Developing Bluetooth Mesh with Rust

Dejan Bosanac
What we’ll discuss today

- What is Bluetooth Mesh
- Current state
- Why Rust?
- Rust Mesh stack
- In practice
What's Bluetooth Mesh

"... nodes connect directly, dynamically and non-hierarchically to as many other nodes as possible and cooperate with one another to efficiently route data to and from clients."

- mesh network based on BLE technology.
- Managed flooding principle
- Publish/subscribe model
How does it work

Nodes

Source: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/
How does it work

Stack

Application Layer
- Bluetooth Mesh Models

Networking Layer
- Bluetooth Mesh Profile

Radio Layer
- Bluetooth Low Energy

Source: https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/
How does it work

Models
How does it work

Networking

- Each element has a unicast address
- Send and receive messages between client and server models
- Group and virtual addresses allow more complex topologies
- Messages are double-encrypted: network key and application key
How does it work

Provisioning

- Provisioner: special device that manages network and adds new nodes
- Manage network key
- Add nodes to the network (and manage keys)
- Setting addresses
Use cases

- Extended range and device number
- More flexible topologies
- Low energy
- Works on existing hardware
Comparisons

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>Bluetooth Mesh</th>
<th>Zigbee</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth Mesh Models</td>
<td>Zigbee Cluster Library</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Networking Layer</td>
<td>Bluetooth Mesh Profile</td>
<td>Zigbee 3.0</td>
<td>Thread</td>
</tr>
<tr>
<td>Radio Layer</td>
<td>Bluetooth Low Energy</td>
<td>IEEE 802.15.4</td>
<td>IEEE 802.15.4</td>
</tr>
</tbody>
</table>
End goal

- Build full stack: embedded, linux and cloud in Rust
- Allow easy application building (both devices and cloud)
- Allow easy network and device provisioning and management

Diagram:

- ○ = Sensor
- □ = Gateway

Cloud
Current state

Embedded

- Open source
  - Zephyr
    - https://docs.zephyrproject.org/3.1.0/samples/bluetooth/mesh/README.html

- Vendor supported SDKs
  - https://www.nordicsemi.com/Products/Development-software/nrf5-sdk-for-mesh
Current state

Linux

- BlueZ D-Bus Mesh API description
  - Using D-Bus to send messages between daemon and applications
Current state

Linux Daemon

```
sudo dnf install -y bluez-mesh

sudo systemctl disable bluetooth
sudo systemctl stop bluetooth

sudo systemctl enable bluetooth-mesh
sudo systemctl start bluetooth-mesh

sudo /usr/libexec/bluetooth/bluetooth-meshd --config ${PWD}/config --storage ${PWD}/lib --debug
```
Current state

Linux Provisioner

$ mesh-cfgclient
[mesh-cfgclient]# discover-unprovisioned on
Unprovisioned scan started
Scan result:
  rssi = -39
  UUID = 0EF817B94FA04859A4F7C80312CD724E
  OOB = A040

[mesh-cfgclient]# provision 0EF817B94FA04859A4F7C80312CD724E
Provisioning started
Assign addresses for 1 elements
Provisioning done:
Mesh node:
  UUID = 0EF817B94FA04859A4F7C80312CD724E
  primary = 00c4

  elements (1):
Current state

Linux Application

blemesh.mesh_net = dbus.Interface(blemesh.bus.get_object(blemesh.MESH_SERVICE_NAME, "/org/bluez/mesh"), blemesh.MESH_NETWORK_IFACE)

blemesh.app = blemesh.Application(blemesh.bus)
blemesh.app.set_agent(blemesh.Agent(blemesh.bus))

first_ele = blemesh.Element(blemesh.bus, 0x00)
second_ele = blemesh.Element(blemesh.bus, 0x01)

first_ele.add_model(blemesh.OnOffServer(0x1000))  # Register OnOff Server model on element 0
first_ele.add_model(blemesh.BurrBoardSensorServer(0x1100))

first_ele.add_model(blemesh.SampleVendor(0x0001))  # Register Vendor model on element 0
second_ele.add_model(blemesh.OnOffClient(0x1001))  # Register OnOff Client model on element 1
second_ele.add_model(blemesh.SensorClient(0x1102))

blemesh.app.add_element(first_ele)
blemesh.app.add_element(second_ele)

blemesh.set_token(token)
blemesh.attach(blemesh.token)
blemesh.mainloop.run()
End goal

- Build full stack: embedded, linux and cloud in Rust
- Allow easy application building (both devices and cloud)
- Allow easy network and device provisioning and management

Cloud = Sensor
□ = Gateway

Cloud

Sensor

Gateway

Device
Why Rust?

Ideal for system programming ...
- Performance: Statically compiled and strongly typed
- Reliability: Memory safety (without runtimes or VMs)
- Productivity: Modern features and tooling
btmesh crate

https://github.com/drogue-iot/btmesh

- Define basic traits for all mesh layers
- no-std so it can be used in embedded
#[derive(Clone, Debug, Default)]
pub struct Temperature(f32);

impl SensorConfig for SensorModel {
    type Data = Temperature;
    const DESCRIPTORS: &'static [SensorDescriptor] = &[SensorDescriptor::new(PropertyId(0x4F), 1)];
}

impl SensorData for Temperature {
    fn decode(&mut self, id: PropertyId, params: &[u8]) -> Result<(), ParseError> {
        if id.0 == 0x4F {
            self.0 = params[0] as f32 / 2.0;
            Ok(())
        } else {
            Err(ParseError::InvalidValue)
        }
    }

    fn encode<const N: usize>(
        &self, _id: PropertyId, xmit: &mut heapless::Vec<u8, N>,
    ) -> Result<(), InsufficientBuffer> {
        xmit.extend_from_slice(&self.0.to_le_bytes()).map_err(|_| InsufficientBuffer)?;
        Ok(())
    }
}
Rust Embedded

https://github.com/rust-embedded/wg

Official working group of the Rust language

- 16 kB - 512 kB RAM
- 128 kB - 2 MB ROM/Flash
- No operating system
- No memory allocator
Embassy / Drogue Device

https://embassy.dev/

- Components from the Embedded Rust ecosystem
  - Embassy: Scheduler, hardware abstractions, time-keeping
  - Board Support Packages (BSP) for selected boards
  - Examples

- Hardware support
  - STM32, nRF, Raspberry Pi Pico, ESP-32
Embassy / Drogue Device


- Firmware update
- Communication
  - TCP, HTTP
  - Bluetooth Mesh
  - Bluetooth Low Energy
  - LoRaWAN
defmt::info!("Read sensor data: {:?}", result);
let message = SensorMessage::Status(SensorStatus::new(result));
match ctx.publish(message).await {
  Ok(_) => {
    defmt::info!("Published sensor reading");
  }
  Err(e) => {
    defmt::warn!("Error publishing: {:?}", e);
  }
}
Bluer

https://github.com/bluez/bluer

Provides the official Rust interface to the Linux Bluetooth protocol stack

- Adapters/Devices
- GATT
- Bluetooth Low Energy
- Bluetooth Mesh (in progress)
Bluer

https://github.com/bluez/bluer

- Runs on Tokio runtime ([https://tokio.rs/](https://tokio.rs/))
- Uses dbus crate ([https://crates.io/crates/dbus](https://crates.io/crates/dbus)) to communicate with meshd
- Use btmesh crate for mesh traits
Bluer

Target architecture
Bluer Mesh support

https://github.com/bluez/bluer/pull/60

```rust
match SensorClient::parse(&received.opcode, &received.parameters) {
    Some(message) => {
        log::trace!("Received {:?}", message);
    },
    None => {} }

let data = serde_json::to_string(&message)??

let message = mqtt::Message::new(topic, data.as_bytes(), 1);
mqtt_client.publish(message).await;
```
Drogue Cloud

https://book.drogue.io/drogue-cloud/dev/

IoT friendly APIs and services for the cloud. Connecting your devices with applications. Solving common IoT tasks in the middle.

- Device registry
- IoT connectivity
- Digital twin
- Firmware Updates
- Scalability
if let Ok(Some(SensorMessage::Status(mut status))) = SensorClient::parse(&opcode, parameters) {

    log::info!("Received sensor status {:?}", status);
    // Temperature is in half degrees
    status.data.temperature /= 2;
    return Some(json!(
        "sensor": {
            "Payload":
                serde_json::to_value(&status.data).unwrap(),
            "location": location,
        }
    ));
}
Workshop architecture

Workshop architecture

Embedded

micro:bit v2

application

drogue-device  btmesh

embassy  embedded-hal  Bluetooth Radio
Workshop architecture

Linux

- Intel NUC
- MicroShift
- Bluetooth
- btmesh-gateway
- Open Cluster Management (OCM)
- Drogue IoT

Infrastructures
Data
Workshop results
Communities

Optional subheading

- https://drogue.io
  - https://matrix.to/#/#drogue-iot:matrix.org
- https://embassy.dev/
- https://github.com/bluez/bluer
Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.