A Sockets API For LoRa

Andreas Färber,
SUSE Labs
afaerber@suse.com
About The Presenter

• Project Manager for arm64 architecture at SUSE Labs
• Involved in arm port of openSUSE Linux distribution
• Kernel maintainer for Realtek and Actions Semi arm SoCs
• Other kernel projects you might know:
  – Odroid-XU, Parallella, Spring Chromebook, GeekBox, …
  – STM32F4, FM4, XMC4500; S905, IAP140, MB86S71, RDA8810PL
• Background in virtualization technologies – QEMU
Why LoRa Technology?

- LoRa = **Long Range** – radio modulation by Semtech
- Low-Power Wide Area Network (LPWAN) with low throughput
- Unlicensed sub-GHz and 2.4 GHz ISM/SRD bands (U-LPWA)
- No dependency on network infrastructure providers
- Wide availability of HW – https://en.opensuse.org/HCL:LoRa
- … and just because it’s possible!
Getting Started With LoRa Chipsets

... and down the rabbithole it goes!
## Types Of LoRa Radio Modules

<table>
<thead>
<tr>
<th>Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain transceiver</td>
<td>- SPI / UART / USB</td>
</tr>
<tr>
<td></td>
<td>- Volatile register settings</td>
</tr>
<tr>
<td></td>
<td>- Software MAC needed</td>
</tr>
<tr>
<td>MCU w/firmware + transceiver</td>
<td>- UART / USB Serial</td>
</tr>
<tr>
<td></td>
<td>- Firmware determines chip features exposed</td>
</tr>
<tr>
<td></td>
<td>- Optional certified MAC</td>
</tr>
<tr>
<td>Plain MCU + transceiver</td>
<td>- n/a – no fixed API</td>
</tr>
<tr>
<td></td>
<td>- Custom MCU code for sending / receiving</td>
</tr>
<tr>
<td></td>
<td>- Optional MAC</td>
</tr>
</tbody>
</table>
Accessing LoRa Hardware Today

<table>
<thead>
<tr>
<th>mm</th>
<th>sched</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi</td>
<td>...</td>
<td>spidev</td>
</tr>
<tr>
<td>ttyS0</td>
<td>8250</td>
<td>pl011</td>
</tr>
<tr>
<td>usb</td>
<td>ftdi_sio</td>
<td>...</td>
</tr>
<tr>
<td>cdc-acm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/dev</td>
<td>spi0.0</td>
<td>ttyS0</td>
</tr>
<tr>
<td>ttyUSB0</td>
<td>ttyACM0</td>
<td></td>
</tr>
</tbody>
</table>

read/write ioctl
Issues With LoRa Open Source Software Today

- No upstream community – per-vendor application forks
- Software license incompatibilities
- Use of spidev kernel module gets ugly in distros
- Hardware detection duplicated into applications

**Idea:** Move chipset drivers into mainline Linux kernel. Encourage generic, community-maintained packet forwarders.
Collecting Requirements

- Shall expose equivalent chipset features as before
- Shall allow implementation of proprietary protocols
- Shall allow reuse of protocols layered on top
- Shall fit all Semtech chipsets and many third-party modules

_Idea:_ Sockets seem an intriguing approach for LoRaWAN. Similarities to Wifi and IEEE 802.15.4 may help users.
Andreas In Wonderland – Sockets (Proposed)

```
mm sched ...
spi ... sx125x sx1276 sx1301 serdev mm002 ...
wimod
```

```
/sys/class/net lora0 lora1 lora2 lora3
```

Sockets

Buffers

bind
read/write
Semtech SX1272 f. / SX1276 ff. Transceivers

- Single channel
- Two modes: FSK/OOK and LoRa (switchable via Sleep mode)
- State machine for RX vs. TX (switchable via Standby)
- SPI register interface
- 256 byte RAM data buffer (LoRa) / 64 byte FIFO (FSK)
Semtech SX1261 f. / SX1268 Transceivers

- Single channel
- Two modes: LoRa and FSK
- State machine for RX vs. TX (switchable via Standby RC/XOSC)
- SPI command interface, indirect register interface
- 256 byte RAM data buffer
Semtech SX1280 f. Transceivers

- Single channel
- Multiple modes: LoRa, FLRC, FSK, BLE and Ranging
- State machine for RX vs. TX (switchable via Standby RC/XOSC)
- UART and SPI command interface, indirect register interface
- 256 byte RAM data buffer
Semtech SX1301 / SX1308 Concentrators

- Multi-channel
- IF0-7 LoRa channels, IF8 LoRa uplink channel, IF9 FSK channel
- Two radio transceivers (SPI/ADC) – SX1255 / SX1257 f.
- SPI register interface – no documentation, only reference code
- 1024 byte data buffer
- Firmware blobs for calibration and operation
LoRa Modules With UART Interface

- The serial device bus allows to attach drivers to tty device
  - Child node of UART in Device Tree
- Callback for reception – might be individual bytes or chunks
- API for sending available
- “AT command” interfaces are not standardized
- Binary interfaces encountered, too
- Interrupts plus active reception, or asynchronous notifications
Unsolved: USB Based Serial Protocols

- Problem: usb-serial devices don’t have an of_node associated
  - Proposal by Johan Hovold disliked by Rob Herring
- Problem: How to tell a USB device which serdev driver to use?
  - DT: via usb<vid>,<pid>?
  - ACPI: overload tables via command line?
- Problem: How to deal with hot-plug and changing ports?
  - Derive USB drivers? Use line discipline?
Socket Addressing For Radios

- Transmission is broadcast
  - Addressing only at MAC layer
- Preamble may serve to recognize frame start, not “metadata”
- Optional filtering by Sync Word

**Idea:** Define address as radio properties that allow reception. (An alternative following later.)
LoRa Socket Address (Proposed)

- Network interface index
- Radio frequency
- Spreading Factor
- Bandwidth
- Sync Word (1 Byte)
LoRa Socket Layers (Proposed)

User

PF_LORA
SOCK_DGRAM

maclorawan

LoRaWAN Hard MAC

LoRa PHY driver

PF_LORAWAN
SOCK_DGRAM + SOCK_SEQPACKET

nllorawan

nllora

genl
LoRaWAN Socket Address (Proposed)

- Network interface index
- Data Rate
  - LoRa: channel frequency, SF, bandwidth
  - FSK: channel frequency, bandwidth
- Port

Data Rate implies a fixed LoRa / FSK Sync Word respectively.
PHY Management Via Generic Netlink (Proposed)

- Socket based command protocol (genl)
- Example: querying frequency of (channel on) device
  - Needs to work for all chipsets and modules
  - Attributes can be added to refine, e.g. channel for SX130x
- TBD: Don’t rely on loraX interface, think of SDR
- Distinction between Device Tree (physical) and NL (config)
LoRaWAN Management Via Netlink (Proposed)

• Similar, but one level higher
  – Implementation might delegate to PHY netlink interface or translate to AT commands directly, depending on device
• Examples: Data Rate, Join
Regulatory Compliance

- wireless-regdb does not cover sub-GHz frequency bands yet
- With SX128x entering 2.4 GHz realm, reuse seems sensible
- Examples: Transmit power limitation in EU, duty-cycle limit
- Plan: Provide configuration commands in nllora that userspace tools could use to change individual settings
Listening Can Be Hard

- Packets can be transmitted with different modes and settings
- Sockets require to receive whenever we’re not transmitting
  - How to detect and handle conflicting settings for reception?
  - When socket is opened, all settings need to have been initialized
- There’s no unified frame format field to detect MAC protocols
  - Need to try to parse incoming frames for each protocol
Protocol Layers Around LoRa

- LoRaWAN
- 6LoWPAN
- BLE
- Weightless, EnOcean, Z-Wave, ...
- IEEE 802.15.4
- Sigfox
- Wireless M-Bus
- FSK modulation
- ASK (OOK)
- FLRC
- LoRa modulation
- User
- Symphony Link, MOST, RadioShuttle, WISE-Link, ...
- User
Frequency-Shift Keying (FSK)

- Address: frequency, sync word (multi-byte), Gauss ...

- Also found in: nRF24L01+, CC1120, MRF89XAM8A, SP1ML
On/Off-Keying (Amplitude-Shift Keying)

- Address: frequency, …

- Also found in: CC1120, MRF89XAM8A
Fighting Pollution: Unified Radio Sockets?

- Can we avoid a socket address for each modulation?
- Use generic **PF_PACKET** + **SOCK_DGRAM** + htons(ETH_P_…)?
  - Would not allow radio configuration via socket address
  - Would still allow **SOCK_RAW** for Software Defined Radio
  - How could we switch modes or detect conflicts? Socket options?
Related: Bluetooth LE Support

- Semtech SX128x: alternative mode
- AppconWireless RF1276TS, Laird RM1xx: separate antenna
- Kernel appears to rely on **HCI** – what to do about raw PDUs?
Test Setup For LoRa Kernel Drivers (1/2)

- arm, arm64 and mips Single Board Computers
- Shield / HAT / Click / XBee expansion boards or flying wires
- Relevant chipsets being tested before pushing to linux-lora.git
  - Limitations: 868 MHz and 433 MHz (EU), donated hardware
- Idea: interoperability and co-existence testing
  - Not fully automated Continuous Integration (yet)
Test Setup For LoRa Kernel Drivers (2/2)

• mips: lora-next branch (based on linux-next)
  – .dts modified
• arm(64): openSUSE Tumbleweed + Kernel:HEAD repo (-rcX)
  – Build modules against host kernels, with tricks for new defines
  – DT Overlays via U-Boot where possible
• https://github.com/afaerber/lora-modules
Action Plan

- Working towards RFC v2 – need to complete regmap adoption
  - Staging branch to be archived and squashed into series
- On top: LoRaWAN soft MAC patch series by Jian-Hong Pan
  - Cf. https://www.slideshare.net/chienhungpan/lorawan-class-module-and-subsystem
- Validate / evolve **ABI** design – needs testing and feedback
- Merge into mainline kernel, enable in openSUSE Tumbleweed
Industry Contributors – Code

Laird™
Industry Supporters – Hardware

- AppconWireless
- Dragino
- Embedded Micro Technology
- Gimasi
- HELTEC Automation
- IMST
- Laird
- Mipot
- Nemeus
- nFuse
Competing LPWAN Technologies
Other U-LPWAN: Sigfox

- Frequency: Unlicensed sub-GHz SRD/ISM bands
- MTU: 12 bytes uplink, 8 bytes downlink

- Why care? Found in Nemeus MM002-LS modules
  - How to expose? Device? PF_SIGFOX? lora0 + sigfox0?
  - How to interact with LoRa sockets?
Other LPWAN: NB-IoT

- Frequency: Licensed 3GPP bands
- MTU: 1500 bytes
- Two modes: UDP and non-IP
- SIM card needed

How to handle in Linux?
Conclusions
Summary

- PoC for LoRa sockets & SX1276 Tx has been implemented
- No clear solution for USB adapters / mPCIe cards found yet
- Not a technology endorsement by openSUSE or SUSE
Resources

• RFC patch series: https://patchwork.ozlabs.org/cover/937545/
• Staging tree with lora-next branch: https://git.kernel.org/pub/scm/linux/kernel/git/afaerber/linux-lora.git/
• Testing hints: https://github.com/afaerber/lora-modules
• Chipset overview and links to SBC expansion boards: https://en.opensuse.org/HCL:LoRa
Questions? Feedback?
Backup
Radio Modulation Types Of Other Technologies

- MIOTY: Lfour: BPSK; TS-UNB: GMSK; DD-UNB: BFSK
- Sigfox: D-BPSK and GFSK
- Weightless-P: GMSK BT=0.3 or OQPSK
- Wireless M-Bus: 4GFSK

- Bluetooth LE: GFSK (2.4 GHz)
License
This slide deck is licensed under the Creative Commons Attribution-ShareAlike 4.0 International license. It can be shared and adapted for any purpose (even commercially) as long as Attribution is given and any derivative work is distributed under the same license.

Details can be found at https://creativecommons.org/licenses/by-sa/4.0/

General Disclaimer
This document is not to be construed as a promise by any participating organisation to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. openSUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for openSUSE products remains at the sole discretion of openSUSE. Further, openSUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All openSUSE marks referenced in this presentation are trademarks or registered trademarks of SUSE LLC, in the United States and other countries. All third-party trademarks are the property of their respective owners.

Credits
Template
Richard Brown
rbrown@opensuse.org

Design & Inspiration
openSUSE Design Team
http://opensuse.github.io/branding-guidelines/