NuttX, Drones, and the Internet of Things (IoT)

Anthony Merlino
Verge Aero
July 17, 2019
Who am I?

- Anthony Merlino
- New Jersey,/Philadelphia USA
- CTO & Co-Founder of Verge Aero

Who is Verge Aero?

- Very small team of engineers/creatives passionate about robotics and automation
- Currently focused on creating a scalable drone entertainment solution - synchronized light shows across swarms of hundreds of drones.
Why NuttX?

- Focus on standards
  - Posix
  - Linux-compatibility (where possible)
- RTOS
- Vendor Neutral - many MCUs are supported
- Most applications can be written and tested entirely in Linux first
- C++ support
- Full network stack
What do 241 instances of NuttX look like?
Thank you Greg!
Thank you NuttX!
Thank you PX4!
Internet of Things (IoT) Protocols

- IEEE 802.15.4
- WiFi (IEEE 802.11)
- Bluetooth
- BLE
- LoRa
- Zigbee (IEEE 802.15.4)
- Thread (IEEE802.15.4 + 6LoWPAN)
- 6LoWPAN
- CoAP
- MQTT
IEEE 802.15.4

- **MAC/PHY Layer**
  - OSI Physical and Data Link
- **Many PHY Layers**
  - 2.4GHz
  - Sub-1Ghz
  - UWB
- **Basic Functionality**
  - Addressing (EUI64, 16-bit short address)
  - Acknowledgement handling
  - PAN Management
  - Types of nodes: PAN Coordinator, Coordinator, Device
- **Advanced Functionality**
  - Beacon-Enabled Networking
  - Ranging

---

<table>
<thead>
<tr>
<th>OSI model</th>
<th>Layer</th>
<th>Protocol data unit (PDU)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>7</td>
<td>Application</td>
<td>High-level APIs, including resource sharing, remote file access</td>
</tr>
<tr>
<td>layers</td>
<td>6</td>
<td>Presentation Data</td>
<td>Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Session</td>
<td>Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Transport Segment, Datagram</td>
<td>Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing</td>
</tr>
<tr>
<td>Media</td>
<td>3</td>
<td>Network Packet</td>
<td>Structuring and managing a multi-node network, including addressing, routing and traffic control</td>
</tr>
<tr>
<td>layers</td>
<td>2</td>
<td>Data link Frame</td>
<td>Reliable transmission of data frames between two nodes connected by a physical layer</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Physical Symbol</td>
<td>Transmission and reception of raw bit streams over a physical medium</td>
</tr>
</tbody>
</table>

MAC consists of 2 “services”
- **MLME** - MAC Layer Management Entity
- **MCPS** - MAC Data Service

**Request/Response** “primitives” are requests from the next highest layer for the MAC to do something

**Indication/Confirmation** “primitives” are notification from the MAC layer to the next highest layer

<table>
<thead>
<tr>
<th>Name</th>
<th>Request</th>
<th>Indication</th>
<th>Response</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLME-ASSOCIATE</td>
<td>8.2.3.1</td>
<td>8.2.3.2</td>
<td>8.2.3.3</td>
<td>8.2.3.4</td>
</tr>
<tr>
<td>MLME-BEAON-NOTIFY</td>
<td>8.2.5.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-BEACON</td>
<td>8.2.18.1</td>
<td></td>
<td></td>
<td>8.2.18.2</td>
</tr>
<tr>
<td>MLME-CALIBRATE</td>
<td>8.2.17.1</td>
<td></td>
<td>8.2.17.2</td>
<td></td>
</tr>
<tr>
<td>MLME-COMM-STATUS</td>
<td>8.2.5.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-DBS</td>
<td>8.2.23.1</td>
<td>8.2.23.2</td>
<td>8.2.23.3</td>
<td>8.2.23.4</td>
</tr>
<tr>
<td>MLME-DA</td>
<td>8.2.24.1</td>
<td>8.2.24.2</td>
<td>8.2.24.3</td>
<td></td>
</tr>
<tr>
<td>MLME-DISASSOCIATE</td>
<td>8.2.4.1</td>
<td>8.2.4.2</td>
<td>8.2.4.3</td>
<td></td>
</tr>
<tr>
<td>MLME-DPS</td>
<td>8.2.15.1</td>
<td>8.2.15.3</td>
<td>8.2.15.2</td>
<td></td>
</tr>
<tr>
<td>MLME-GET</td>
<td>8.2.6.1</td>
<td></td>
<td></td>
<td>8.2.6.2</td>
</tr>
<tr>
<td>MLME-GTS</td>
<td>8.2.7.1</td>
<td>8.2.7.3</td>
<td></td>
<td>8.2.7.2</td>
</tr>
<tr>
<td>MLME-IE-NOTIFY</td>
<td>8.2.5.3</td>
<td></td>
<td>8.2.8.2</td>
<td></td>
</tr>
<tr>
<td>MLME-ORPHAN</td>
<td>8.2.8.1</td>
<td>8.2.8.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-PHY-OP-SWITCH*</td>
<td>8.2.20</td>
<td>8.2.21</td>
<td></td>
<td>8.2.22.3</td>
</tr>
<tr>
<td>MLME-POLL</td>
<td>8.2.14.1</td>
<td></td>
<td></td>
<td>8.2.14.2</td>
</tr>
<tr>
<td>MLME-RESET</td>
<td>8.2.9.1</td>
<td></td>
<td></td>
<td>8.2.9.2</td>
</tr>
<tr>
<td>MLME-RIT-REQ</td>
<td>8.2.25.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-RIT-RES</td>
<td>8.2.25.2</td>
<td>8.3</td>
<td>8.2.25.4</td>
<td></td>
</tr>
<tr>
<td>MLME-RX-ENABLE</td>
<td>8.2.10.1</td>
<td></td>
<td></td>
<td>8.2.10.2</td>
</tr>
<tr>
<td>MLME-SCAN</td>
<td>8.2.11.1</td>
<td></td>
<td>8.2.11.2</td>
<td></td>
</tr>
<tr>
<td>MLME-SET</td>
<td>8.2.6.3</td>
<td></td>
<td>8.2.6.4</td>
<td></td>
</tr>
<tr>
<td>MLME-START</td>
<td>8.2.12.1</td>
<td></td>
<td>8.2.12.2</td>
<td></td>
</tr>
<tr>
<td>MLME-SYNC</td>
<td>8.2.13.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-SYNC-LOSS</td>
<td>8.2.13.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-SOUNDING</td>
<td>8.2.16.1</td>
<td></td>
<td>8.2.16.2</td>
<td></td>
</tr>
</tbody>
</table>
Frame Types

- 4 Primary Frame Types
  - Beacon
  - Data
  - ACK
  - MAC Command

### Table 7-1—Values of the Frame Type field

<table>
<thead>
<tr>
<th>Frame type value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b2 b1 b0</td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>Beacon</td>
</tr>
<tr>
<td>001</td>
<td>Data</td>
</tr>
<tr>
<td>010</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>011</td>
<td>MAC command</td>
</tr>
<tr>
<td>100</td>
<td>Reserved</td>
</tr>
<tr>
<td>101</td>
<td>Multipurpose</td>
</tr>
<tr>
<td>110</td>
<td>Fragment or Frak&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>111</td>
<td>Extended</td>
</tr>
</tbody>
</table>
IEEE 802.15.4 in NuttX

- **Kernel**
  - Software MAC Layer
    - `wireless/ieee802154/mac802154*`
  - Phy/Lower-level MAC
    - `drivers/wireless/ieee802154/*`

- **Apps:**
  - `libmac` - helper library that wraps socket/char driver calls to call MAC functionality.
  - `i8sak` - CLI for testing/performing MAC calls (set short address, set channel, associate, etc.)
  - `i8shark` - Wireshark ZEP (Zigbee Encapsulation Protocol)
802.15.4 Radio Driver/Low-level MAC

- Radio driver responsible for all PHY functionality and some MAC functionality; primarily anything related to timing
- MAC functionality required
  - Frame Check Sequence (FCS) injection/validation
  - Frame filtering - Incoming frame has a valid FCS, is not an ACK, and the frame is destined for either the node's short or extended address
  - Clear Channel Assessment (CCA)
  - Carrier Sense Multiple Access (CSMA)
  - Timing of TX
6LoWPAN (RFC 4944)

- IPv6 over IEEE 802.15.4
- Fragmentation
  - IEEE 802.15.4 frames much smaller than IPv6 packets
- Compression
  - IPv6 header compression
    - IPv6 address derived from EUI-64 or Short Address
    - IPv6 Prefix can be compressed further if necessary using shared address contexts
  - UDP header compression
How the pieces stack

User Application
Character Driver

mac802154_device
mac802154
Radio Driver (ex. MRF24J40)

User Application
Socket Interface

mac802154_netdev
mac802154
Radio Driver (ex. MRF24J40)

User Application
Socket Interface

6LoWPAN

xbee_netdev

Radio Driver (ex. XBee)
Board bring-up example

```c
/* Initialize and register the SPI MR24J40 device */
radio = mrf24j40_init(spi, &priv->dev);
if (radio == NULL)
    wlerr("ERROR: Failed to initialize SPI bus \%d\n", priv->spidev);
    return -ENODEV;

/* Create a 802.15.4 MAC device from a 802.15.4 compatible radio device. */
mac = mac802154_create(radio);
if (mac == NULL)
    wlerr("ERROR: Failed to initialize IEEE802.15.4 MAC\n");
    return -ENODEV;

#ifndef CONFIG_IEEE802154_NETDEV
/* Use the IEEE802.15.4 MAC Interface instance to create a 6LoWPAN
 * network interface by wrapping the MAC interface instance in a
 * network device driver via mac802154dev_register().
 */
ret = mac802154netdev_register(mac);
if (ret < 0)
    { wlerr("ERROR: Failed to register the MAC network driver wpam\d: \%d\n", \%
        0, ret);
        return ret;
    }
#endif

#ifndef CONFIG_IEEE802154_MACDEV
/* If you want to call these APIs from userspace, you have to wrap the MAC
 * interface in a character device via mac802154dev_register().
 */
ret = mac802154dev_register(mac, 0);
if (ret < 0)
    { wlerr("ERROR: Failed to register the MAC character driver /dev/ieee\d: \%d\n", \%
        0, ret);
        return ret;
    }
#endif
```
Application Protocols

- MQTT - Message Queue Telemetry Transfer - TCP/IP
  - UDP
  - Designed from HTTP as starting point
  - Everything exposed via a URI

- Constrained Application Protocol (CoAP) - RFC 7252
  - Designed from HTTP as starting point
  - Everything exposed via a URI

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>GET</td>
<td>[RFC7252]</td>
</tr>
<tr>
<td>0.02</td>
<td>POST</td>
<td>[RFC7252]</td>
</tr>
<tr>
<td>0.03</td>
<td>PUT</td>
<td>[RFC7252]</td>
</tr>
<tr>
<td>0.04</td>
<td>DELETE</td>
<td>[RFC7252]</td>
</tr>
</tbody>
</table>

Table 5: CoAP Method Codes

Figure 4: Two GET Requests with Piggybacked Responses

“Smart” Light Demo