Optional section marker or title

Developing Bluetooth Mesh with Rust

Dejan Bosanac

Red Hat

Agenda

What we'll discuss today

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

What's Bluetooth Mesh

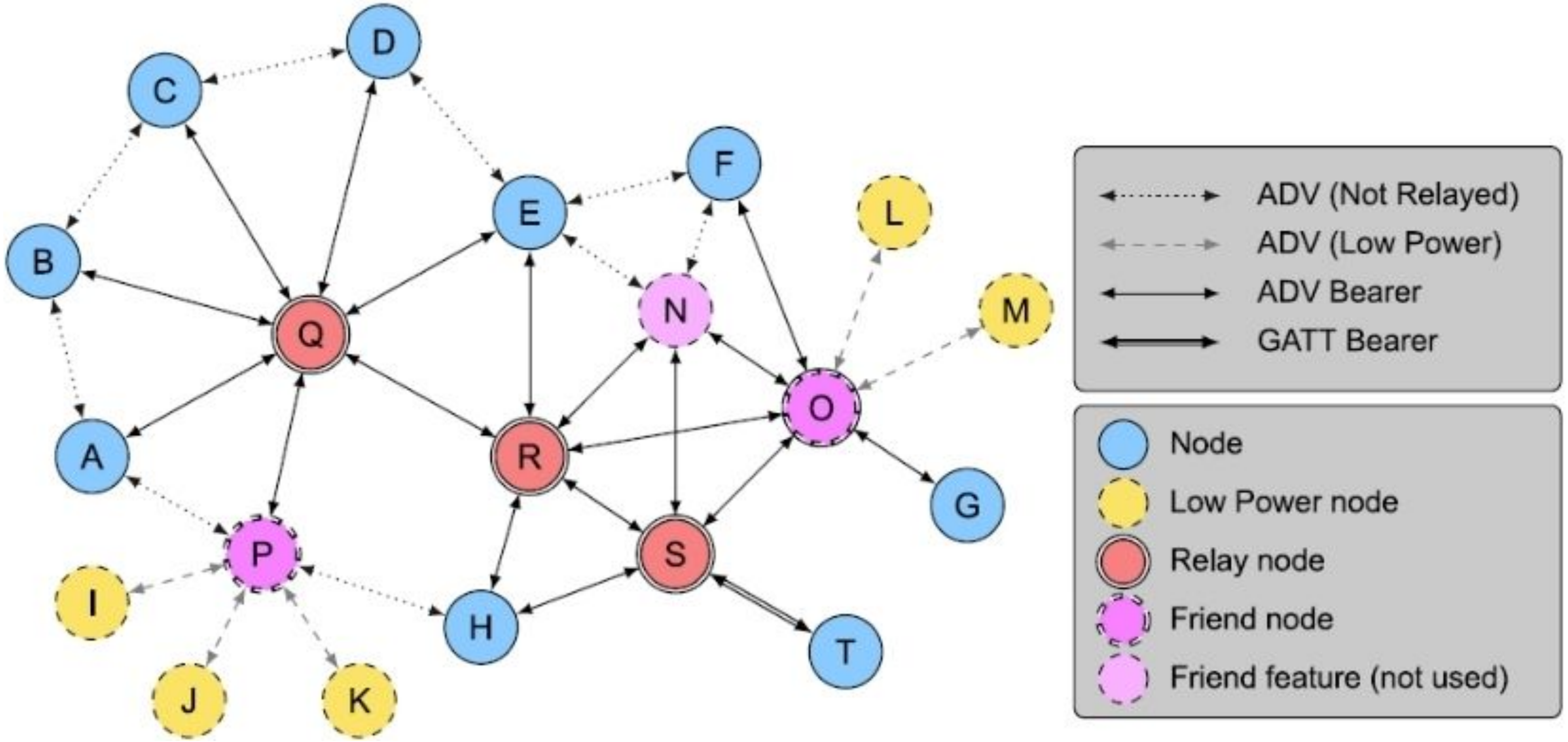
"... nodes connect directly, dynamically and non-hierarchically to as many other nodes as possible and cooperate with one another to efficiently route data to and from clients."

- mesh network based on BLE technology.
- Managed flooding principle
- Publish/subscribe model

Source: https://en.wikipedia.org/wiki/Mesh_networking

How does it work

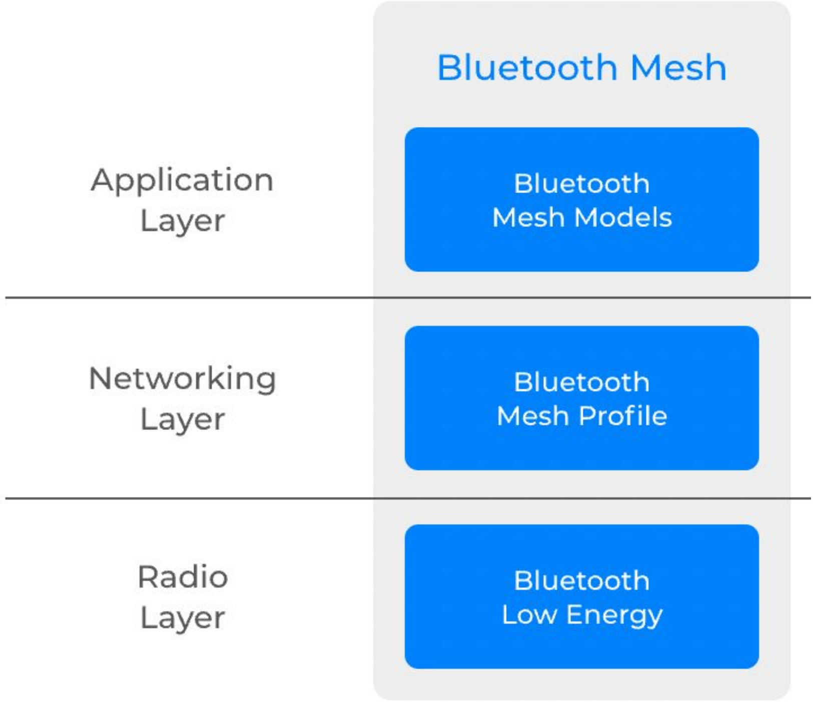
Nodes



Source: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/

How does it work

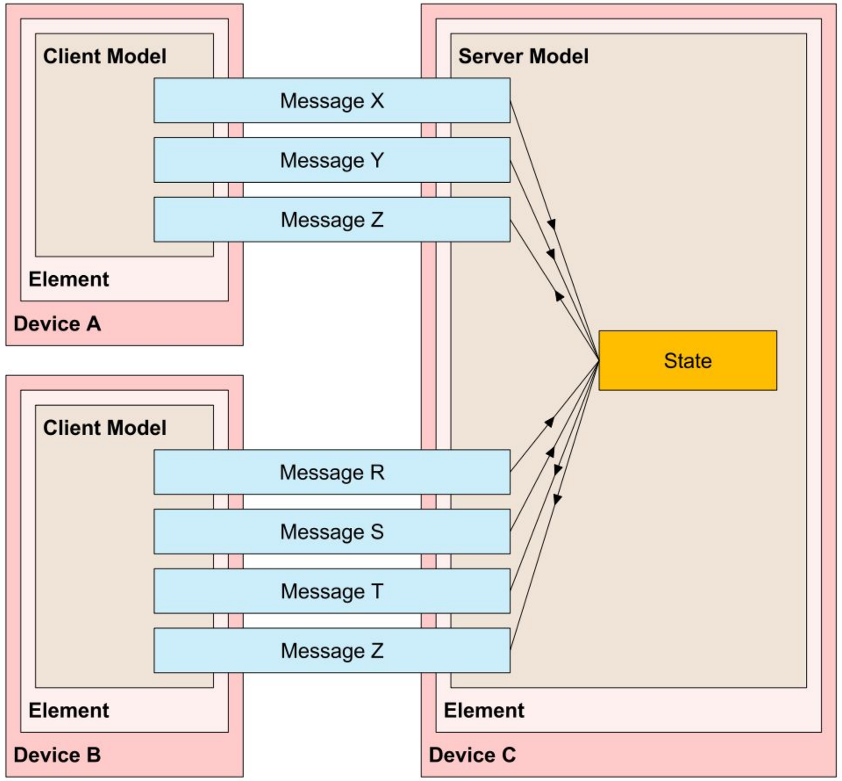
Stack



Source: https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/

How does it work

Models



Source: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/

How does it work

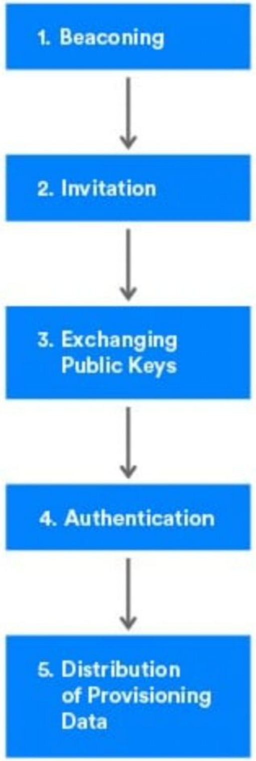
Networking

- 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

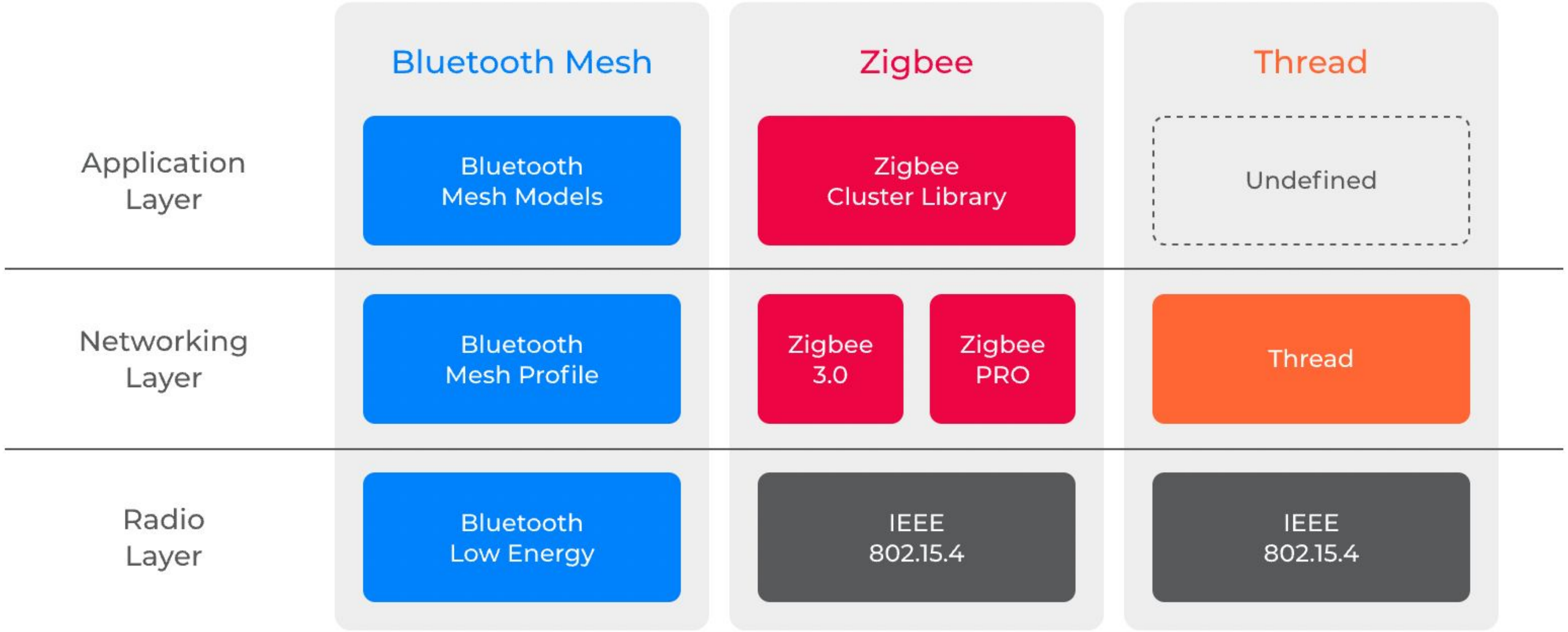

1. Beaconing
2. Invitation
3. Exchanging Public Keys
4. Authentication
5. Distribution of Provisioning Data

The provisioning process

Use cases

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

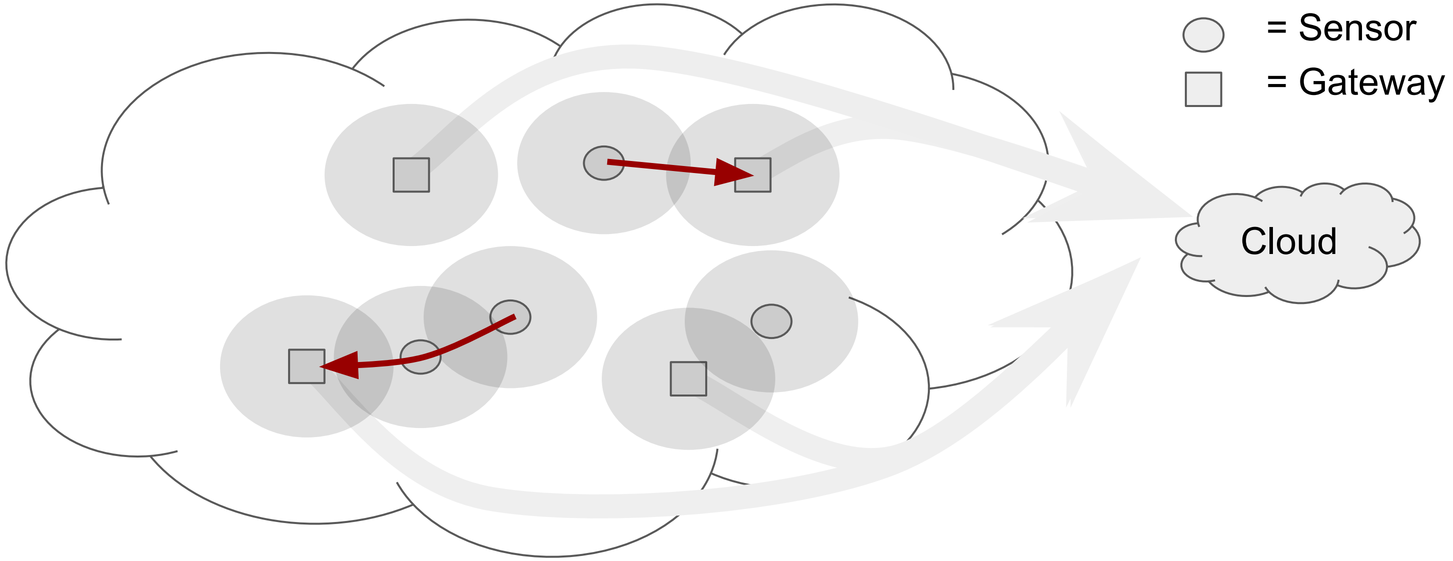
Comparisons



	Bluetooth Mesh	Zigbee	Thread
Application Layer	Bluetooth Mesh Models	Zigbee Cluster Library	Undefined
Networking Layer	Bluetooth Mesh Profile	Zigbee 3.0 Zigbee PRO	Thread
Radio Layer	Bluetooth Low Energy	IEEE 802.15.4	IEEE 802.15.4

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



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
  - https://www.st.com/en/embedded-software/x-cube-blemesh1.html

Current state

Linux

- http://www.bluez.org/ – Official Linux Bluetooth protocol stack
- BlueZ D-Bus Mesh API description
  - https://github.com/bluez/bluez/blob/master/doc/mesh-api.txt
  - 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):
```

Source:
https://blog.drogue.io/bluetooth-mesh/

Current state

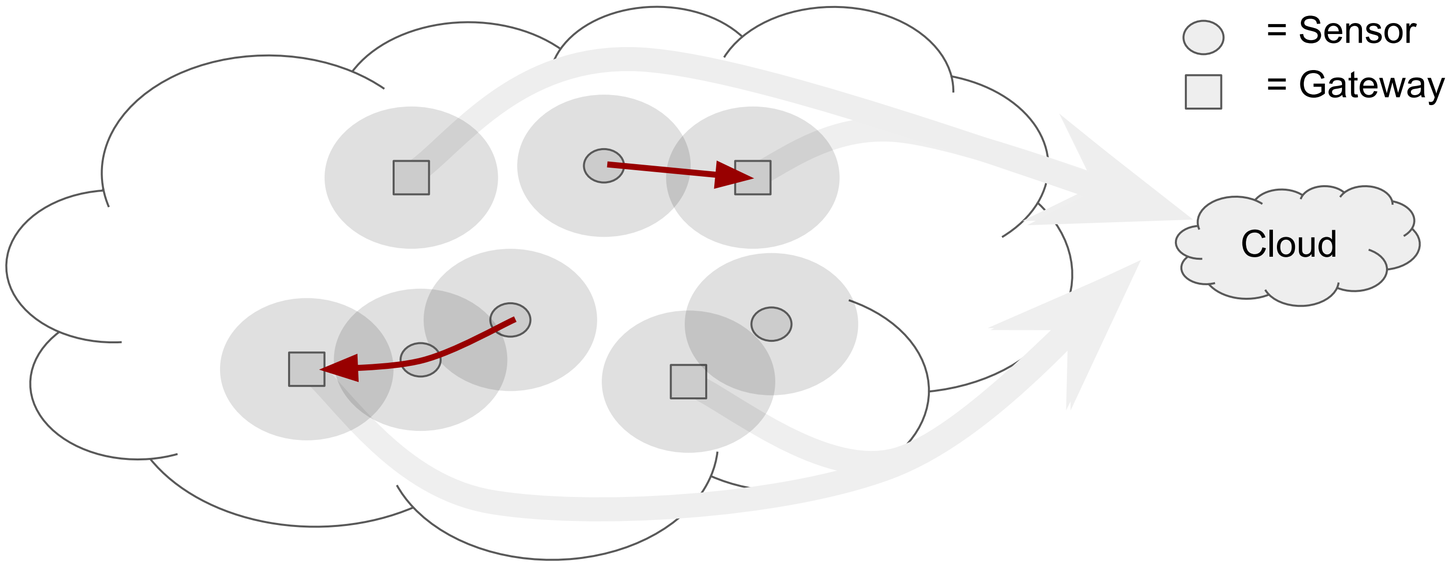
Linux Application

```
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()
```

Source:
https://www.bluetooth.com/bluetooth-resources/bluetooth-internet-gateways/

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



○ = Sensor

□ = Gateway

Cloud

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

btmesh crate

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

```
#[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, _: 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

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

- Firmware update
- Communication
  - TCP, HTTP
  - Bluetooth Mesh
  - Bluetooth Low Energy
  - LoRaWAN

Embedded Mesh example

```
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)

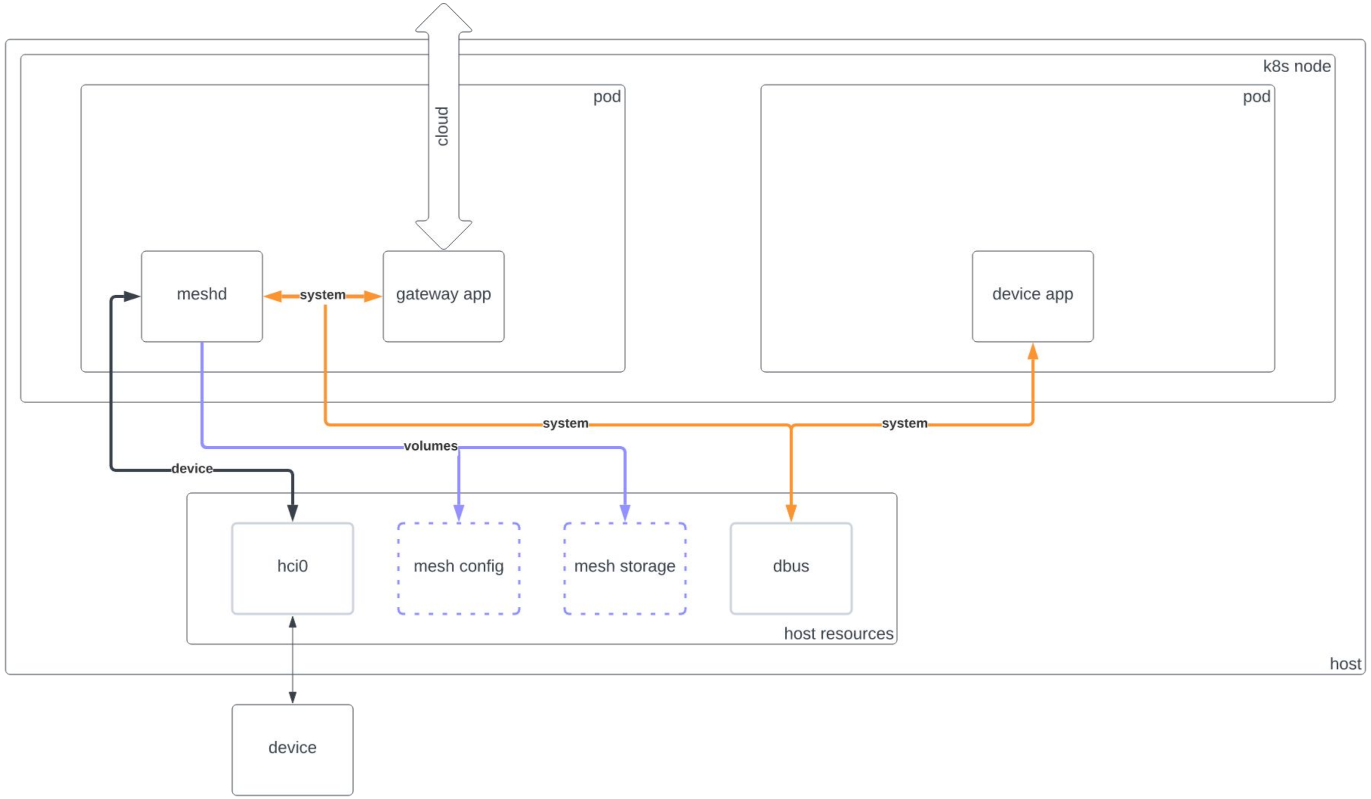
Bluer

https://github.com/bluez/bluer

- Runs on Tokio runtime (https://tokio.rs/)
- Uses dbus crate (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

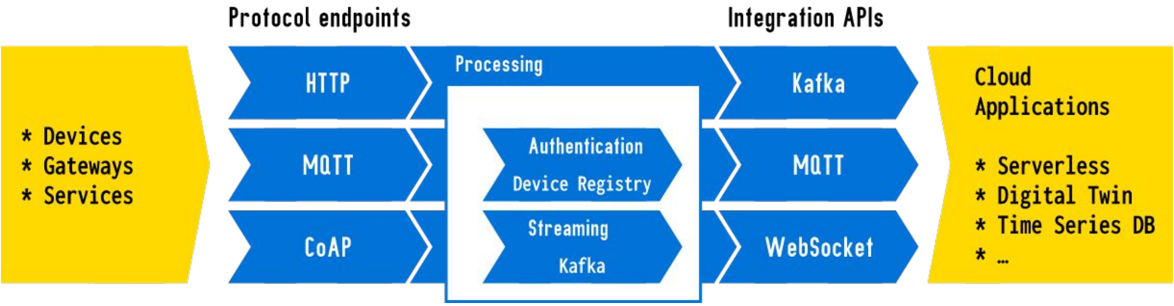
```
match SensorClient::parse(&received.opcode, &received.parameters) {
      Some(message) => {
          log::trace!("Received {:?}", message);
      },
      None => {}
}
let data = serde_json::to_string(&message)?;


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



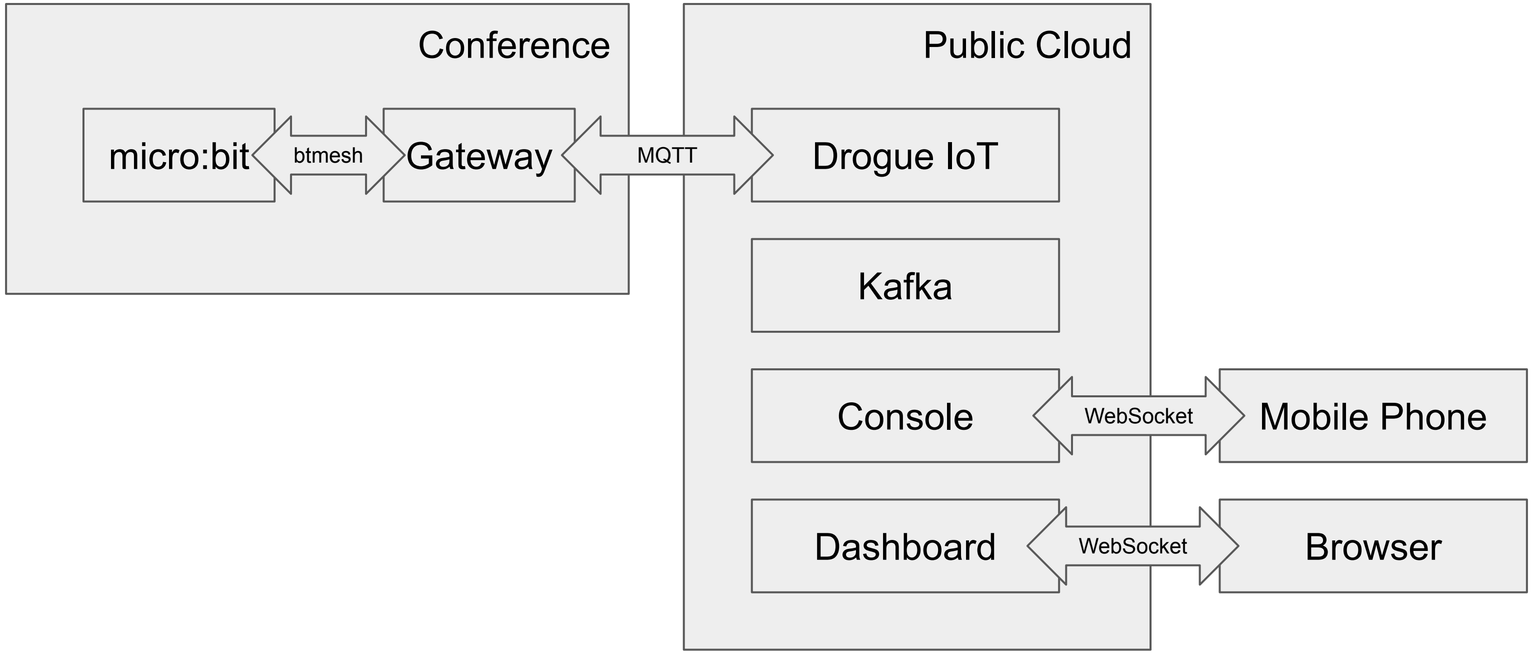
Drogue Cloud

Payload converter

```
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,
        }
    }));
}
```

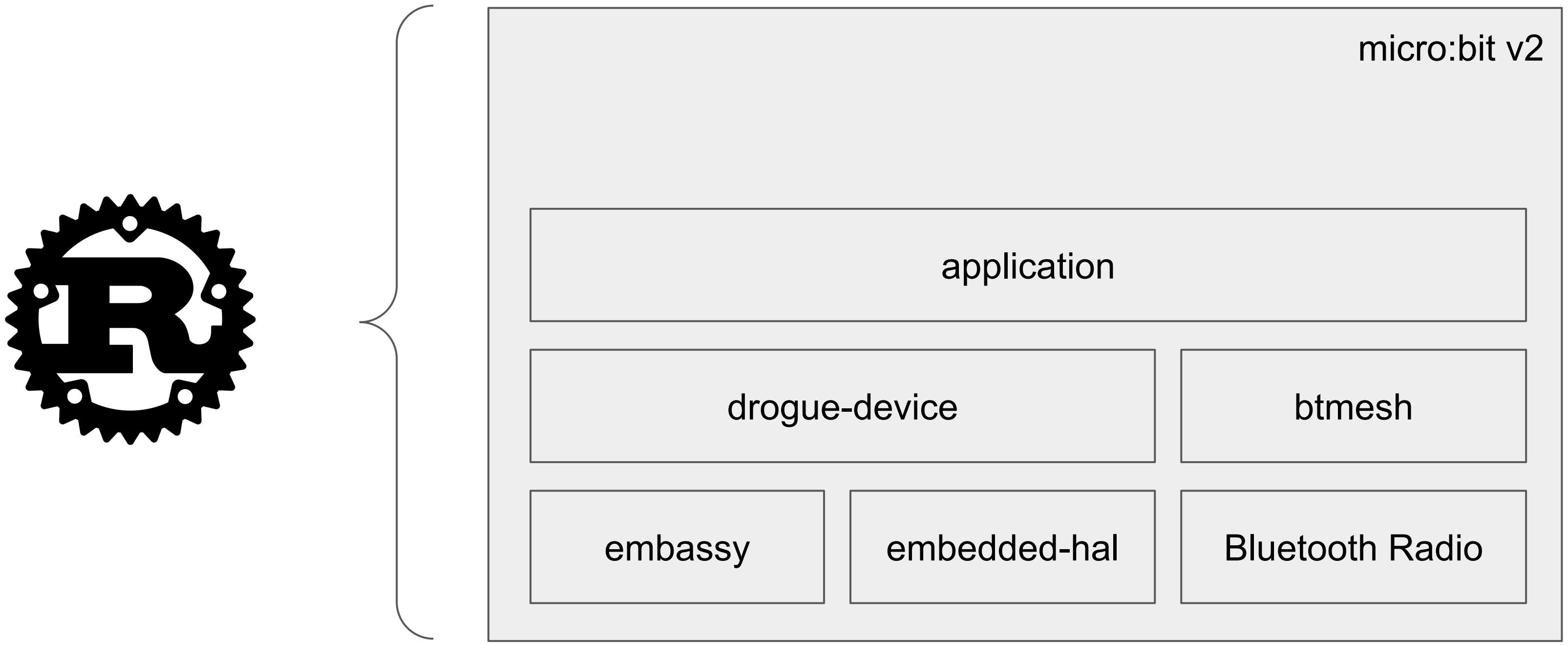
Workshop architecture

https://github.com/Eclipse-IoT/eclipsecon-2022-hackathon



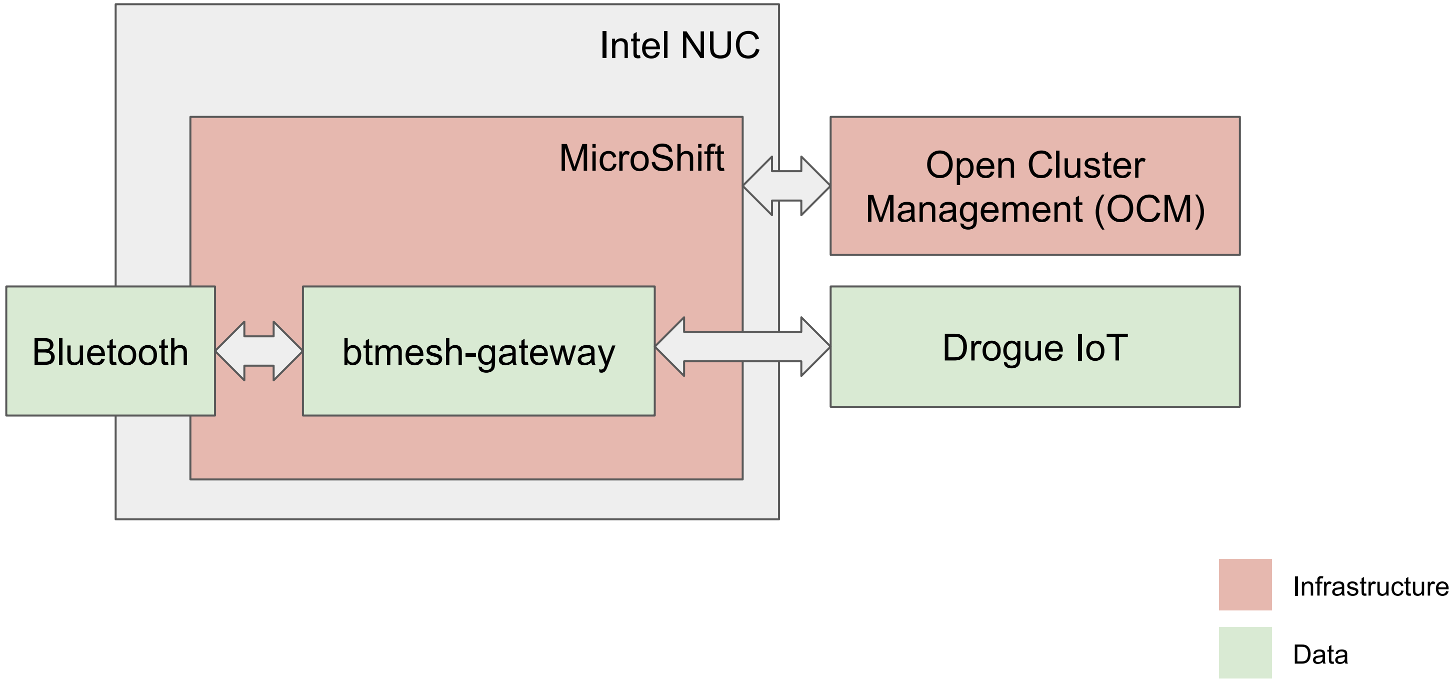
Workshop architecture

Embedded



Workshop architecture

Linux



Workshop results

Communities

Optional subheading

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

Optional section marker or title

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.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat