Chapter 7A: Bluetooth Mesh Introduction

Time 2 Hours

This chapter covers the basics of the Bluetooth Mesh network protocol.

7A.1 Mesh Network Fundamentals

7A.1.1 Overview

Traditional Bluetooth LE devices use point-to-point communication. That is, each pair of devices send data back and forth to each other. Each of these connections has a GAP Central and a GAP Peripheral.

In contrast, in a mesh network every device in the mesh can communicate (either directly or indirectly) with every other device in the network. Some devices in the network can relay messages that they receive so that the overall communication range is extended beyond the radio range of each individual device. In theory, the range of a mesh network is unlimited as long as you have at least one relay device within range of every device in the network.

In Bluetooth Mesh, messages are sent using advertising packets. That is, no connections are made. Rather, data is broadcast by a sending device using advertising packets which can be received by any devices that are in range of the sender.

Devices in a mesh network are called "nodes". Devices that are not part of a mesh network (yet) are called "unprovisioned devices". The process of provisioning a node will also be covered later.

A mesh network can have one or more subnets that enable isolation of related groups of nodes. A subnet is a group of nodes that can communicate with each other at the network layer because they share a network key. The difference between a network and a subnet is that a node may belong to more than one subnet by having more than one network key.

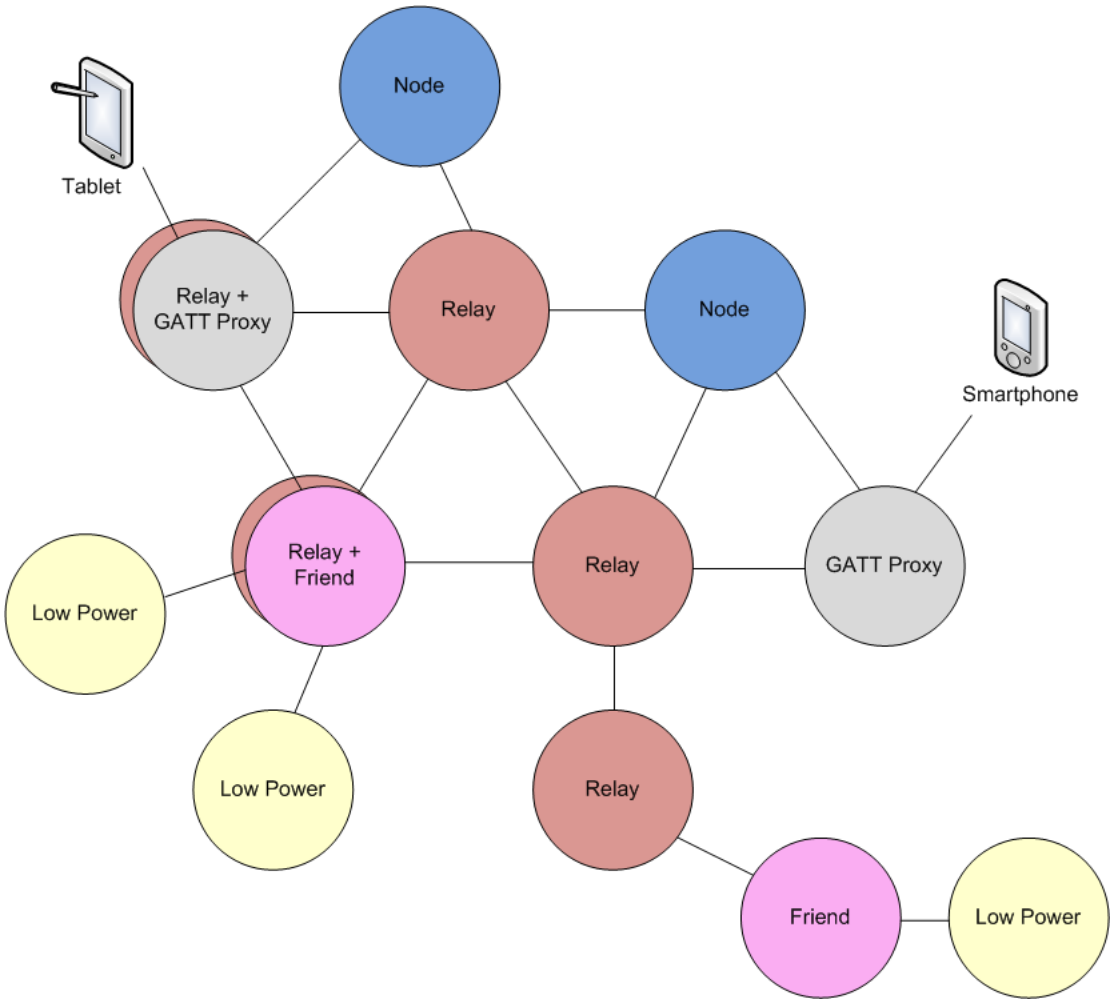
7A.1.2 Mesh Specs

Before going into more detail, it is worth noting that the Bluetooth SIG provides three specifications that contain every detail of the mesh protocol. These are:

1. Mesh Profile – defines fundamental requirements for mesh networking
2. Mesh Model Specification – defines models which are used to define basic functionality of nodes in a mesh network
3. Mesh Device Properties – defines device properties required for the mesh model spec

7A.1.3 Nodes

The following figure shows an example mesh network topology. Each of the types of node will be discussed in detail in the following sections. It is suggested that you refer to this figure while reading the descriptions.



Each node in a mesh network can send and receive messages. Each node may also implement one or more of the following features depending on its capabilities:

1. Relay
2. GATT Proxy
3. Friend
4. Low Power

Relay, GATT Proxy and Friend features can all be implemented on the same node. Typically, it doesn't make sense for a Low Power node to implement any of the other features as you will see in a minute.

Standard Node

The standard node functionality involves sending and receiving mesh messages. Every node in the network must be able to act as a standard node.

Message Caching

Each node must maintain a message cache containing all recently received messages. If a message is received more than once, it is immediately discarded. In this way, if a message is relayed by multiple nodes to a final destination, the destination only acts on the message one time.

Relay Node

Relay nodes can receive a message for the network and then retransmit it to other devices in range. This is the method by which mesh networks can cover larger distances than the range of any single device. For a network to operate, every node must be within range of at least one relay so that its messages can be forwarded on to nodes that it cannot directly communicate with.

It is common for all except low power nodes to implement a relay feature in order to maximize the possible paths through a mesh network.

Due to the message cashing described above, a relay node will only relay a given message one time.

TTL

Each message has a field called the Time To Live (TTL). This is used to determine how many times a given message will be retransmitted. By understanding the basic topology of a mesh network, the TTL can be used to prevent messages from being retransmitted too many times. This allows the mesh network to be more efficient.

In fact, there are heartbeat messages sent periodically which include, among other things, information that allow receiving nodes to determine how many hops away the sender is. Networks can use this information to adapt TTL settings to optimize the network.

Security

A relay node only decodes enough of the message to decide what to do with it. For example, it decodes the addresses for the message but not the payload if it is not intended for that node. In fact, due to the security architecture, the relay node cannot decode the payload for any messages that are not from the same network application (e.g. lighting). Security will be discussed in detail later.

GATT Proxy Node

Many existing BLE devices support traditional BLE GATT communication but not mesh communication. Most smartphones and tablets fall into this category. Since you may want to interact with a mesh network from one of those devices, the GATT proxy was created. A GATT proxy node has both a mesh interface and a GATT interface. The GATT interface is used to communicate with BLE devices that don't possess a mesh stack and then relay those messages to/from the mesh network. That is, the GATT proxy acts as a bridge between the mesh network and the traditional BLE GATT device.

Friend and Low Power Nodes

Friend and Low Power Nodes are used to optimize power consumption for constrained devices. Devices that are power constrained (e.g. a battery powered device) are designated as low power nodes. Every low power node in the network must be associated with exactly one friend node. Friend nodes are devices which are not power constrained (e.g. a device plugged into AC power) that support 1 or more low power nodes depending on its capabilities (e.g. available RAM).

When a low power node is added to a mesh network it broadcasts a request for a friend. Each friend in range that can handle a new low power node replies and the low power node selects the best friend based on how many messages the friend can store; the RSSI and the timing accuracy.

Once the relationship is established, the friend node will receive and store messages for any low power nodes that it is associated with. The low power node will periodically ask the friend node for any messages that the friend has stored for it. In this way, the low power node does not need to listen continuously for mesh packets. Instead, it can be in a low power mode most of the time and can wake up only periodically for a very short time.

For example, consider a battery powered mesh connected thermostat. It will measure the actual temperature and may send a mesh message with the temperature once per minute. This can be done with very low power consumption since the device can be sleeping all the time except for a short period each minute to send the value. However, it must also be possible to change the set point of the thermostat. In this case, instead of sending messages, the thermostat must be listening for messages. If it listens constantly for messages the power consumption will be unacceptably high, but if it only listens occasionally for messages it will likely miss messages. By making the thermostat a low power node we get the best of both worlds - it can send messages once a minute and receive any stored messages regarding the set point from its friend node. No messages are missed even though the thermostat is awake only a very small percentage of the time.

7A.1.4 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.

7A.1.5 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.

7A.1.6 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.

7A.1.7 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.

7A.1.8 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	0b0xxxxxxxxxxxxxxx (excluding 0b0000000000000000)	32767
Group	0b11xxxxxxxxxxxxx	16384
Virtual	0b10xxxxxxxxxxxxx	16384 hash values

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
0xFF00 – 0xFFFB	Reserved
0xFFFC	all-proxies
0xFFFD	all-friends
0xFFFE	all-relays
0xFFFF	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.

7A.1.9 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.

7A.2 Models

7A.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 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 OnOff functionality included for free.

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

7A.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

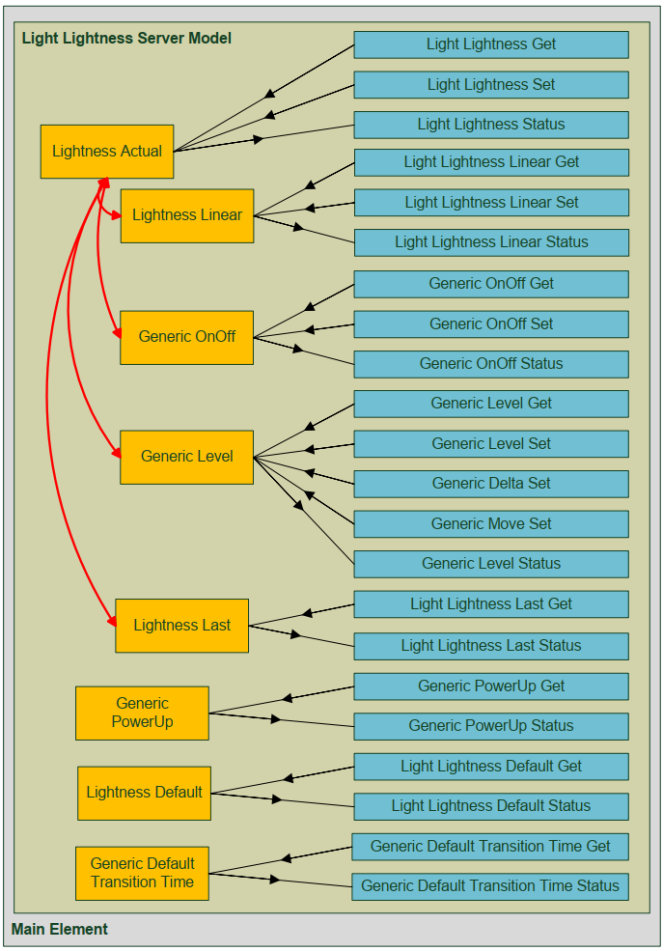
Light Lightness Server

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

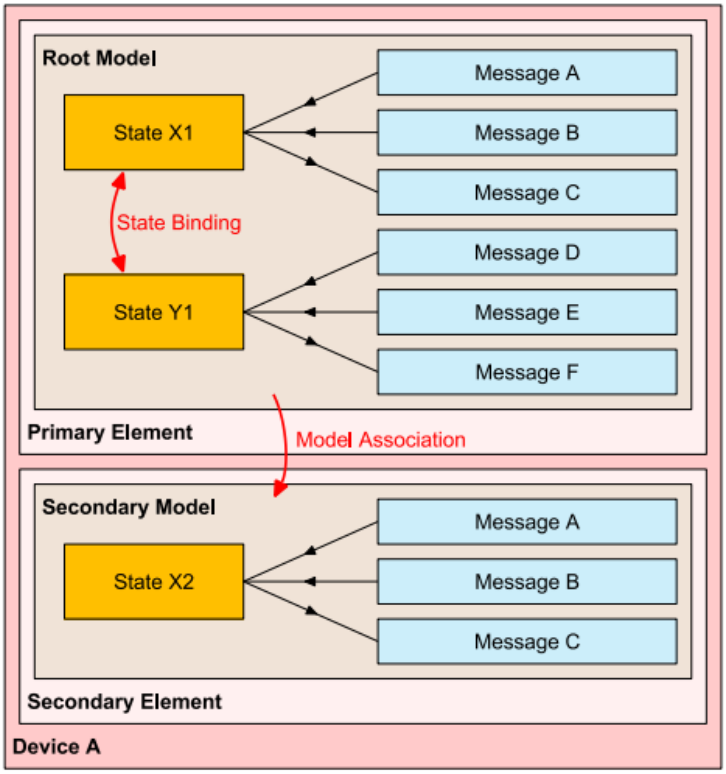
HSL (Hue, Saturation, Lightness) and CTL (Color Temperature and Lightness) Servers

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)

7A.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.

7A.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.

7A.3 Provisioning and Configuration/Management

To get a new device up and running on a Bluetooth Mesh network, it must be provisioned and configured. These are often thought of as a single step, but they are unique processes with different protocols. These will each be described separately below.

7A.3.1 Provisioning

Provisioning is the process by which a device is made a member of the mesh network and becomes a node. To be a node on a mesh network, a device needs to have the network key (and other associated network security information like the IV index) and it needs to have a unicast address assigned to its primary element. Provisioning can be done using either a GATT connection (PB-GATT) or an advertising channel (PB-ADV) as the bearer. (PB = Provisioning Bearer).

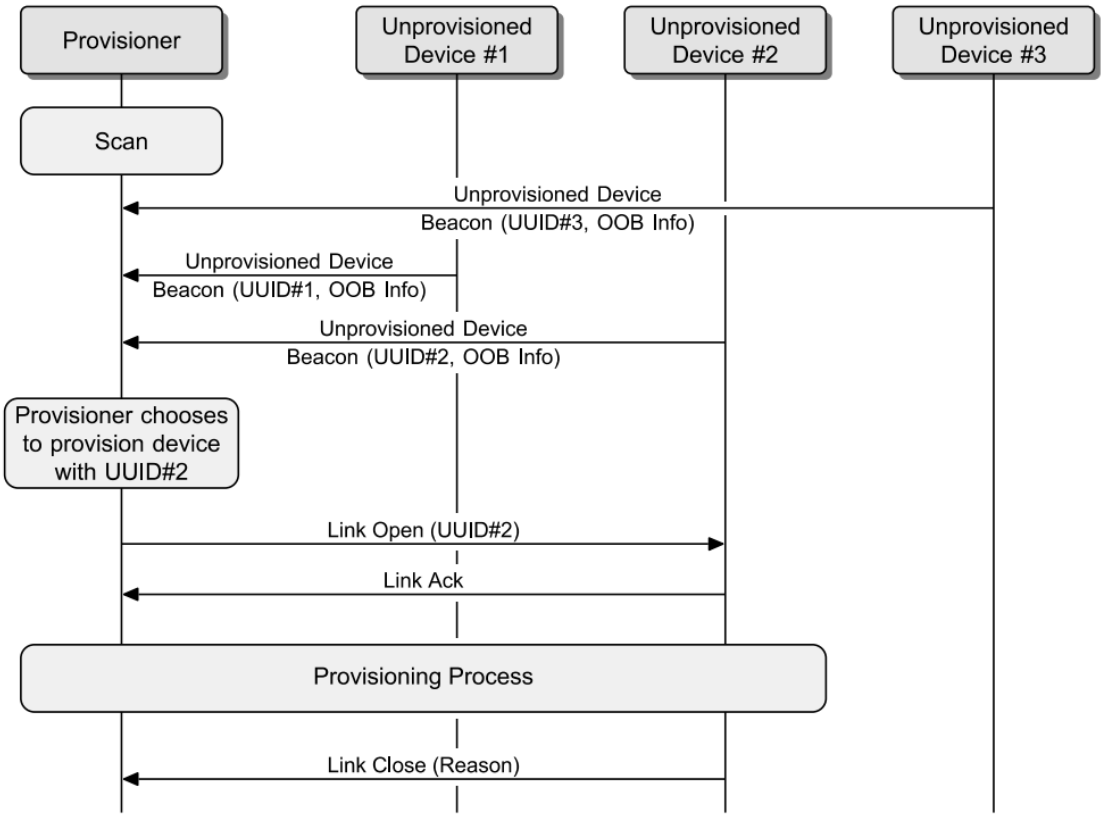
Provisioning is most commonly done using an application on a smartphone or a tablet. Note that smartphones currently do not support provisioning over an advertising channel, so from a practical standpoint, all devices should support provisioning over GATT. The Bluetooth Mesh spec strongly recommends that unprovisioned devices support both.

Beaconing

Any unprovisioned device will indicate its availability to be provisioned by sending out advertising packets of the type "Mesh Beacon".

Link Establishment

A provisioner will scan for unprovisioned devices and will choose (usually via input from the user) which device to provision. The provisioner sends a Link Open message to the device to be provisioned which will in turn respond with a Link ACK message. Once provisioning completes, the provisioner sends a Link Close message. These steps are illustrated in the figure below.



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

The remaining steps detailed below occur within the box labeled "Provisioning Process" in the figure above.

Invitation

The provisioner sends an invitation to the device being provisioned in the form of a provisioning invite protocol data unit (PDU). The device being provisioned responds with information about itself in the form of a provisioning capabilities PDU.

Exchanging Public Keys

The provisioner and the device to be provisioned exchange public keys either directly or using an out-of-band (OOB) method.

Authentication

Authentication is performed using an OOB method that depends on the capabilities of the device being provisioned. For example, if the device to be provisioned has some output mechanism, it creates a random number and indicates that number to the user (e.g. it may flash an LED a random number of times, beep a random number of times, or show the number on a display). The user then enters that number into the provisioner.

If the device has some input mechanism, then the provisioner creates a random number and presents it to the user. The user then inputs that number on the device (e.g. by pressing a button the specified number of times or entering the number using a keypad).

Either way, once the random number has been generated on one side and entered on the other, a cryptographic exchange happens between the two devices using that random number.

Distribution of Provisioning Data

Once authentication is done, a session key is derived by each device from its private key and the public key from the other device. The session key is used to secure subsequent distribution of the data needed to complete provisioning. Once provisioning is completed, the provisioned device has the network's key (NetKey), a security parameter called the IV index, and its Unicast address which was allocated by the provisioner. The device is now a node and is a part of the network. The provisioner then sends a Link Close message as described previously.

7A.3.2 Configuration/Management

Once provisioning is done, the same smartphone or tablet (i.e. the provisioner) then uses the mesh network to configure the new node. This includes distribution of application keys, assigning group addresses to models, etc.

Note that smartphones currently do not support Bluetooth mesh directly so at least one device should be configured as a GATT Proxy to allow a smartphone to do configuration once provisioning is done. The only alternative currently is to have a gateway on the mesh network that allows the smartphone to access the mesh network indirectly.

7A.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.

7A.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.

7A.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.

7A.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.

7A.4.4 Privacy

A privacy key that is derived from the NetKey is used to obfuscate header values such as the source address. This prevents passive eavesdropping from tracking devices and makes attacks based on traffic analysis difficult.

7A.5 Exercises

Exercise 7A.1 Program the LightDimmable Application and Get it Added to a Network

In this exercise, you will program one of the demo mesh applications into a mesh kit. Your device will then be added to a class-wide mesh network so that all kits in the network can be controlled simultaneously.

1. In ModusToolbox IDE, create a new application for:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_LightDimmable
2. Open the file "light_dimmable.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Dimmable Light").
3. Program the project to one of the CYBT-213043-MESH kits.
    a. Hint: You should open a terminal window for the PUART to see messages. **The default PUART baud rate for the mesh applications is 921600**.
4. Tell the instructor when you are ready so that he/she can provision your device onto a mesh network.
5. Observe the instructor as he/she controls the devices on the mesh network.