2 and All
    c. Light 3 is in Room 1, Room2 and All
    d. Light 4 is only in All