ieee802154 and rpld

LPC 2020
IoT Microconference
2020-08-27, Virtual

Stefan Schmidt
stefan@datenfreihafen.org
Who am I

- FOSS developer since 2006
- EFL developer and release manager
- Linux kernel ieee802154 subsystem maintainer
Agenda

- IPv6 over LoWPAN
- Linux-wpan
- rpld
IPv6 over LoWPAN (6LoWPAN)
Products

- Products with IEEE 802.15.4 transceivers (using 6LoWPAN, newer ZigBee profiles or OpenThread):
  - Nest devices (e.g. thermostat, protect, etc)
  - Google WiFi / OnHub router
  - IKEA Tradfri system
  - Philips Hue light system
  - And many, many more
Motivation 6LoWPAN

- “Things” might have restricted wireless connectivity
- Using IPv6 instead of something proprietary allows the usage of existing and proven protocols driving the Internet
- **But** unmodified TCP/IP protocol headers can clash with MTU limitations
- Things often only need to transfer small amounts of data
IEEE 802.15.4

- IEEE specifications for Low-Rate Wireless Personal Area Networks (LoWPAN)
- Not only low-rate, but also low-power
- PHY and MAC layer with star and peer-to-peer topologies
- Addressing but no routing defined
- Mesh routing possible with layers on top
- Designed for small sensors to run months/years on battery with the right duty cycle
- **127 bytes MTU** and 250 kbit/s
- Often mixed-up with ZigBee as it is used as PHY and MAC layer
- Compared to Bluetooth it is older than BTLE and less complex
6LoWPAN

- Physical and MAC layer defined by IEEE 802.15.4
- Series of IETF specifications from 2007 onwards (RFCs 4944, 6282, etc)
- Goal was to use IPv6 in sensor networks based on IEEE 802.15.4
- Direct IP addressing of nodes

- Adaptation layer between data-link and network layer
- Address auto-configuration
- Frame encapsulation and fragmentation
- Header compression

L5 Application Layer
L4 Transport Layer
L3 Network Layer
L2 Data Link Layer
L1 Physical Layer

Application
TCP | UDP | ICMP
IP
Ethernet MAC
Ethernet PHY

Application
UDP | ICMPv6
IPv6
6LoWPAN
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
The Header Size Problem

- Worst-case scenario calculations
- Maximum frame size in IEEE 802.15.4: 127 bytes
- Reduced by the max. frame header (25 bytes): 102 bytes
- Reduced by highest link-layer security (21 bytes): 81 bytes
- Reduced by standard IPv6 header (40 bytes): 41 bytes
- Reduced by standard UDP header (8 bytes): 33 bytes
- This leaves only **33 bytes** for actual payload
- The rest of the space is used by headers (~ 3:1 ratio)
The Header Size Solution

- IPv6 with link-local and UDP on top
- IPHC with NHC for UDP
- The 48 bytes IPv6 + UDP header could in the best cases be reduced to 6 bytes
- That allows for a payload of 75 bytes (~ 2:3 ratio)
Linux-wpan
Why linux-wpan?

- Goal: IEEE 802.15.4 and 6LoWPAN support in mainline
- Platforms already running Linux would benefit from native IEEE 802.15.4 and 6LoWPAN subsystems
- IEEE 802.15.4 transceivers can easily be added to existing hardware designs (SPI + few GPIOs)
- Battery powered sensors are more likely to run a RTOS like RIOT, Contiki or Zephyr, but they need a border router
- Started in 2008 as linux-zigbee project, from 2012 mainline (renamed to linux-wpan)
Development Boards

- Raspberry Pi’s with Openlabs shield (AT86RF233)
- Ci40 Creator (CA-8210)
  - Transceivers can be hooked up via SPI
    (drivers have devicetree bindings)
- ATUSB USB dongle
Network Interfaces

- The wpan0 interface shows up automatically
- IEEE80215.4 specific configuration over netlink, e.g. with wpan-tools
- Setting up the basic parameters:
  - $ ip link set lowpan0 down
  - $ ip link set wpan0 down
  - $ iwpan dev wpan0 set pan_id 0xabcd
  - $ iwpan phy phy0 set channel 0 26
  - $ ip link add link wpan0 name lowpan0 type lowpan
  - $ ip link set wpan0 up
  - $ ip link set lowpan0 up

Setting up the interface in promiscuous mode:
  - $ iwpan dev wpan0 del
  - $ iwpan phy phy0 interface add monitor%d type monitor
  - $ iwpan phy phy0 set channel 0 26
  - $ ip link set monitor0 up
  - $ wireshark -i monitor0

- No automatic channel hopping (changing the channel manually in the background is possible)
Socket Interfaces

- **AF_INET6 Socket**
  - Can be used like a normal IPv6 socket
  - Transparently handled

  ```
  sd = socket(PF_INET6, SOCK_DGRAM, 0);
  dst.sin6_family = AF_INET6;
  sendto(sd, ...);
  ```

- **AF_IEEE802154 Socket**
  - Direct IEEE 802.15.4 communication
  - Short and extended addressing schemes as well as network PAN ID handling

  ```
  sd = socket(PF_IEEE802154, SOCK_DGRAM, 0);
  dst.family = AF_IEEE802154;
  dst.addr.pan_id = 0x0023;
  dst.addr.addr_type = IEEE802154_ADDR_LONG;
  memcpy(&dst.addr.hwaddr, long_addr, IEEE802154_ADDR_LEN);
  or
  dst.addr.addr_type = IEEE802154_ADDR_SHORT;
  dst.addr.short_addr = 0x0002;
  sendto(sd, ...);
  ```
Current Status

- 6LoWPAN with fragmentation and reassembly (RFC 4944)
- Header compression with IP header compression (IPHC) and next header compression (NHC) for UDP (RFC 6282), shared with Bluetooth subsystem
- ieee802154 layer with softMAC drivers for at86rf2xx, mrf24j40, cc2520, atusb, adf7242, ca8210 and mcr20a
- Hwsim virtual driver module for testing
- USB dongle to be used on your workstation
- Link Layer Security
New Webpage launched

- Recent effort to move project page
  wpan.cakelab.org → https://linux-wpan.org

- Old one was too difficult to maintain and access

- New one is with GitHub pages, well known, easy edits

- Existing content has been moved and a redirect is in place

- More content (e.g. rpld) is needed and better integration with the kernel docs
Wpanusb - past

- Generic USB device driver (like btusb)
- Originally developed by Andrei Emeltchenko @intel
- ieee802154 does not specify an HCI
- Own USB interface spec based on atusb
- Developed together with Zephyr firmware
- Project stalled in 2018 as Andrei has no time
Wpanusb – future

- Taken over by Koen, Eric and Stefan now
- Extend USB interface spec to be more flexible
e.g. capabilities provided by firmware, different
  frequency bands, power levels, permanent
  address, etc
- CAN subsystem maintainers expressed interest for a
  similar generic USB driver
- RIOT OS firmware to be developed in parallel
- Zephyr firmware needs to be updated to new spec
- Bare metal or Contiki implementations?
Kselftest support

- Hwsim will be hooked in kselftest to give an easy way of regression testing
- Will be used in review process
- Useful during network stack re-work’s
- Basic suite of tests to start with (ieee802154 frames in different sizes, 6lowpan packets in different sizes, header compression on and off)
Linux-wpan Future

- Implement missing parts of the IEEE 802.15.4 specification
- Beacon and MAC command frame support
- Coordinator support in MAC layer and wpan-tools
  - Scanning
- Add better support for HardMAC transceivers
- Configuration interface for various header compression modules
- Expose information for routing protocols (started with LQI already)
RTOS Systems
RTOS Systems

- Various real time operating systems support IEEE 802.15.4 and 6lowpan
  - RIOT
  - Contiki
  - Zephyr
  - OpenThread
  - MbedOS (nanostack finally open source from mbed-os-5.7 onwards)
## Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Linux</th>
<th>RIOT</th>
<th>Contiki</th>
<th>Zephyr</th>
<th>OpenThread</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 802.15.4: data and ACK frames</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>IEEE 802.15.4: beacon and MAC command frames</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>IEEE 802.15.4: scanning, joining, PAN coordinator</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>IEEE 802.15.4: link layer security</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: frame encapsulation, fragmentation, addressing</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: IP header compression (RFC 6282)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: next header compression, UDP only (RFC 6282)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: generic header compression (RFC 7400)</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>6LoWPAN: neighbor discovery optimizations (RFC 6775)</td>
<td>Partial</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>Mesh link establishment draft</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
RPL

- Routing protocol for low power and lossy networks
- IETF approach, route over protocol
- IPv6 Routing Protocol for Low-Power and Lossy Networks (RFC6550, RFC6553)
- Tree like topology in a mesh, one parent, n childs
- Constructs a directed acyclic graph in an attempt to minimize the routing costs
RPL- up and down routes
rpld

- Unstrung was the Linux user-space reference for rpld as new alternative
  https://github.com/linux-wpan/rpld
- Part of linux-wpan and only around for a year
- Developed by Alexander Aring
- Netdev Conf 0x14 talk RPL: IPv6 Routing Protocol for LLNs
rpld – non-storing mode

- In a nutshell RPL has two modes of operation:
  - Storing: routes are propagated via ICMPv6 messages into the routing table
  - Non-storing: source routing with routing header

- First implementation with non-storing mode on Linux by extending the existing Kernel segment routing

- https://netdevconf.info/0x14/session.html?talk-ex tend-segment-routing-for-RPL
rpld – non-storing mode

- Source routing extention header only inserted by root node
- IPv6 only
- Forwarding with address swapping and loop detection
- Compression of addresses in headers
- Details on kernel implementation in the mentioned talk
Thank you!
Bonus
Address Auto-configuration & Fragmentation

**Stateless address auto-configuration:**
- Used for IPv6 networks without DHCP
- Based on layer 2 address
  - Extended address uses EUI-64 as is
  - Short address uses EUI-48 to EUI-64 mapping
    (16 Bit PAN+16 Bit zero+16 Bit short address)

**Fragmentation:**
- IPv6 requires the link to allow for a MTU of at least 1280 bytes
- Impossible to handle in the 127 bytes MTU of IEEE 802.15.4
- 6LoWPAN adds a 11 bit fragmentation header allows for 2048 bytes
- Fragmentation should still be avoided for best performance
IPv6 Header Compression (IPHC)

**IPHC (RFC6282)**
- Deprecates HC1 & HC2 compressions from RFC4944
- Better compression for global and multicast address, not only link-local
  - Compress header fields with common values: version, traffic class, flow label, hop limit
- NHC IPv6 Extension Header compression (RFC6282)
  - Hop-by-Hop, Routing Header, Fragment Header, Destination Options Header, Mobility Header
- NHC UDP Header compression (RFC6282)
  - Compressing ports range to 4 bits
  - Allows to omit the UDP checksum for cases where upper layers handle message integrity checks
Generic Header Compression

- Generic approach instead of defining a scheme for each header
- Plugging into NHC
- Useful for header like payload e.g. DTLS or RPL (addresses elided from dictionary)
- 6CIO option in neighbour discovery messages to indicate support
- LZ-77 style compression with byte codes (RFC7400)
  - Appending zeroes, back referencing to a static dictionary and copy
Mesh-under

- Allow for fast packet forwarding without travelling the IP stack
- IEEE 802.15.4 does not include mesh routing in the MAC specification, mesh implementations is an extra layer above the MAC but below the network layer
- Various (proprietary) implementations (e.g. WirelessHART, ZigBee mesh, RF mesh, etc)
- IEEE 802.15.5 can also to be used for mesh on top of 15.4
- 6LoWPAN specification has a field for mesh headers
- Lost fragments of bigger packets will cause troubles
Thread

- Mesh network specification from Thread Group
- OpenThread implementation from NestLabs
- Routing Information Protocol (RIP) algorithms are used, but not RIP itself
- Distribution of route information is handled by mesh link establishment (MLE, IETF drafts dropped)
- MLE allows router to update the tables of routing costs periodically in a compressed form
- Due to MLE no on-demand route discovery is needed
References

- IEEE 802.15.4 specification (PHY and MAC layer)
  
  http://standards.ieee.org/about/get/802/802.15.html

- RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks
  

- RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks
  

- RFC 7400: 6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)
  

- Linux-wpan source (wpan-tools & rpld) and project pages
  
  https://github.com/linux-wpan