Bluetooth Mesh and Zephyr

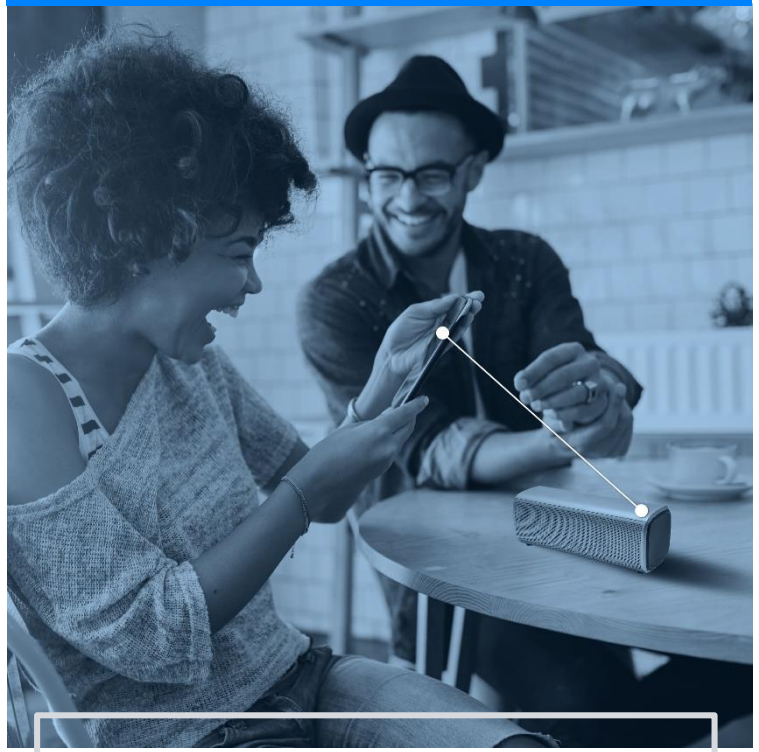
Martin Woolley

Bluetooth SIG Developer Relations Manager, EMEA

Twitter: @bluetooth_mdw

Bluetooth now comes in three delicious flavours

BR/EDR

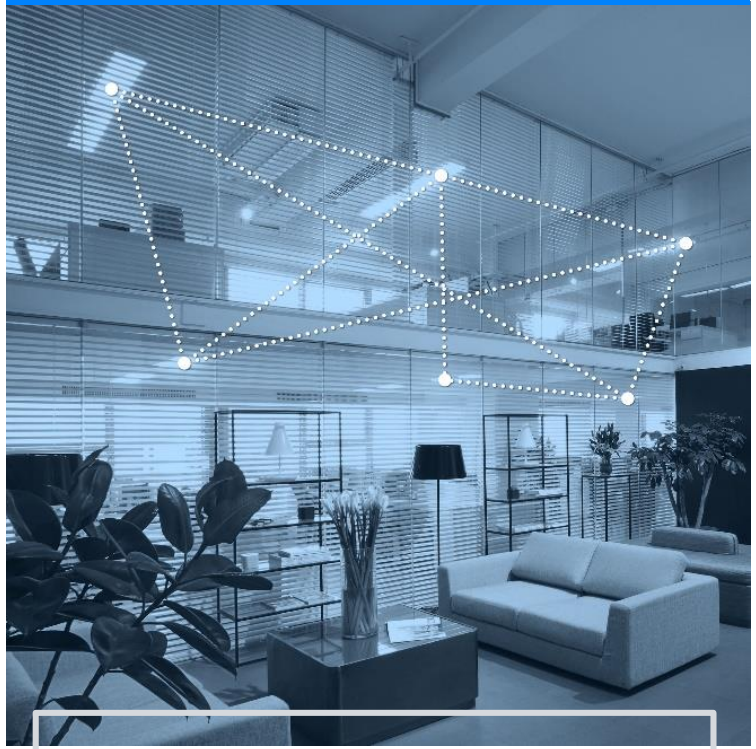


point-to-point 1:1

Low Energy (LE)

broadcast 1:m

Mesh

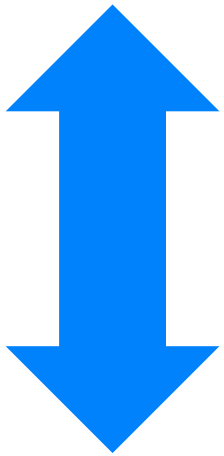


many to many m:m

relationship between Bluetooth technologies

NETWORKING

Bluetooth mesh networking



RADIO

Bluetooth BR/EDR

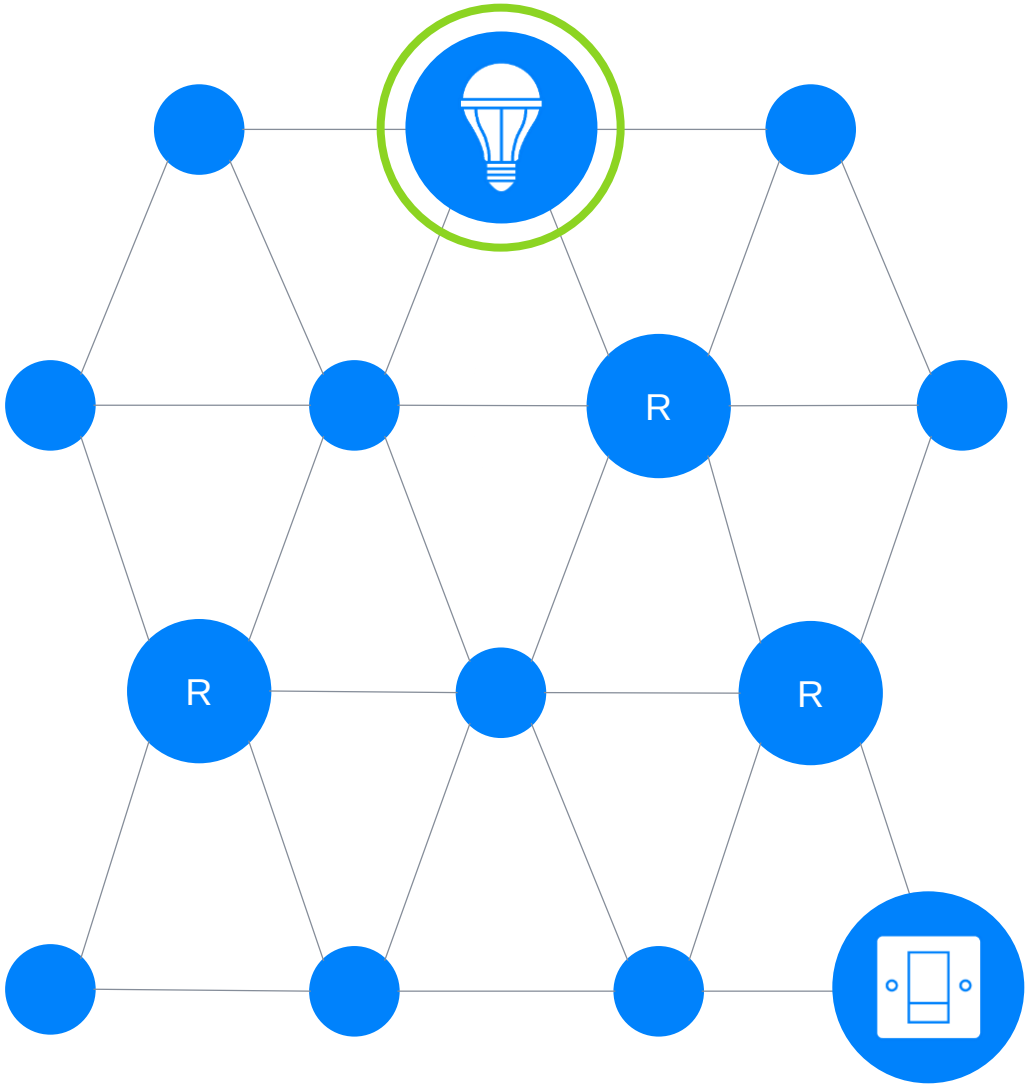
Bluetooth Low Energy

Bluetooth Mesh Networks

multi-hop, multi-path, multicast

Bluetooth Mesh

Node Network Roles



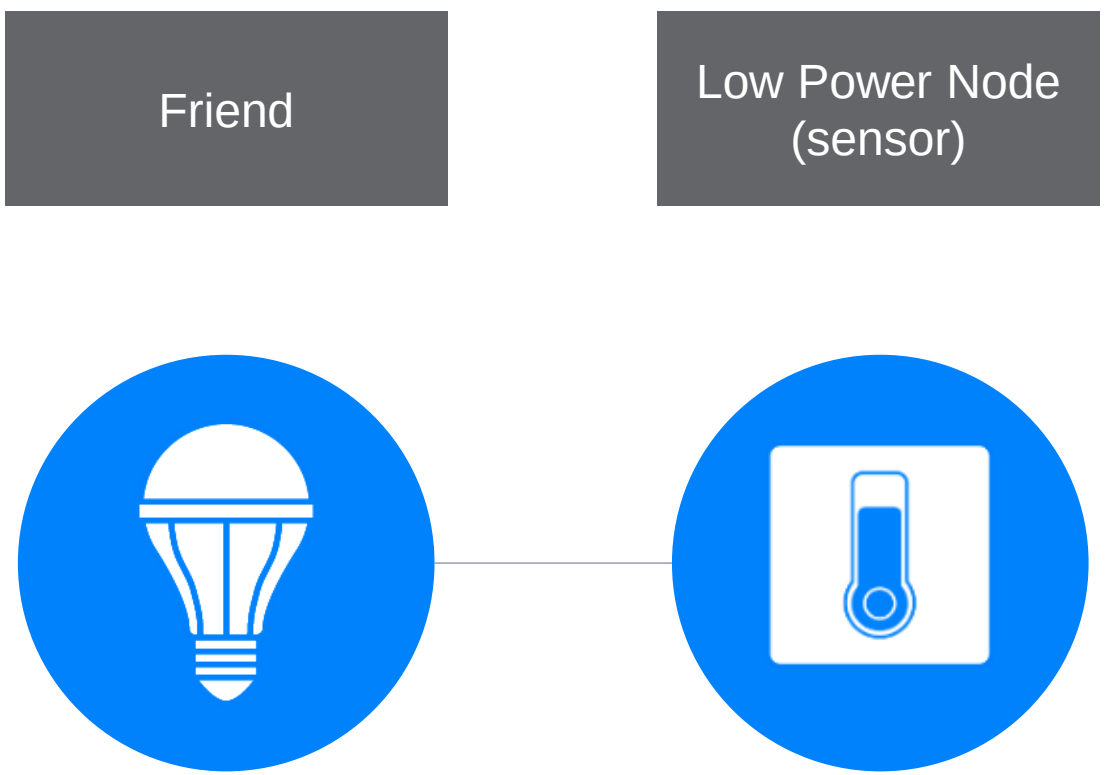
R = Relay function on

relay nodes

Messages get sent to other nodes that are in direct radio range of the publishing node

Some nodes can act as “relays” however

Relays retransmit messages so that they can travel further, in a number of “hops”

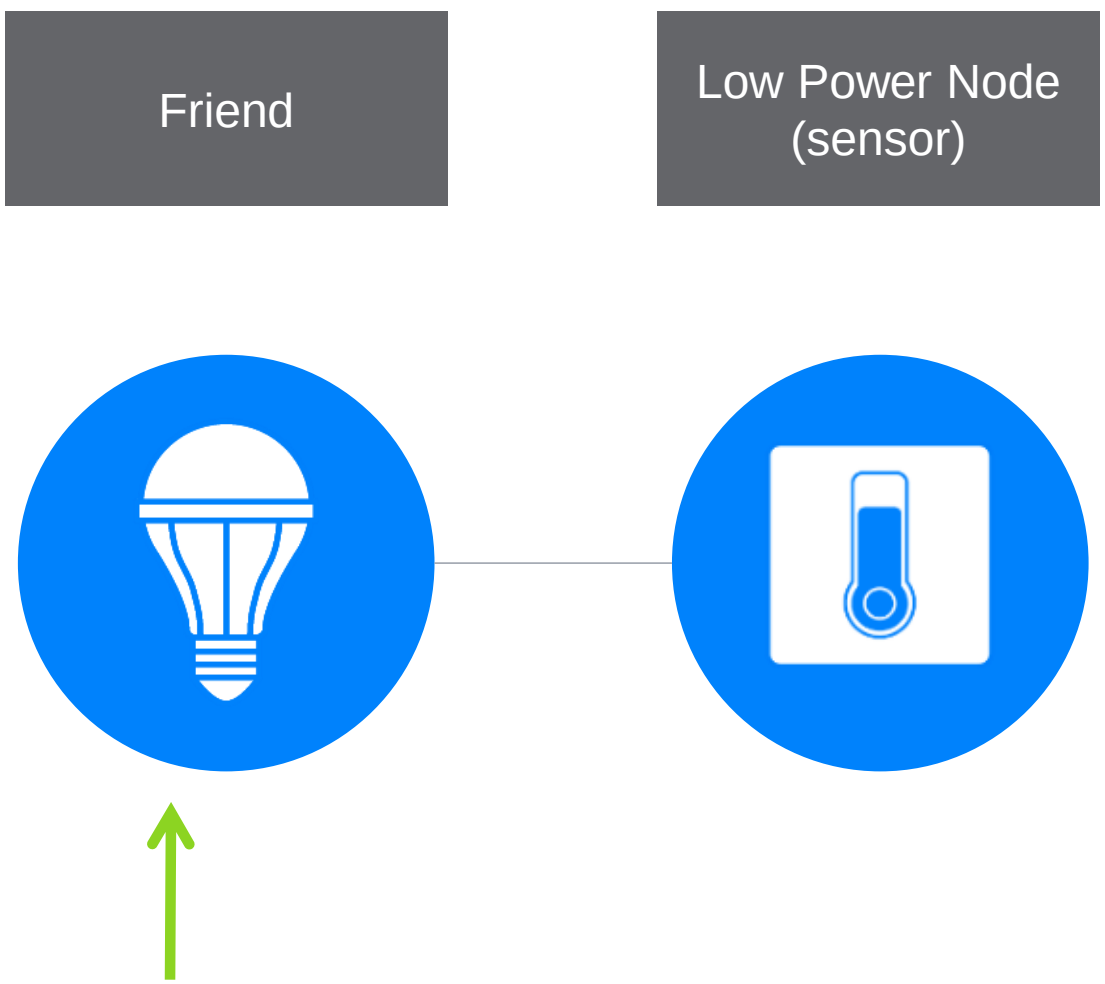


friend nodes and low power nodes

Low power nodes (LPNs) are highly power constrained

To avoid the need to operate at a high(er) duty cycle to receive messages from the mesh, an LPN works with a Friend

Friend nodes store messages addressed to LPNs they are friends with and forward them when the LPN occasionally polls

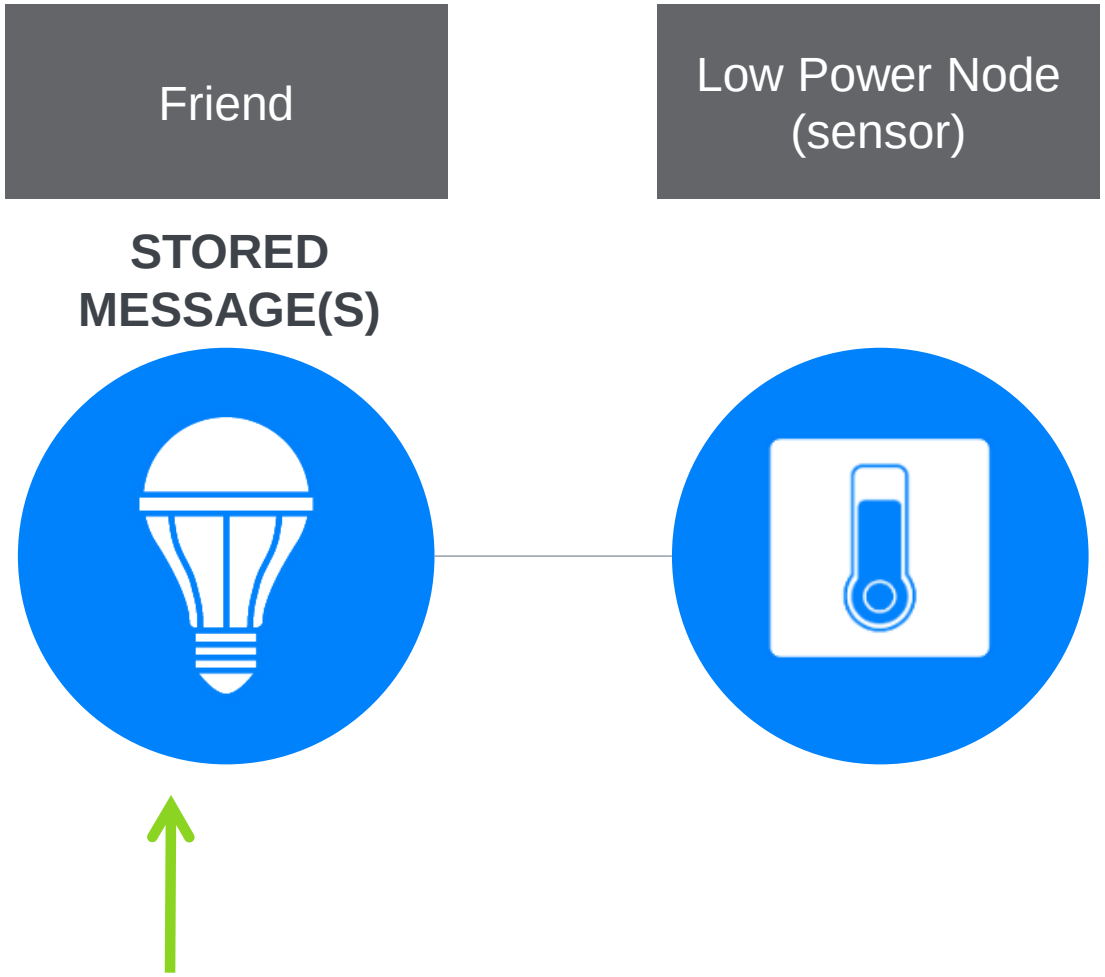


friend nodes and low power nodes

Low power nodes (LPNs) are highly power constrained

To avoid the need to operate at a high(er) duty cycle to receive messages from the mesh, an LPN works with a Friend

Friend nodes store messages addressed to LPNs they are friends with and forward them when the LPN occasionally polls

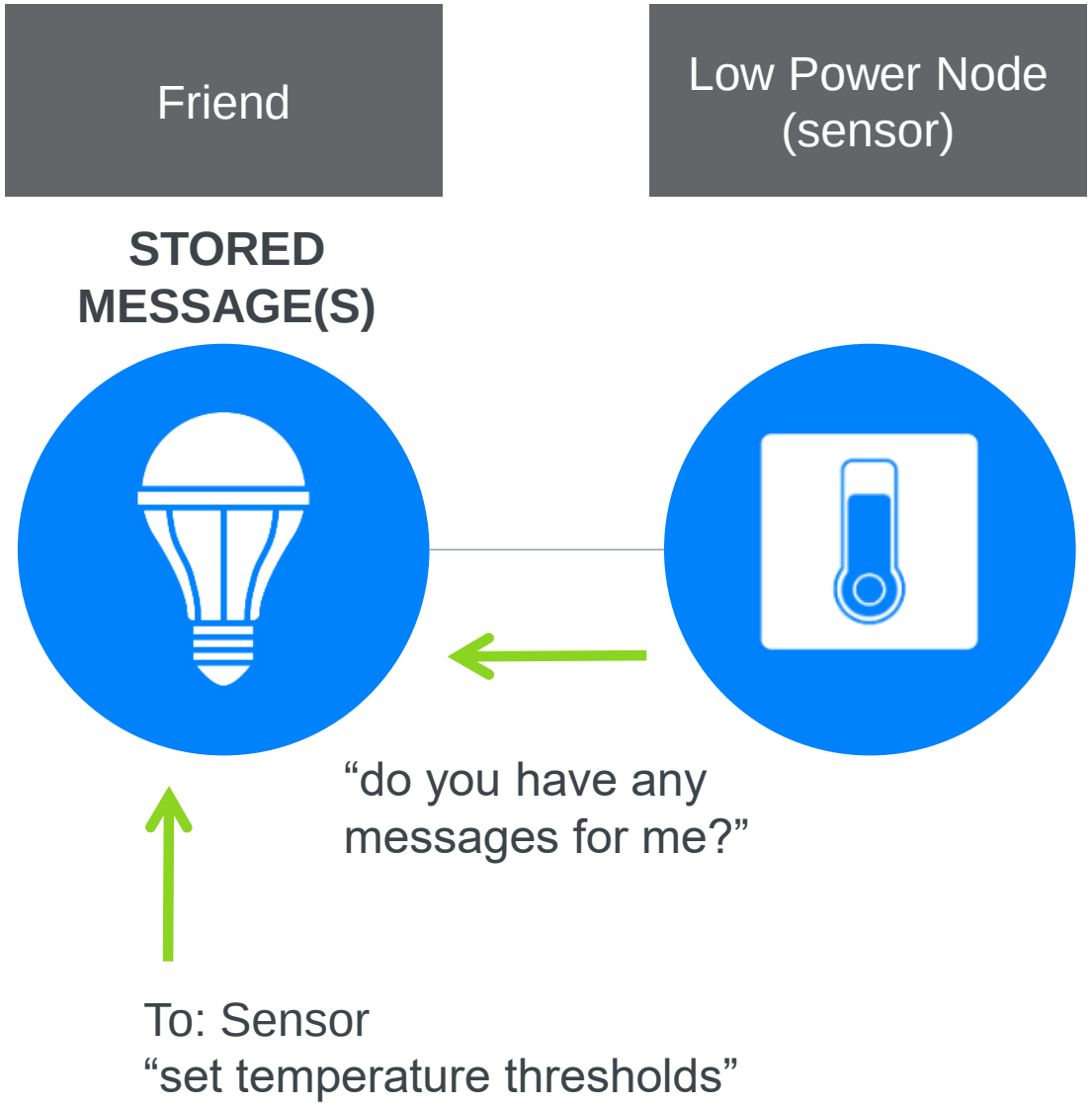


friend nodes and low power nodes

Low power nodes (LPNs) are highly power constrained

To avoid the need to operate at a high(er) duty cycle to receive messages from the mesh, an LPN works with a Friend

Friend nodes store messages addressed to LPNs they are friends with and forward them when the LPN occasionally polls

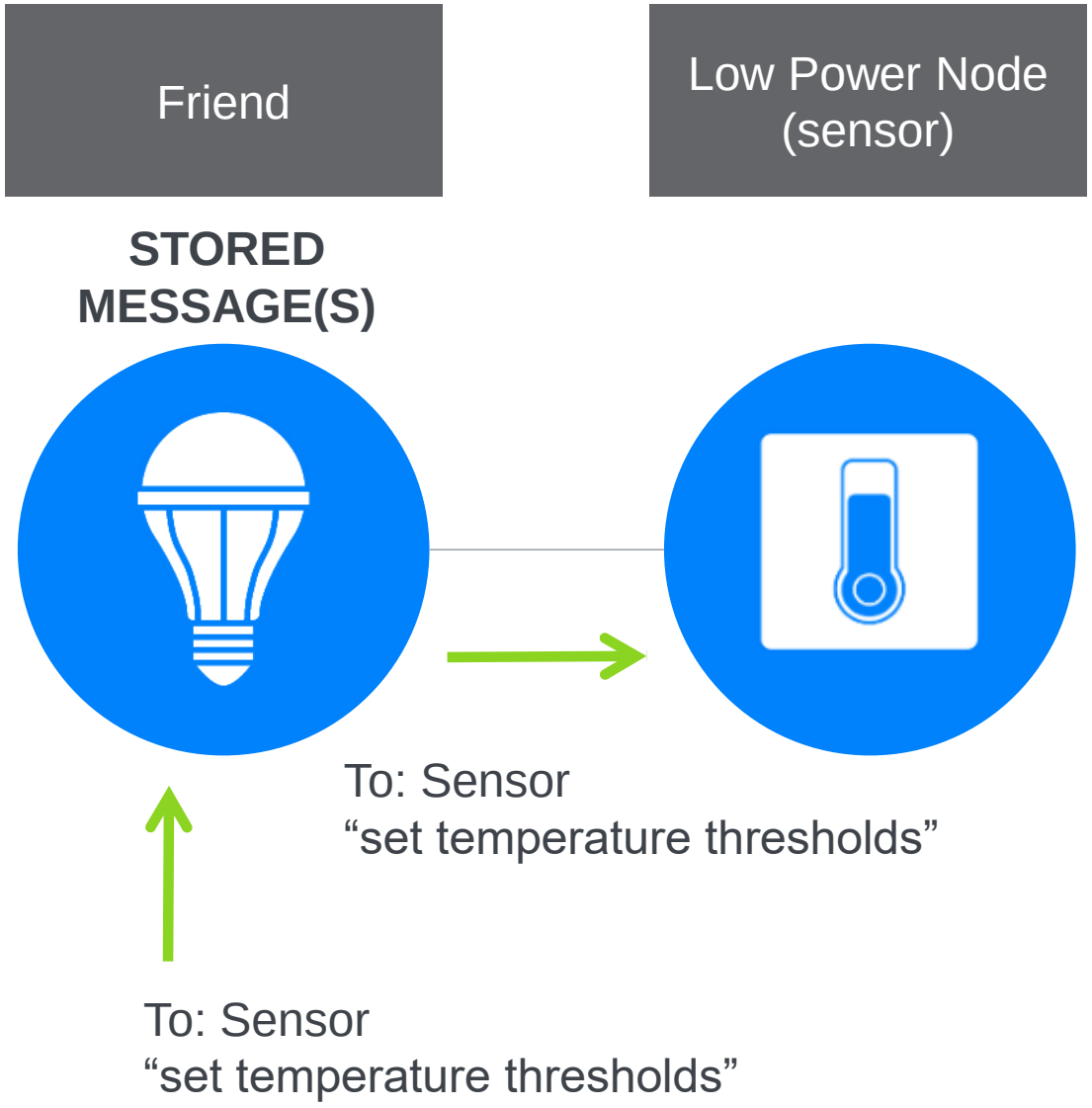


friend nodes and low power nodes

Low power nodes (LPNs) are highly power constrained

To avoid the need to operate at a high(er) duty cycle to receive messages from the mesh, an LPN works with a Friend

Friend nodes store messages addressed to LPNs they are friends with and forward them when the LPN occasionally polls

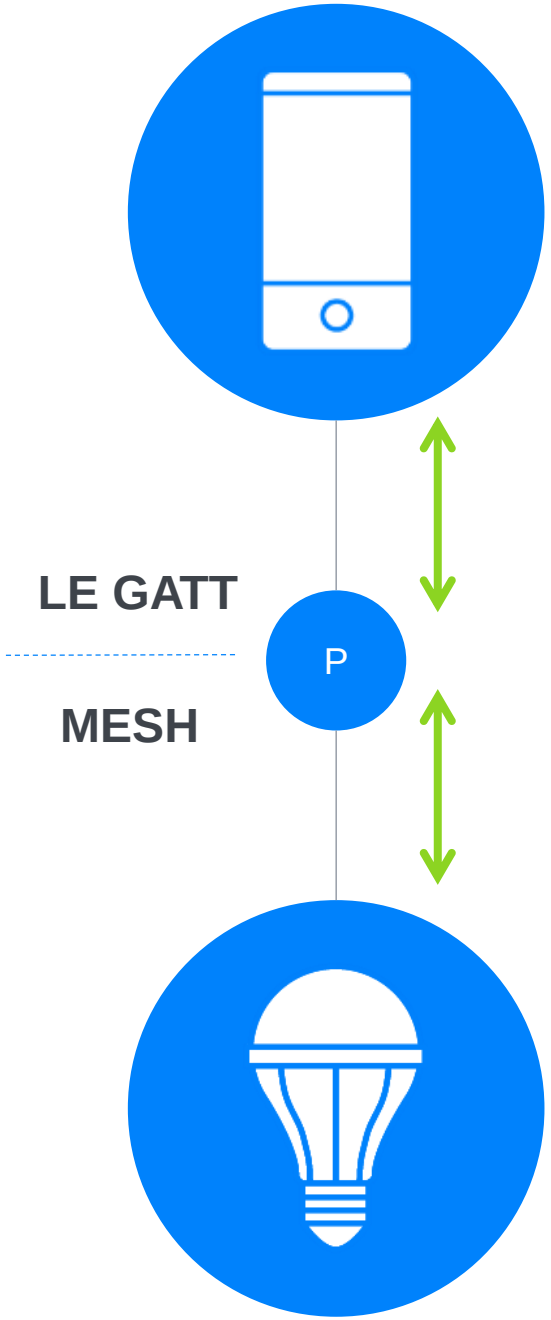


friend nodes and low power nodes

Low power nodes (LPNs) are highly power constrained

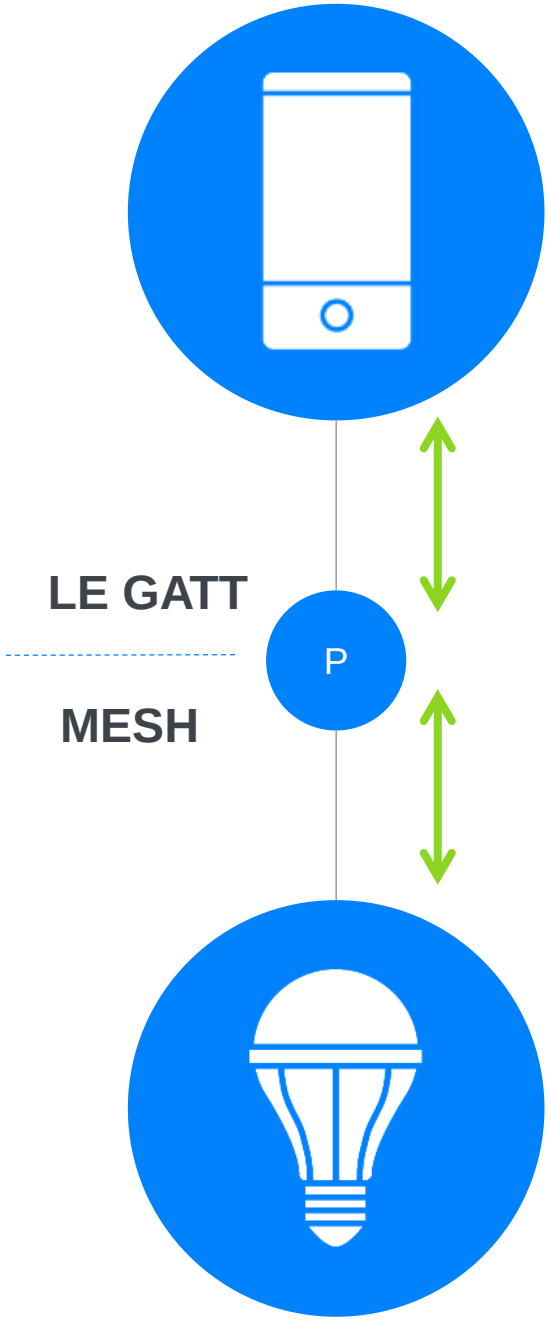
To avoid the need to operate at a high(er) duty cycle to receive messages from the mesh, an LPN works with a Friend

Friend nodes store messages addressed to LPNs they are friends with and forward them when the LPN occasionally polls



proxy nodes

Bluetooth low energy devices like smartphones can communicate with a mesh network via a proxy node



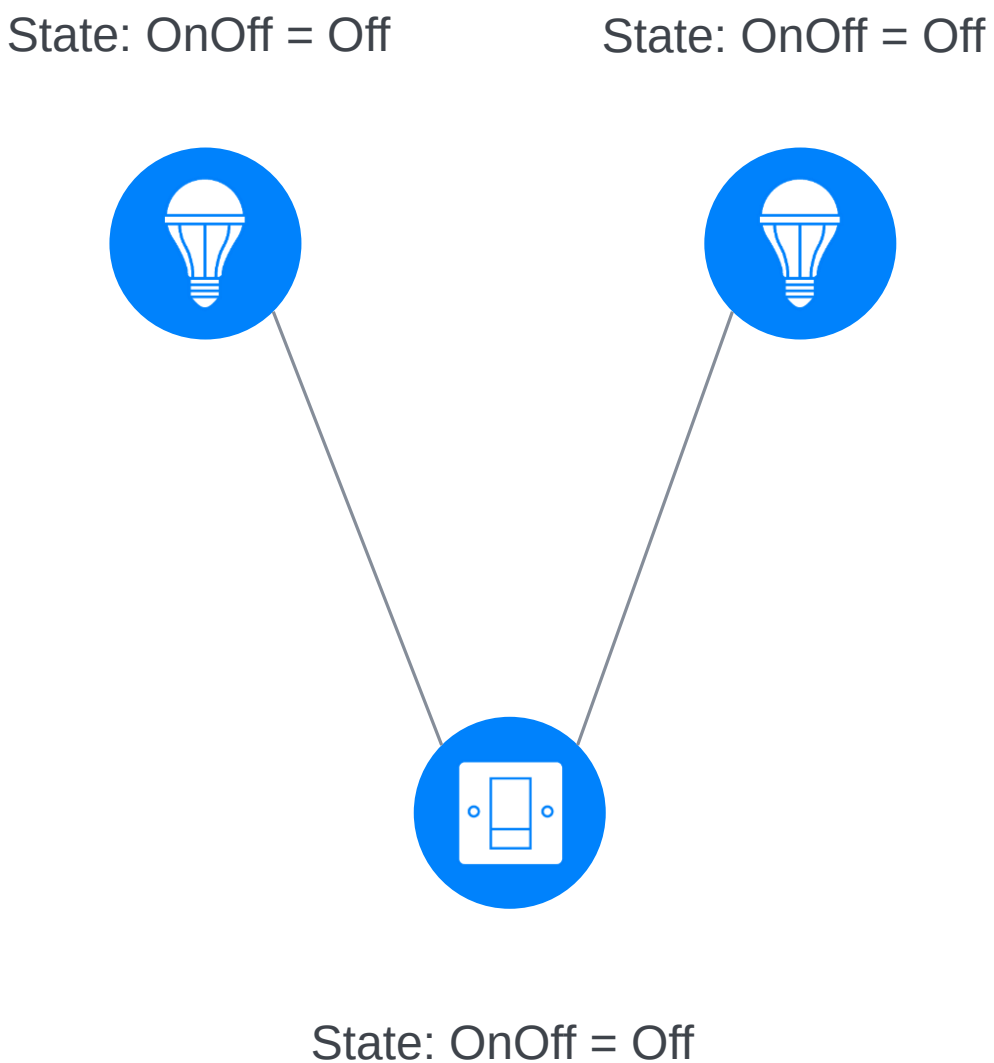
proxy nodes

Bluetooth low energy devices like smartphones can communicate with a mesh network via a proxy node

mesh monitoring and control applications

Bluetooth Mesh

Communication and Interaction



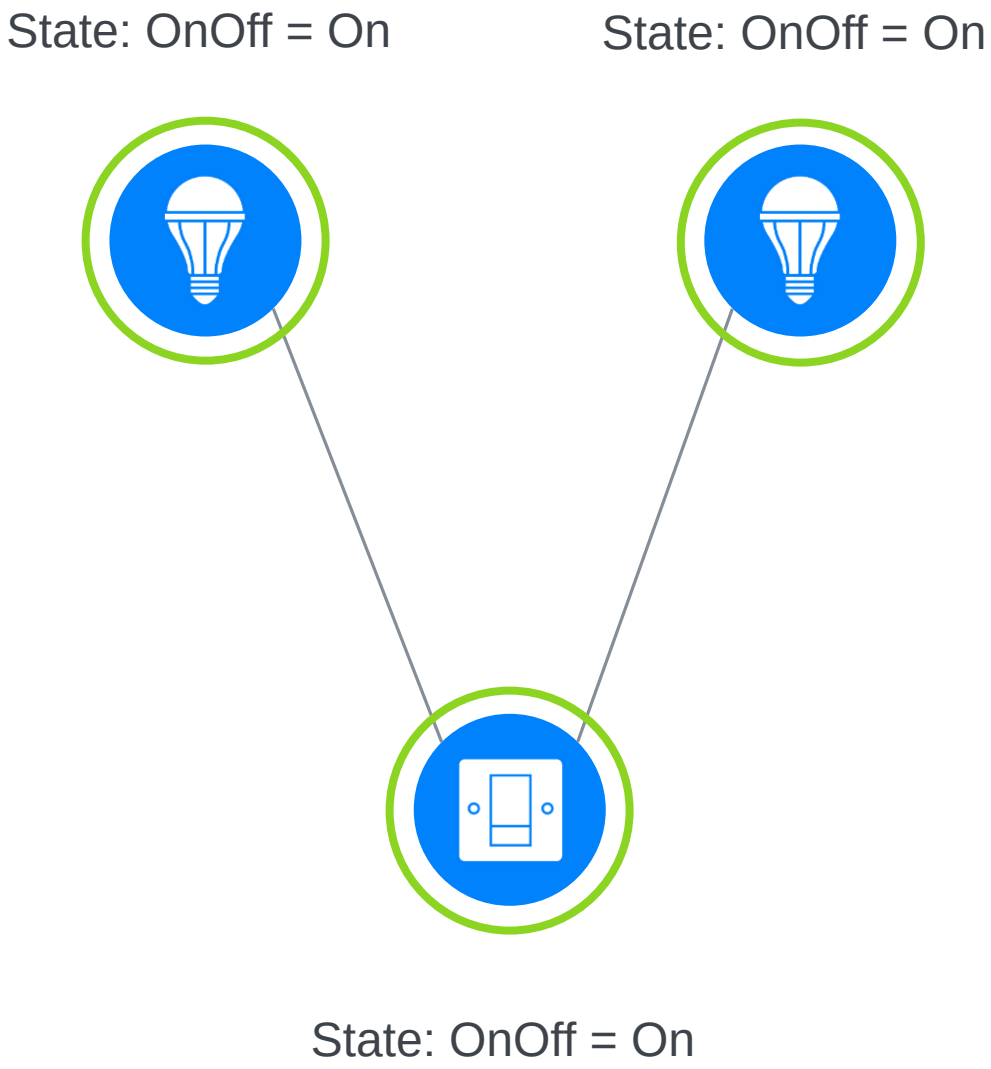
messages and state

nodes communicate with each other by sending messages

nodes have state values which reflect their condition (e.g. ON or OFF)

access messages operate on state values

- **SET** - change of **state**
- **GET** - retrieve **state** value
- **STATUS** - notify current **state**
- **ACK vs UNACK**



messages and state

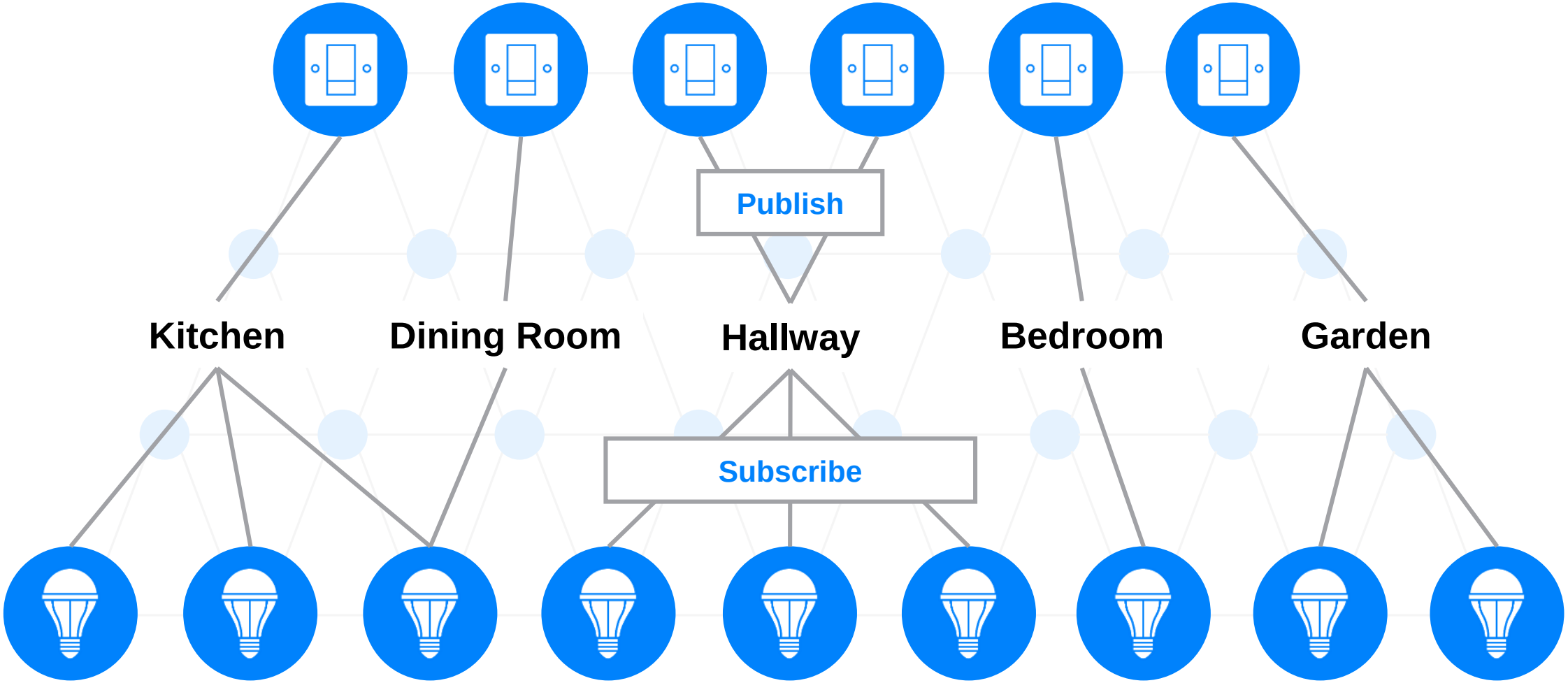
nodes communicate with each other by sending messages

nodes have state values which reflect their condition (e.g. ON or OFF)

access messages operate on state values

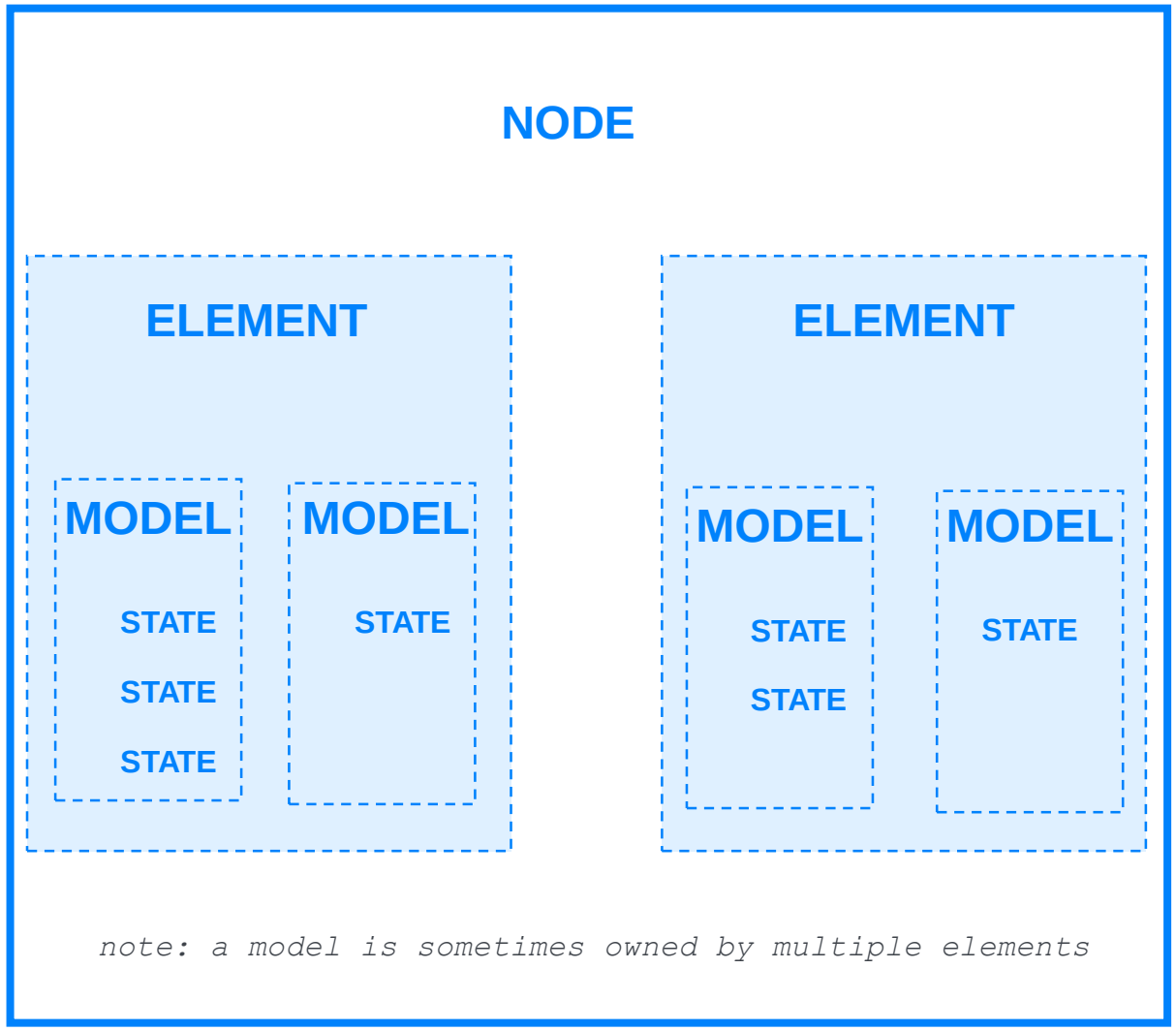
- **SET** - change of **state**
- **GET** - retrieve **state** value
- **STATUS** - notify current **state**
- **ACK vs UNACK**

the publish/subscribe communication model



Bluetooth Mesh

Node Composition

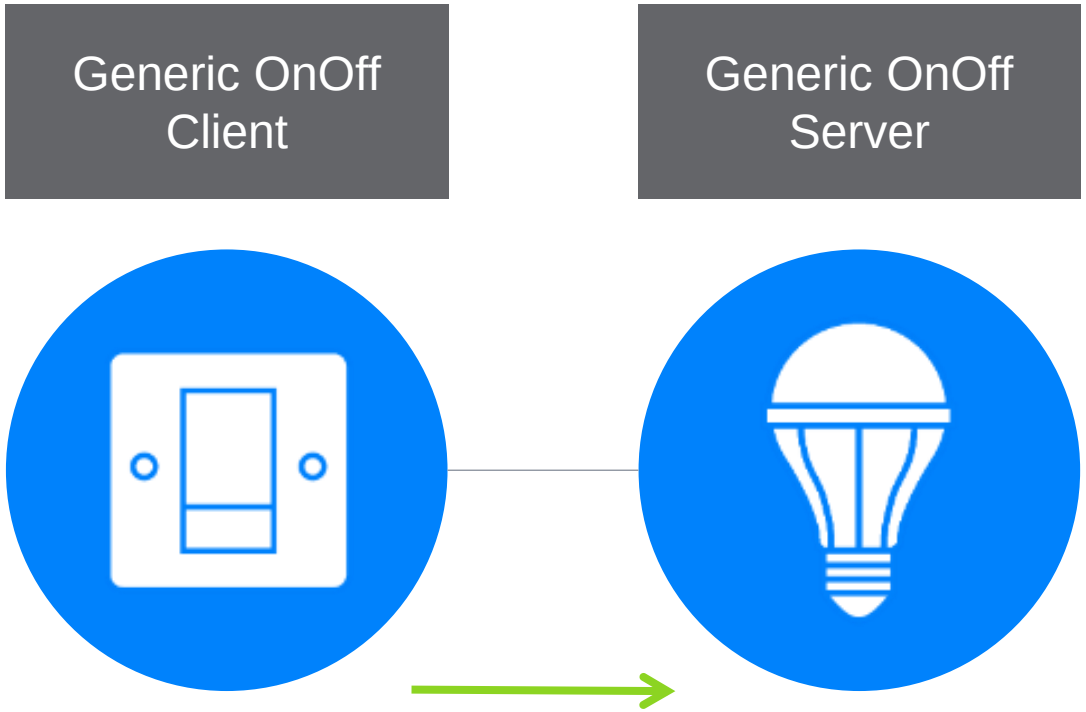


node composition

a node consists of an arrangement of

elements
models
states

each element has its own address



models

define node functionality

define states, messages, state transitions and behaviors

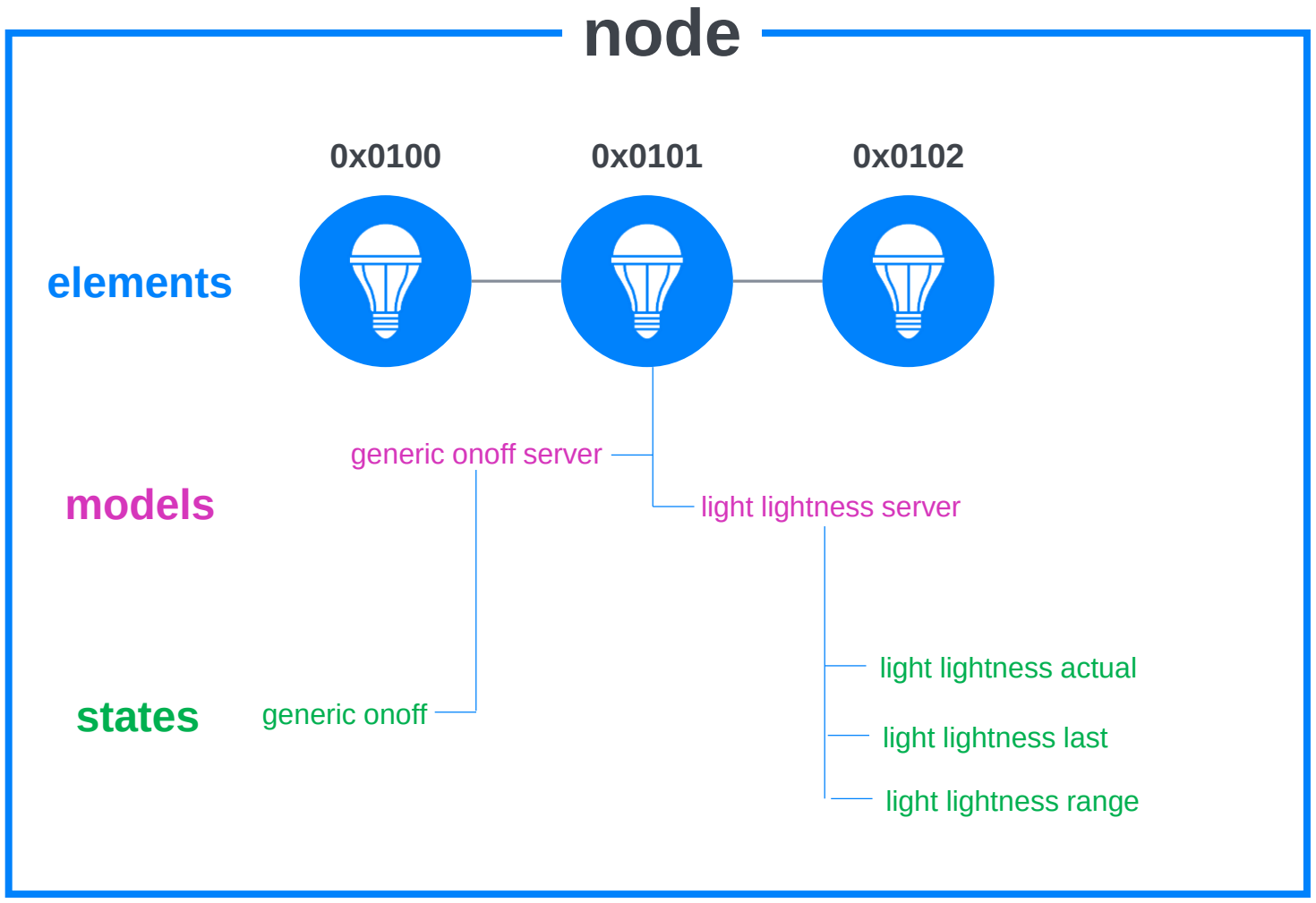
client, server and control types

generics such as onoff client and server

lighting, sensors, scenes & time

node composition

single node
3 elements
multiple models and states



Bluetooth Mesh

Demonstration

Bluetooth Mesh

Zephyr Code

Node Composition

```
// 1. Models Supported
static struct bt_mesh_model sig_models[] = {
  BT_MESH_MODEL_CFG_SRV(&cfg_srv),
  BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV, generic_onoff_op, &generic_onoff_pub,NULL),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_LEVEL_SRV, generic_level_op, &generic_level_pub,NULL)};

// 2. The models each element contains
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, sig_models, BT_MESH_MODEL_NONE),
};

// 3. The elements in this node (one only here)
static const struct bt_mesh_comp comp = {
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};
```

Models and Message Handlers

```
// 4. 16-bit message opcodes
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_GET BT_MESH_MODEL_OP_2(0x82, 0x01)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_SET BT_MESH_MODEL_OP_2(0x82, 0x02)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_SET_UNACK BT_MESH_MODEL_OP_2(0x82, 0x03)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_STATUS BT_MESH_MODEL_OP_2(0x82, 0x04)



// 5. mapping message opcodes to RX message handler functions
static const struct bt_mesh_model_op generic_onoff_op[] = {
    {BT_MESH_MODEL_OP_GENERIC_ONOFF_GET, 0, generic_onoff_get},
    {BT_MESH_MODEL_OP_GENERIC_ONOFF_SET, 2, generic_onoff_set},
    {BT_MESH_MODEL_OP_GENERIC_ONOFF_SET_UNACK, 2, generic_onoff_set_unack},
    BT_MESH_MODEL_OP_END,
};
```

RX Message Handling

```
// 6. RX message handler for generic onoff set unacknowledged
static void generic_onoff_set_unack(struct bt_mesh_model *model,
                                    struct bt_mesh_msg_ctx *ctx,
                                    struct net_buf_simple *buf) {
  // message payload is in a network buffer
  u8_t buflen = buf->len;
  // unpack using Zephyr network buffer API
  target_onoff_state = net_buf_simple_pull_u8(buf);
  u8_t tid = net_buf_simple_pull_u8(buf);
  transition_time = 0;
  // extract optional message parameters
  if (buflen > 4){
    transition_time = net_buf_simple_pull_u8(buf);
    delay = net_buf_simple_pull_u8(buf);
  }
  // process the transition
  k_work_submit(&onoff_set_work);
}
```

TX Message Sending

```
// 7. generic onoff status TX message producer
void generic_onoff_status(u8_t present_on_or_off, u16_t dest_addr, u8_t
transitioning, u8_t target_on_or_off, u8_t remaining_time){
  // create a network buffer for the message
  // 2 bytes for the opcode, 1 byte present onoff value
  // 2 optional bytes for target onoff and remaining time
  // 4 additional bytes for the TransMIC

  u8_t buflen = 7;

  if (transitioning == 1)  {
    buflen = 9;
  }

  NET_BUF_SIMPLE_DEFINE(msg, buflen);
```

TX Message Sending

```
// 7. generic onoff status TX message producer (cont)
 // create a message context (select keys, set dest addr, set TTL)
  struct bt_mesh_msg_ctx ctx = {
    .net_idx = net_idx,
    .app_idx = app_idx,
    .addr = dest_addr,
    .send_ttl = BT_MESH_TTL_DEFAULT };

  // initialise message buffer with opcode
  bt_mesh_model_msg_init(&msg, BT_MESH_MODEL_OP_GENERIC_ONOFF_STATUS);

  // populate message with fields
  net_buf_simple_add_u8(&msg, present_on_or_off);
  if (transitioning == 1)  {
    net_buf_simple_add_u8(&msg, target_on_or_off);
    net_buf_simple_add_u8(&msg, remaining_time);
  }
```

TX Message Sending

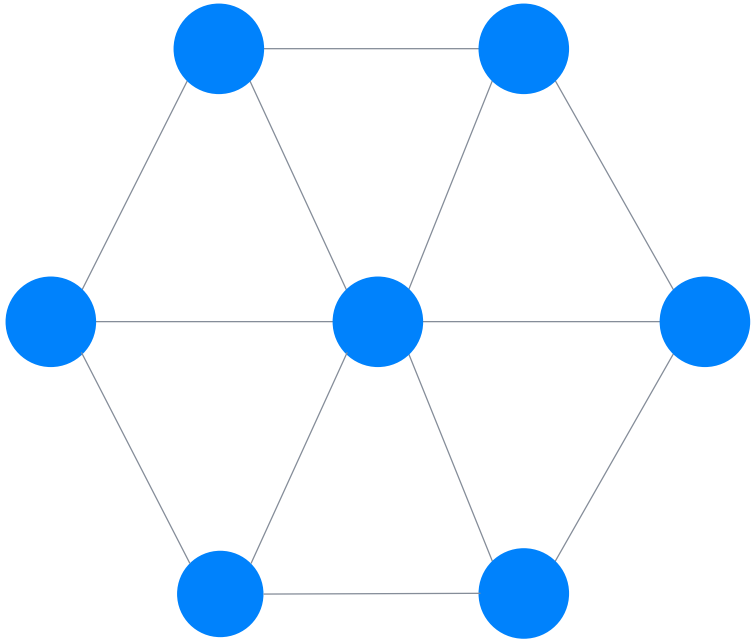
```
// 7. generic onoff status TX message producer (cont)

    // send the message
    if (bt_mesh_model_send(&sig_models[3], &ctx, &msg, NULL, NULL)){
        printk("Unable to send generic onoff status message\n");
    }

    // job done!
    printk("onoff status message %d sent\n", present_on_or_off);
}
```

Bluetooth Mesh

Security



Device is now a
node on the network

devices and network membership

Bluetooth mesh networks are secure

only members of the same network can talk to each other

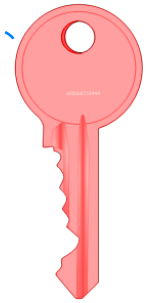
a security process called **provisioning** makes a device a member of a network

Bluetooth mesh: Security

- Mandatory
- Encryption and authentication
- Separate security for network and each application
- Area isolation
- Message obfuscation
- Protection from replay and trashcan attacks
- Secure device provisioning

devkey is bound to all
netkeys known to a node

appkey is bound to a netkey



network key (netkey)
origin: provisioning
use: derivation of other keys

application key (appkey)
origin: created by the config. client and provided to nodes after provisioning
use: secures application data at the upper transport layer
Bound to one or more models.

device key (devkey)
origin: established during provisioning
use: secures communication between the config. client and individual node

encryption key
origin: derived from netkey using the k2 function
use: secures data at the network layer

privacy key
origin: derived from netkey using the k2 function
use: obfuscation of network header information

ref: mesh profile 1.0 section 2.3.9.1

Bluetooth Mesh

Where next?

Bluetooth SIG Resources - Reading Material

Mesh Resources

- Mesh Networking Specifications
- The Case for Bluetooth Mesh
- Paving the Way for Smart Lighting
- Bluetooth Mesh FAQ
- Bluetooth Mesh Performance Study (Ericsson)
- Bluetooth Mesh Overview
- Bluetooth Mesh Technology Overview
- Related Mesh Blog Posts
- Bluetooth Mesh Glossary of Terms
- Webinar: What Makes Bluetooth Mesh So Disruptive?

Bluetooth SIG Resources - hands-on education

Bluetooth Mesh Developer Study Guide

Mesh Proxy Kit

questions?

Twitter: @bluetooth_mdw

Bluetooth®

Unthinkably Connected