Lower Network Stack redesign

José I. Álamos (@jia200x)

HAW Hamburg

September 15, 2020
Scope

Upper layers

Sock API  Netif API

Network Stack

South Bound API  South Bound API

IEEE 802.15.4 Lower Layer  Ethernet Lower Layer
South-bound API requirements (OSI, communication)

- OpenThread: PHY (full L2 frame, PSDU)
  - IEEE 802.15.4 SubMAC (a.k.a "PHY with steroids")
- OpenWSN: RADIO (direct access to transceiver)
  - IEEE 802.15.4 Transceiver
- GNRC: LINK LAYER (full L3 frame, MSDU)
  - IEEE 802.15.4 MAC, Ethernet MAC, LoRaMAC
- LWIP: PHY (full L2 frame, PSDU)
  - Ethernet PHY, IEEE 802.15.4 SubMAC
State of IEEE 802.15.4 Link Layer

- Missing IEEE 802.15.4 MAC (only framing in `gnrc_netif_ieee802154`).
  - No Security (L2 encryption, authentication, etc)
  - Not standard with standard IEEE 802.15.4 devices
  - Missing Low Power friendly features (e.g Indirect Transmission, Slotted Mode)
State of IEEE 802.15.4 PHY

- `netdev` with one of `netdev_xxx` variants.
  - Not well defined for each network device type (who cares about retransmissions? CSMA-CA?) and PHY configurations (modulations, bands, etc).
  - It adds a generic layer where the lower layer is already known (`gnrc_netif_ieee802154`)
    - More boilerplate code, ROM usage (e.g `gnrc_netif`)
    - It rules out certain optimizations.
State of IEEE 802.15.4 RADIO

- netdev with one of netdev Xxx variants.
  - Too generic
    - Semantics of standarized IEEE 802.15.4 radios are well defined
    - NETOPTs... NETOPTs everywhere.
    - Some transceiver operations require heavy semantic overload (e.g. NETOPT_PRELOAD)
State of IEEE 802.15.4 RADIO

- IRQ handling (Bottom Half Processing)
  - Call tree is not defined for the event callback. (See LoRaWan node ISR stack overflowed (#14962))
  - Makes strong assumptions on how the IRQ events should handled (at86rf215, nrf52840)
  - Code duplication between several network stacks (Most network stacks don’t expect to do Bottom Half Processing)
Pulls the whole PHY when only sub components are needed (e.g. OpenWSN)

- E.g MAC Information Base (MIB) and Physical Information Base (PIB) are usually handled by upper layers

Not well defined for the same kind of devices (e.g. NETDEV_EVENT_TX_MEDIUM_BUSY, NETDEV_EVENT_CRC_ERROR)

- It makes network stack integration hard.
“Hardware dependent” HAL: All radios are equal, but some radios are more equal than others.

- TX with CCA, CSMA-CA or direct transmission depends on the radio
  - Poor QoS in cc2538, nrf52840, etc
- Some radios block on send.

Hard to test
Proposals (so far!)

- **RADIO: IEEE 802.15.4 Radio HAL**
  - Community driven RDM: #13943
    - At least 7 members of the RIOT community contributed with the requirements, design and review.
  - Implementation: #14371

- **PHY: IEEE 802.15.4 SubMAC**
  - RFC: #13376
  - Implementation: #14950
IEEE 802.15.4 Radio HAL

- Lightweight API to provide uniform access to IEEE 802.15.4 compatible radios.

- Addresses community requirements:
  - Fully asynchronous (low power friendly, low memory footprint)
  - Well defined access to optional hardware acceleration (CSMA-CA with frame retransmissions, Auto CCA, etc).
  - Compatible with the requirements of current network stacks (OpenWSN, GNRC, etc)
  - Leaves PIB and MIB to upper layers.
Radio HAL Architecture (Old vs New)

- GNRC Netif
- GNRC Netif 802.15.4
- netdev

- GNRC Netif
- GNRC Netif 802.15.4
- IEEE 802.15.4 MAC
- IEEE 802.15.4 Radio HAL

- OpenWSN
Radio HAL components
Radio Ops

- Exposes common operations to control IEEE 802.15.4 devices. Some functions are similar to those available in netdev (e.g. recv mechanism).
  - Set the transceiver state
  - Set the PHY configuration (channel, tx power, etc)
  - Load and transmit a frame
  - Check if a hardware capability is supported (frame retransmissions, CSMA-CA, etc)
Request/Confirm pattern

- Blocking operations are defined by request and confirm functions.
  1. `request_xxx` requests an operation.
  2. Returns 0 if the operation was requested successfully. Otherwise, negative errno.
  3. If request was OK, `confirm_xxx` indicates the upper layer that the request finished (-EAGAIN or 0)

- The device MAY generate an event to indicate when to call the respective `confirm_xxx`
Request/Confirm pattern

Upper Layer:

Radio HAL:

request_cca()

confirm_cca()

-EAGAIN

CCA_DONE

confirm_cca()

OK
Request/Confirm pattern

This allows to use the API with polling mode (e.g polling confirm_xxx until it returns 0) or interrupts (e.g waiting for the specific event before calling confirm_xxx)
Event Notification

- Informs the upper layer about a specific event (TX done, CCA done, RX done, ACK timeout, etc).
- For certain radios this can occur in ISR context.
- IRQ processing is out of the scope of the Radio HAL and it’s up to the implementor of the bootstrap code (e.g. auto_init).
Implementation status

- API was merged (#14371)
- cc2538 and nrf802154 in master.
- Tracker in #14792.
  - Help wanted!!
## Memory footprint

<table>
<thead>
<tr>
<th>Radio</th>
<th>ROM netdev</th>
<th>ROM radio_hal</th>
<th>ROM Diff</th>
<th>RAM netdev</th>
<th>RAM radio_hal</th>
<th>RAM Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrf802154*</td>
<td>1585</td>
<td>1819</td>
<td>14%</td>
<td>305</td>
<td>318</td>
<td>4%</td>
</tr>
<tr>
<td>at86rf2xx**</td>
<td>3800</td>
<td>2158</td>
<td>-43%</td>
<td>72</td>
<td>40</td>
<td>-44%</td>
</tr>
<tr>
<td>cc2538</td>
<td>2622</td>
<td>2447</td>
<td>-8%</td>
<td>48</td>
<td>17</td>
<td>-64%</td>
</tr>
</tbody>
</table>

- *: the netdev version doesn’t implement all CCA variants and power functions.
- **: Doesn’t include SubGHz variants
Some comments: Function pointers vs Switch-Case

- It was decided to use Function Pointers instead of switch cases
  - In best case switch-cases might be implemented as a jump table (if indexes are ordered and there are only function calls)
  - For most cases, resulting code size and execution time might increase
- Easier to maintain.

See https://embeddedgurus.com/stack-overflow/2010/04/efficient-c-tip-12-be-wary-of-switch-statements/
IEEE 892.15.4 SubMAC

- Common layer that unifies common IEEE 802.15.4 lower MAC operations.

- In a nutshell, it provides “hardware independent” IEEE 802.15.4 compliant data transmission.
  - CSMA-CA Algorithm
  - Frame retransmissions

- It also stores the PIB (and some MIB attributes like CSMA-CA parameters)

- It uses the IEEE 802.15.4 Radio HAL and fill the gaps if the radio doesn’t support a specific hardware acceleration (e.g frame retransmissions).
IEEE 802.15.4 SubMAC API

- Send PSDU (Full L2 frame) with CSMA-CA (and possibly retransmissions)
- Set PIB parameters (channel, page, TX power)
  - All hardware independent validations are done here, not in the radios
- Set "MAC" states
- Allocation of frames is up to the network stack (same as netdev).
Implementation status

- PR in #14950
- Includes a netdev_ieee802154_submac transition layer to ease migration.
  - It implements a “generic IEEE 802.15.4 netdev device with CSMA-CA and frame retransmissions”
  - Fully compatible with GNRC, OpenThread, LWIP, etc.
Results

- Ping between cc2538_rf and at86rf2xx
- 1024 bytes payload
- 170 ms interval
- 1000 packets
Direct transmission

— fe80::2068:3123:fe42:2e25 PING statistics —
1000 packets transmitted, 626 packets received, 37% packet loss
round-trip min/avg/max = 129.693/140.842/155.274 ms
CSMA-CA and retransmissions

— fe80::2068:3123:fe42:2e25 PING statistics —
1000 packets transmitted, 1000 packets received, 0% packet loss
round-trip min/avg/max = 133.318/145.909/161.423 ms
OpenThread works out of the box on CC2538! (the same is expected for the others)

LWIP and the other stacks should work too.
Comments?
Next steps?

1. What about the other low layers? (e.g. BLE)
2. IEEE 802.15.4 MAC
   - IEEE Indirect Transmission
   - IEEE 802.15.4 security (L2 encryption, Section 7.5.8 from IEEE 802.15.4 2006 standard)
   - TSCH (OpenWSN? "GNRC TSCH"?)
   - Reuse Link Layer from OpenWSN and/or OpenThread
3. Descriptors allocation + bootstrap
   - Unify allocation?
   - IRQ handling? Event handler?
   - Common auto_init for all stacks?
4. How to send L2 data?
   - Do we need an L2 interface?
5. GNRC: One stack per netif?
6. Frame buffers vs zero copy.