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

Anthony Merlino
Verge Aero
July 17, 2019

VERGE AERO

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, Devic
- Advanced Functionality
  - Beacon-Enabled Networking
  - Ranging

OSI model

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

https://en.wikipedia.org/wiki/OSI_model

Services and Primitives

- 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 8-74—MCPS-SAP primitives

MCPS-SAP primitive	Request	Confirm	Indication
MCPS-DATA	8.3.1	8.3.2	8.3.3
MCPS-PURGE	8.3.4♦	8.3.5♦	—

Table 8-1—Summary of the primitives accessed through the MLME-SAP

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

Frame Types

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

Table 7-1—Values of the Frame Type field

Frame type value b2 b1 b0	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command
100	Reserved
101	Multipurpose
110	Fragment or Frak[a]
111	Extended

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

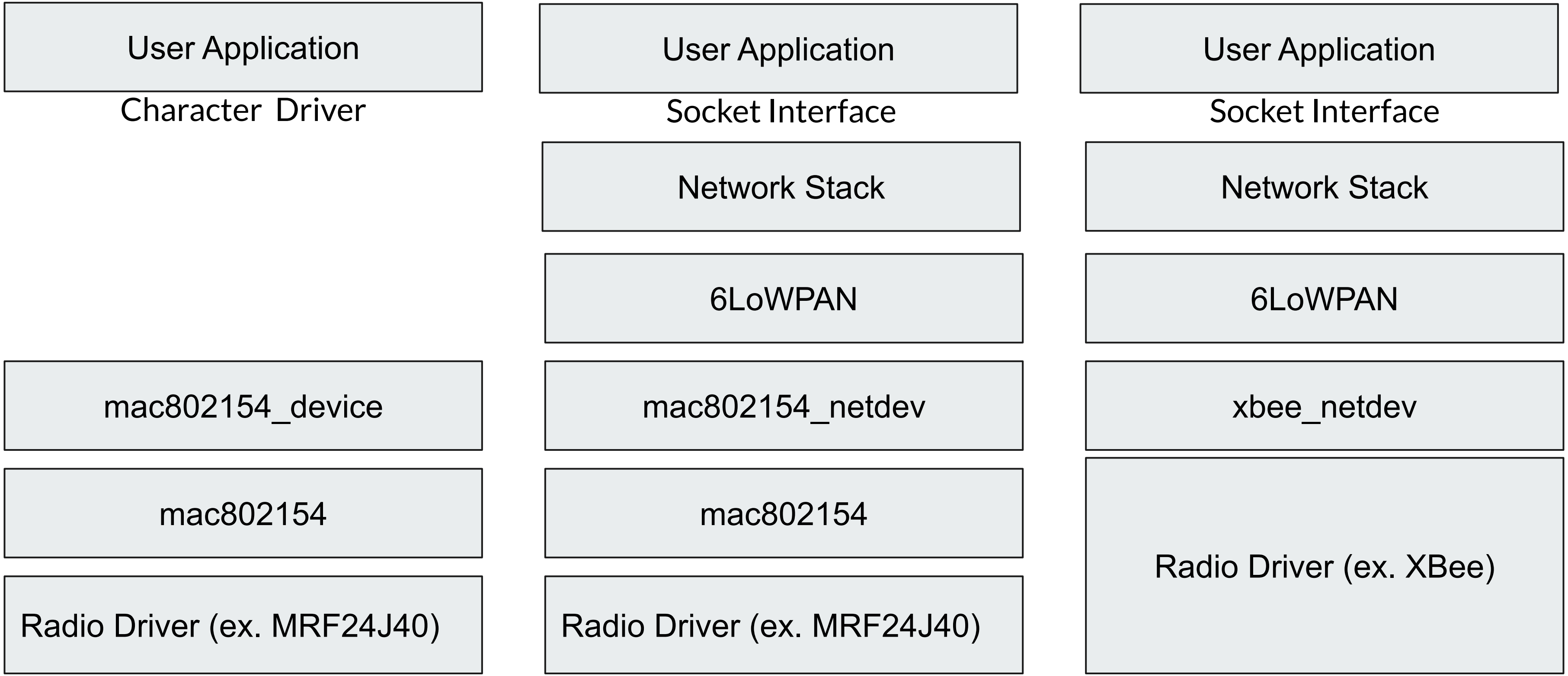
- Radio driver responsible for all PHY functionality and some MAC functionality; primarily anything related to timing
- MAC functionality required
  - Frame Check Sequence (FCS) injection/validation
  - Frame filtering - Incoming frame has a valid FCS, is not an ACK, and the frame is destined for either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

```
/* IEEE802.15.4 Radio Interface Operations **********************************/

struct ieee802154_radiocb_s
{
  CODE int (*poll) (FAR const struct ieee802154_radiocb_s *radiocb,
                    bool gts, FAR struct ieee802154_txdesc_s **tx_desc);
  CODE void (*txdone) (FAR const struct ieee802154_radiocb_s *radiocb,
                    FAR struct ieee802154_txdesc_s *tx_desc);
  CODE void (*rxframe) (FAR const struct ieee802154_radiocb_s *radiocb,
                    FAR struct ieee802154_data_ind_s *ind);
  CODE void (*sfevent) (FAR const struct ieee802154_radiocb_s *radiocb,
                    enum ieee802154_sfevent_e sfevent);
  CODE void (*edresult) (FAR const struct ieee802154_radiocb_s *radiocb,
                    uint8_t edval);
};

struct ieee802154_radio_s
{
  CODE int (*bind) (FAR struct ieee802154_radio_s *radio,
                    FAR struct ieee802154_radiocb_s *radiocb);
  CODE int (*reset) (FAR struct ieee802154_radio_s *radio);
  CODE int (*getattr) (FAR struct ieee802154_radio_s *radio,
                    enum ieee802154_attr_e ,
                    FAR union ieee802154_attr_u *attrval);
  CODE int (*setattr) (FAR struct ieee802154_radio_s *radio,
                    enum ieee802154_attr_e ,
                    FAR const union ieee802154_attr_u *attrval);
  CODE int (*txnotify)(FAR struct ieee802154_radio_s *radio, bool gts);
  CODE int (*txdelayed)(FAR struct ieee802154_radio_s *radio,
                    FAR struct ieee802154_txdesc_s *txdesc,
                    uint32_t symboldelay);
  CODE int (*rxenable) (FAR struct ieee802154_radio_s *radio, bool enable);
  CODE int (*energydetect) (FAR struct ieee802154_radio_s *radio,
                              uint32_t symboldelay);
  CODE int (*beaconstart)(FAR struct ieee802154_radio_s *radio,
                    FAR const struct ieee802154_superframespec_s *sfspec,
                    FAR struct ieee802154_beaconframe_s *beacon);
  CODE int (*beaconupdate)(FAR struct ieee802154_radio_s *radio,
                    FAR struct ieee802154_beaconframe_s *beacon);
  CODE int (*beaconstop)(FAR struct ieee802154_radio_s *radio);
  CODE int (*sfupdate)(FAR struct ieee802154_radio_s *radio,
                    FAR const struct ieee802154_superframespec_s *sfspec);
};
```

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



Board bring-up example

```c
/* Initialize and register the SPI MRF24J40 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;
  }
```

```c
#ifdef CONFIG_IEEE802154_NETDEV
  /* Use the IEEE802.15.4 MAC interface instance to create a 6LoWPAN
   * network interface by wrapping the MAC intrface 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 wpan%d: %d\n",
            0, ret);
      return ret;
    }
#endif

#ifdef CONFIG_IEEE802154_MACDEV
  /* If want to call these APIs from userspace, you have to wrap the MAC
   * interface in a character device viamac802154dev_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

Code	Name	Reference
0.01	GET	[RFC7252]
0.02	POST	[RFC7252]
0.03	PUT	[RFC7252]
0.04	DELETE	[RFC7252]

Table 5: CoAP Method Codes

- MQTT - Message Queue Telemetry Transfer - TCP/IP
- Constrained Application Protocl (CoAP) - RFC 7252
  - UDP
  - Designed from HTTP as starting point
  - Everything exposed via a URI

```
Client              Server       Client              Server
   |                  |             |                  |
   |   CON [0xbc90]   |             |   CON [0xbc91]   |
   | GET /temperature |             | GET /temperature |
   |   (Token 0x71)   |             |   (Token 0x72)   |
   +----------------->|             +----------------->|
   |                  |             |                  |
   |   ACK [0xbc90]   |             |   ACK [0xbc91]   |
   |   2.05 Content   |             |   4.04 Not Found |
   |   (Token 0x71)   |             |   (Token 0x72)   |
   |     "22.5 C"     |             |   "Not found"    |
   |<-----------------+             |<-----------------+
   |                  |             |                  |
```

Figure 4: Two GET Requests with Piggybacked Responses

https://tools.ietf.org/html/rfc7252

“Smart” Light Demo