Networking in Zephyr LTS and beyond

Paul Sokolovsky
Linaro LITE
Zephyr RTOS: Short intro/recap

- [https://www.zephyrproject.org/](https://www.zephyrproject.org/)
- A Small, Scalable Open Source RTOS for IoT Embedded Devices
- “The Zephyr™ Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.”
- 29K commits, 450 contributors, 83 releases (including RCs).
- 8 architectures, ~150 boards
- 1.14: Long-awaited LTS release!
Zephyr’s “OS personalities”

- Zephyr’s own native API
- CMSIS RTOS v1/v2
- Most importantly, POSIX
  - Still in early stages (comparing to what the POSIX standard provides).
  - E.g. there’re filesystem API, pthread API, ... well, mostly, that’s it.
  - POSIX also includes BSD Socket API
  - Zephyr’s BSD Socket subsystem started on its own, currently not well integrated with the POSIX subsystem (addressing that is an ongoing task).
net_app API removal, BSD Sockets as main API

Zephyr before 1.14:

- Application-level protocol libs
- BSD Socket API
- Net_app API
- Zephyr native networking API (low-level)
net_app API removal, BSD Sockets as main API

Zephyr 1.14:
net_app API removal, BSD Sockets as main API

What was wrong with net_app?

- While higher-level, convenience wrapper around native API, it’s still:
- An adhoc API just for Zephyr.
- Low-level, uses net_pkt structure directly.
- Due to this, error-prone to use for TCP protocols. There was a bunch of net_app-using code which assumed that one application-level protocol message equals one network packet (app proto message starts at the network packet boundary, crosses it only in “right” boundaries, there’s single app proto message per network packet, etc. - none of these are true in a general case).
- Uses callbacks (just like native API). This is immediate problem for kernel vs userspace split mode, which is security-driven operation mode for Zephyr.
net_app API removal, BSD Sockets as main API

Summary:
- Zephyr-adhoc API
- Low-level and error-prone
- Callback-based, not compatible with userspace mode

Verdict:
Switch to BSD Sockets API as the main user-facing API.

Breaking news:
As maintaining 2 APIs is cumbersome, remove net_app API before 1.14 LTS, as the LTS is expected to be supported for 2 years.
net_app API removal, BSD Sockets as main API

Benefits of BSD Sockets API:
- Standard API, not just Zephyr’s NIH
- Thus, many programmers know it, and can leverage their knowledge with Zephyr
- Can port existing apps, not just write apps specifically for Zephyr from scratch
- Tested and tried memory separation model (good for security)

(Current) drawbacks of BSD Sockets API:
- Less efficient - there’s some room for optimization.
- In particular no “zero copy” support. Addressing that would be hard (adding adhoc extensions to the API).
- Many features are still missing.
Sockets-based application protocol libs

- With net_app API gone, higher, application-level protocols implemented on top of it are affected too:
- Some are ported from net_app to sockets (e.g. LWM2M, SNTP)
- Some got a new, natively socket-based implementation (e.g. MQTT)
- Some are gone - for now (e.g. HTTP, WebSocket)

Losing protocols like HTTP is definitely sad. But remember what was said about net_app based protocol implementations: many of them had issues. HTTP lib also implemented only HTTP/1.1 and only chunked transfer encoding (i.e. implemented quite an adhoc subset of HTTP functionality).

A call for stakeholders of HTTP, etc. was made. And there’re now 2 choices: port an existing lib (or libs), or write one from scratch for Zephyr.
TLS on socket level

- Adhoc API specific to Zephyr
- Thus, not portable to other OSes (e.g. no easy debugging on Linux)
- Arguably, layering violation in the stack
- BUT, an easy to use API, and a way it recently was done in a few other systems too. Also, a popular way to implement it in product-specific vendor SDKs - thus, offloading possibility.
Packet and CAN sockets

- Based on BSD, Linux API (extension to POSIX)
- Packet sockets allow to perform L2 communication using Sockets API. Currently, only Ethernet is supported.
- CAN sockets are implemented, roughly compatible with Linux (work undergoing to polish the rough edges).
Elaborating Sockets API

- Mostly based on case studies with porting POSIX software
- Example: OPC-UA protocol implementation - open62541 lib ([https://github.com/open62541/open62541](https://github.com/open62541/open62541))
- Working on Zephyr port:
  - Exposed a few bugs which were hard to reproduce previously
  - Led to work on adding a number of missing features in the API
  - Exposed the problematic situation with BSD Sockets vs POSIX subsystems integration
  - Some changes based on this work are already in 1.14!
net_buf refactor

Zephyr internal representation of a network packet:

Zephyr 1.13 and below:

- net_pkt contains metadata about the networking packet, and pointer to “fragment chain”
- net_buf contains alloc-related metadata, “generic” metadata, and pointer to data
- net_buf’s are allocated from pool of fixed-size blocks.
- net_buf is used both by IP stack and some other subsystems, e.g. Bluetooth.
net_buf refactor

Zephyr 1.13 and below packet structure - issues:

- Generic metadata size is compile-time constant. *Every* net_buf contains it, regardless whether used or not. (That metadata is good when 1 net frame == 1 net_buf, e.g. BT, 15.4)

- Each net_buf has a fixed size, and that size is also a compile-time constant. (As it’s reused by different subsystems, that poses an issue).

net_buf refactor idea:

- Get rid of that “generic metadata” storage, subsystem which need it can deal with it themselves, without incurring overhead on entire system.

- Separate “metadata” and “data” portions of net_buf, explicitly allowing data to be elsewhere.

- Explicitly allow variable-size data part of net_buf.
net_buf refactor

Zephyr 1.14 net_buf refactor - what actually came out of it:

- Massive refactor throughout the IP stack (100+ commits), work on which went well into the RC phase.
- Pervasively affected the native internal network API (that kinda was an idea - to optimize the stack, which requires new API, the point that the old API is gone, and new API uses names from old API with different signatures).
- There may be some regressions.
- There may be more changes to do (e.g., update documentation).
Future work

- Better integration between POSIX and Socket subsystems
- Socket-based protocol libs for HTTP, WebSocket
- More socket features/options
- New TCP implementation
  - TCP implementation in Zephyr has issues, even though a number of fixes were applied for 1.14. Performance is also not great. There’s an experimental project to write new TCP implementation from scratch.
- Zero-copy socket options/extensions
- epoll
  - Zephyr implements (subset of) POSIX poll(), and 1.14 adds select() on top of it. But poll() is known to be not scalable, so we may want to implement Linux epoll().
- Better DMA support
- Testing, testing, testing
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org