Enabling TCP/IP Offload in Zephyr with TI SimpleLink

Gil Pitney
Introduction

- The TI SimpleLink CC32xx family of MCUs provides an SoC and supporting SDK which completely offloads the WiFi stack (OSI Layers L4+) onto an integrated network coprocessor (NWP).
  - This provides significant memory, CPU, and energy savings.
  - Eg: The app MCU can deep sleep while TCP sessions are managed efficiently by the NWP.
- The SimpleLink SDK currently has no explicit support for the Zephyr IoT OS, but is designed to be portable to a new OS.
- A native IP stack for the Zephyr IoT OS is under development, which includes an experimental “IP offload” option.
- Looking at past attempts to integrate TCP/IP offload engines into Linux, we can anticipate various challenges.
- We briefly review those challenges, and look at options for offloading WiFi from Zephyr to the TI SimpleLink network coprocessor.
Challenges integrating TCP/IP Offload into Linux

- **Security**: TOE (TCP/IP Offload Engine) is implemented in hardware, so patches cannot be made to quickly address security issues.
- **Complexity**: TOE requires very large changes to a networking stack in order to be supported properly; even when that is done, features like QoS, packet filtering, and multiple bearer support may not work.
- **Proprietary**: TOE is implemented differently by each hardware vendor. So more code must be rewritten to deal with the various implementations. TOE firmware cannot be easily modified since it is closed-source.
- **Vendor-specific tools**: In order to configure a TOE NIC, hardware-specific tools are usually required.
- **Sources**:
  - [https://wiki.linuxfoundation.org/networking/toe](https://wiki.linuxfoundation.org/networking/toe)
TI SimpleLink (CC32xx) Architecture & APIs

- **Device API**: Manages hardware-related functionality such as start, stop, set, and get device configurations.
- **WLAN API**: Manages WLAN, 802.11 protocol-related functionality such as device mode (station, AP, or P2P), provisioning method, connection profiles, and connection policy.
- **BSD Socket API**: All higher level protocols (MQTT, SMTP, TFTP, HTTP) and networking apps built on standard API.
- **NetApp API**: Enables offload networking servers (HTTP, DHCP, mDNS).
- **NetCfg API**: Configures network parameters (MAC address, acquiring IP address by DHCP, setting the static IP address).
- **Serial Flash API**: for networking or user proprietary data.

Sources: [swru368a](#), [swru369c](#)
LAS16 Demo: TI MQTT over SimpleLink on Zephyr

- Ported TI's OS layer to Zephyr kernel:
  - semaphore, mutex, message queue, task and interrupt kernel services
- SimpleLink Host driver:
  - Has dedicated SPI driver to send/recv commands & data to Network Coprocessor
- TI MQTT client -> TI Socket APIs -> SimpleLink Host Driver (libsimplelink.a).

- But this bypasses the Zephyr Native IP Stack
- We'd like Zephyr Networking Apps to leverage SimpleLink w/o modification.
Zephyr Native IP Stack

- **Dual IPv6 & IPv4 stack**
- **Network buffers:**
  - Managed by the kernel, not the app.
  - Chains of fragments, not continuous
  - Shared Rx/Tx net_buf pools
  - Goal is efficient memory usage, eliminate data copies app<->driver.
- **Application APIs:** net_context - socket-like, but not standard
- **Other Features:**
  - Multiple bearer support (BLE / 802.15.4)
  - Multiple adapter instances
  - No adaptation of a 3rd party IP stack.

Source: Native IP Stack for Zephyr OS
“IP Offload” [Experimental] Option in Zephyr

- Kconfig option enables a “vampire tap” to a parallel stack.
- net_context APIs similar to BSD sockets
  - `get()/put()`, `bind()/listen()/connect()/accept()`
  - `send()/sendto()`, `recv()`
- Differences: net_context vs BSD
  - Receive is asynchronous vs synchronous
  - IP stack allocates and manages buffers
  - Application needs to assemble/parse data into/from network buffer chains.
  - No set/get socket options API.
  - Network interface exposed to app via network buffers (not opaque).
Options for Offloading to the SimpleLink coprocessor

Option 1: Full TCP/IP Offload:

- **Pros:**
  - Code/memory/energy savings using full TCP/IP offload to a coprocessor.
  - Zephyr Offload APIs inspired by BSD socket APIs, so mapping possible.
  - Integration will highlight areas where Zephyr IP stack abstractions need to be improved.

- **Cons:**
  - Mapping net_context APIs to BSD sockets adds some inefficiency:
    - net_buf chains vs linear buffers.
    - Offload server to trigger callbacks

Option 2: Write a Zephyr L2 Driver:
(using SimpleLink “Transceiver” Mode)

- **Pros:**
  - Hooks deeply into the Zephyr IP Core, so more functionality is “open”
  - Enables Zephyr use cases like packet routing across network interfaces.

- **Cons:**
  - Does not enable full entitlement of the SimpleLink hardware:
    - Completely offloaded network buffer allocation, management and packet stats
    - DHCP, DNS
    - Secure sockets (via TLS socket opts)
Control Plane APIs, and Connection Security

Independent of the data plane solution, there are other issues:

- **Management APIs:**
  - Currently no standard APIs for setting up WiFi connections, configuring network parameters.
  - Could propose the TI SimpleLink Wlan, and NetCfg APIs for (WiFi) control plane APIs.

- **Secure Sockets:**
  - TLS is handled elegantly in SimpleLink by setting Socket Options, then using socket APIs as normal.
  - Zephyr mbedTLS solution requires complex wrappers around a special socket library.
  - How to reconcile these two models?
Thank You

#BUD17

For further information: www.linaro.org
BUD17 keynotes and videos on: connect.linaro.org