WiFi and Secure Socket Offload in Zephyr™

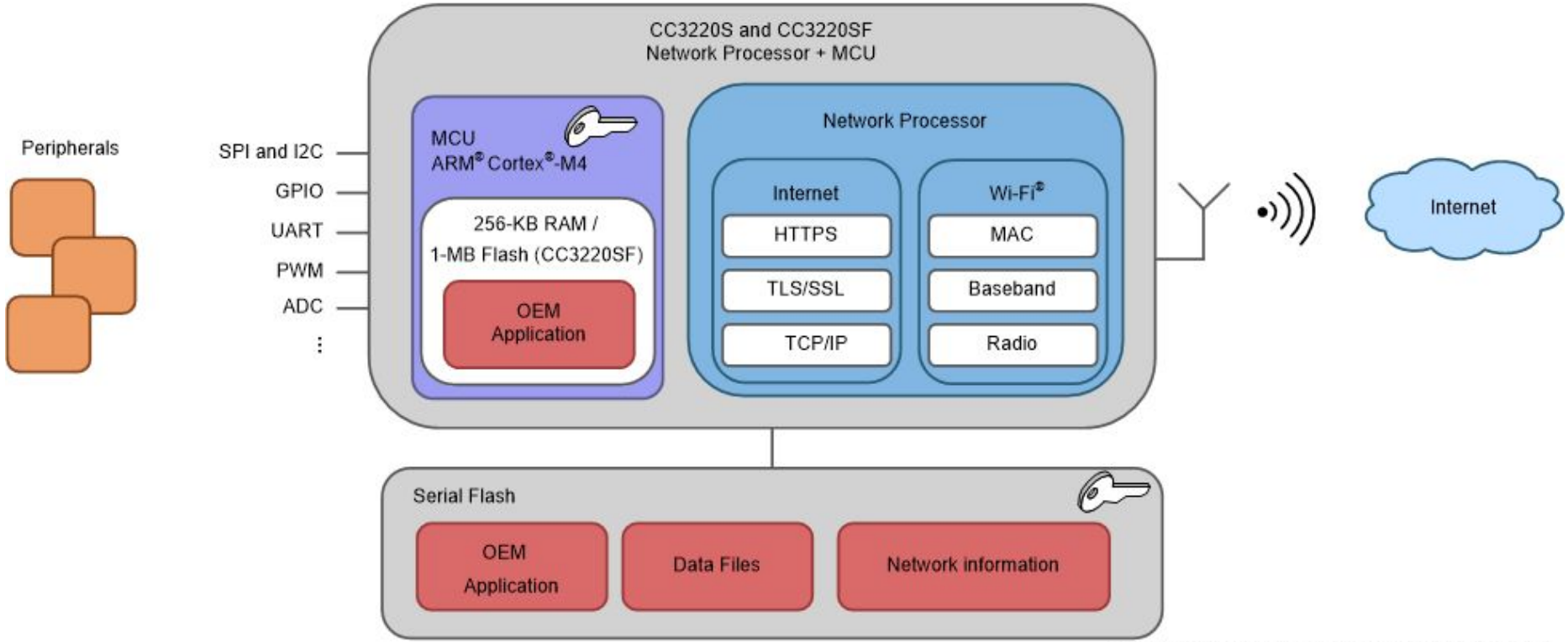
Gil Pitney / Texas Instruments

gpitney@ti.com

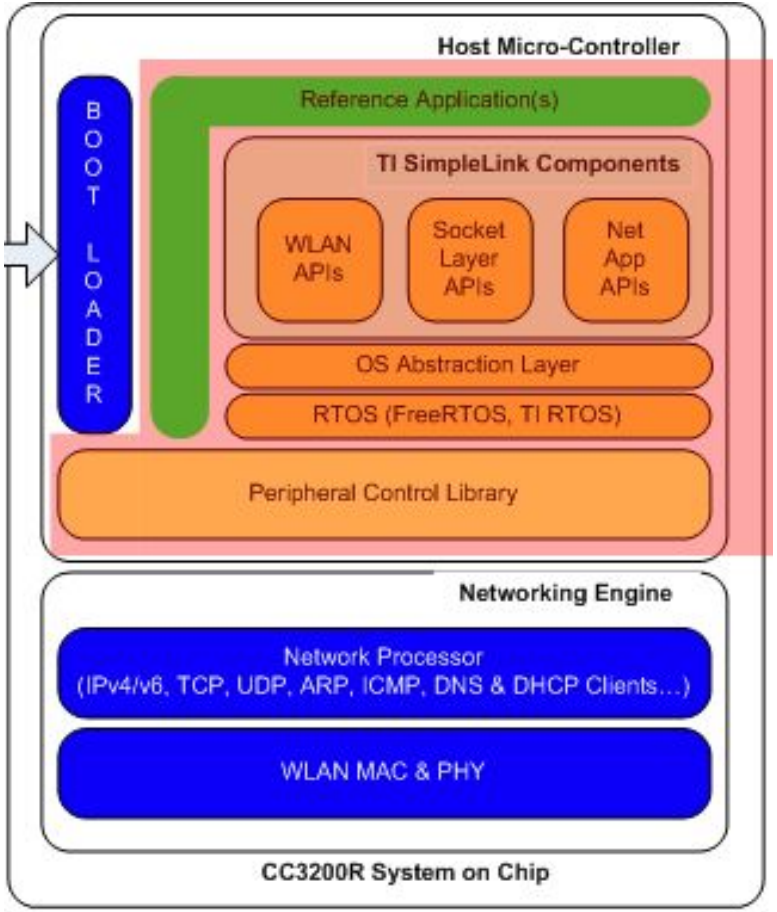
Motivation

- The TI SimpleLink CC32xx family of MCUs provides an SoC and supporting SDK which completely offloads the WiFi stack onto an integrated network coprocessor (NWP).
  - This provides significant memory, CPU, and energy savings.
  - All secure communications, certificate/key storage, crypto and power management is handled on the NWP.
  - The SimpleLink SDK has no support for the Zephyr OS, but is designed to be portable.
- Zephyr networking stack has support for WiFi via an offload tap (data plane), and wifi management events (control plane).
- **The goal is to efficiently integrate the full SimpleLink offload capabilities into Zephyr, while still leveraging Zephyr networking applications and protocols.**
  - All work done on the **CC3220SF-LaunchXL** Development board.

TI CC3220SF SoC H/W Architecture



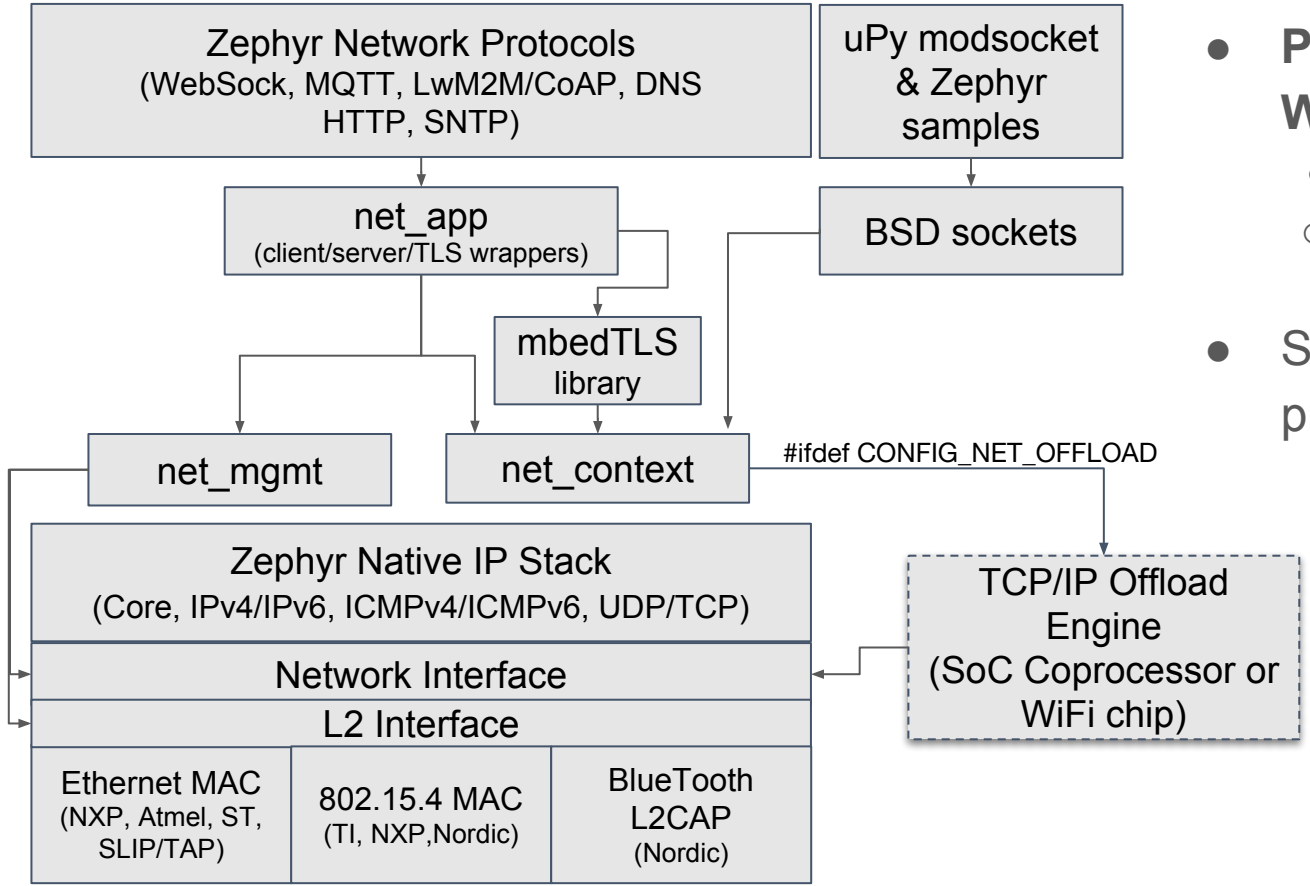
TI SimpleLink CC32xx SDK Architecture & APIs



- 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: Standard API, but TLS handled **under** the Socket 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: swru368, swru369c

Zephyr Network Stack (Previous State)



- **Plan has been to support WiFi via offload chips.**
  - No WiFi L2 Drivers
  - No WiFi supplicant, or provisioning support (yet).
- Secure comms (SSL/TLS) provided by mbedTLS library

Options for Offloading to the WiFi coprocessor (1/2)

Option 1: Full TCP/IP Offload:

- How:
  - Port SimpleLink NWP driver -> Zephyr
  - `#include <SL_SDK>/simplelink.h`
  - `#include <SL_SDK>/sys/socket.h`
- Pros:
  - Offers fullest H/W entitlement.
  - Zephyr apps get full access to SimpleLink WLAN, NetApp, Socket APIs.
  - Can still use Zephyr drivers: I2C, GPIO..
- Cons:
  - No integration with Zephyr WiFi event management.
  - Will not leverage Zephyr's socket-based network protocols.

Option 2: Write an L2 Driver:

- How:
  - Use SimpleLink Raw Sockets.
  - Implement L2 send(), reserve() fxns.
  - Push received data via net_pkt to Zephyr IP core.
- Pros:
  - Hooks deeply into the Zephyr IP Core.
  - Enables Zephyr use cases like packet routing across network interfaces.
- Cons:
  - Does not leverage SimpleLink:
    - network buffer allocation, management
    - DHCP, DNS offloaded
    - Secure socket offloading

Options for Offloading to the WiFi coprocessor (2/2)

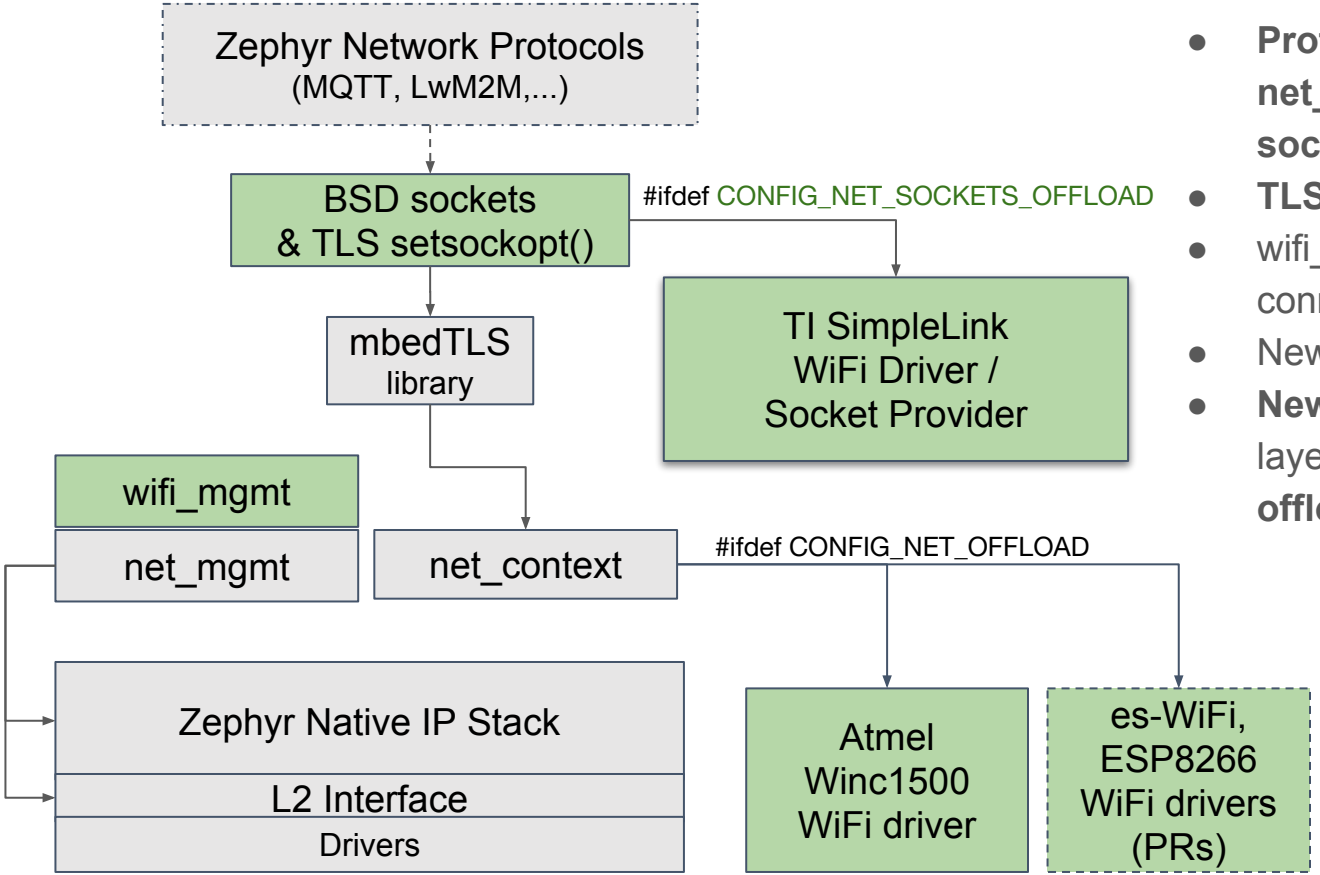
Option 3: Offload at net_context():

- How:
  - Enable `CONFIG_NET_OFFLOAD`
  - Write a full Zephyr WiFi driver
- Pros:
  - TCP/IP is offloaded to the coprocessor.
  - Enables Zephyr use cases like packet routing across network interfaces.
- Cons:
  - Overheads:
    - Mapping sync BSD socket APIs to async net_context APIs and back.
    - Received data copied into net_bufs and queued.
    - Driver thread to poll sockets and trigger callbacks
  - **Security: Crypto and TLS handshake are not offloaded**

Option 4: Offload at BSD socket layer:

- How:
  - Enable `CONFIG_NET_ SOCKETS_OFFLOAD`
  - Write a Zephyr WiFi driver (control only)
  - Register offloaded socket fxns w/ Zephyr.
- Pros:
  - Avoids overheads of Option 3)
  - **Secure socket communications get fully offloaded.**
  - DNS offloaded too (`getaddrinfo()`)
- Cons:
  - Currenlty, only one socket provider in the system
  - No packet routing across net interfaces.

Zephyr Network Stack (New State)



- **Protocols being migrated from net_app/net_context to BSD sockets.**
- **TLS added to socket APIs**
- wifi_mgmt handles scan, connect/disconnect events
- New WiFi net offload drivers
- **New offload tap** at BSD socket layer, allowing full **secure socket offload**.

Zephyr Socket APIs + TLS

- Why?
  - TLS is hard to get right; many TLS library APIs and configuration options.
  - Should make it easy to add TLS to non-secure **socket-based** networking apps/protocols.
- Adding TLS to a networking app via mbedTLS involves:
  - Creation/initialization of mbedtls ssl, config contexts, registration of entropy generator.
  - Certificates setup.
  - Socket creation (standard POSIX); then connection via `mbedtls_net_connect()`
  - Configuration of the TLS/SSL layer.
    - Set server/clilent mode
    - Set certificate authentication mode
    - Specify RNG and DBG functions
    - Set network tx/rx functions via `mbedtls_ssl_set_bio()`
  - Read/Write via `mbedtls_ssl_read()`/`mbedtls_ssl_write()`
  - Teardown of mbedtls contexts.
- Zephyr wrapped all this with net_app, but we want to leverage standard APIs...

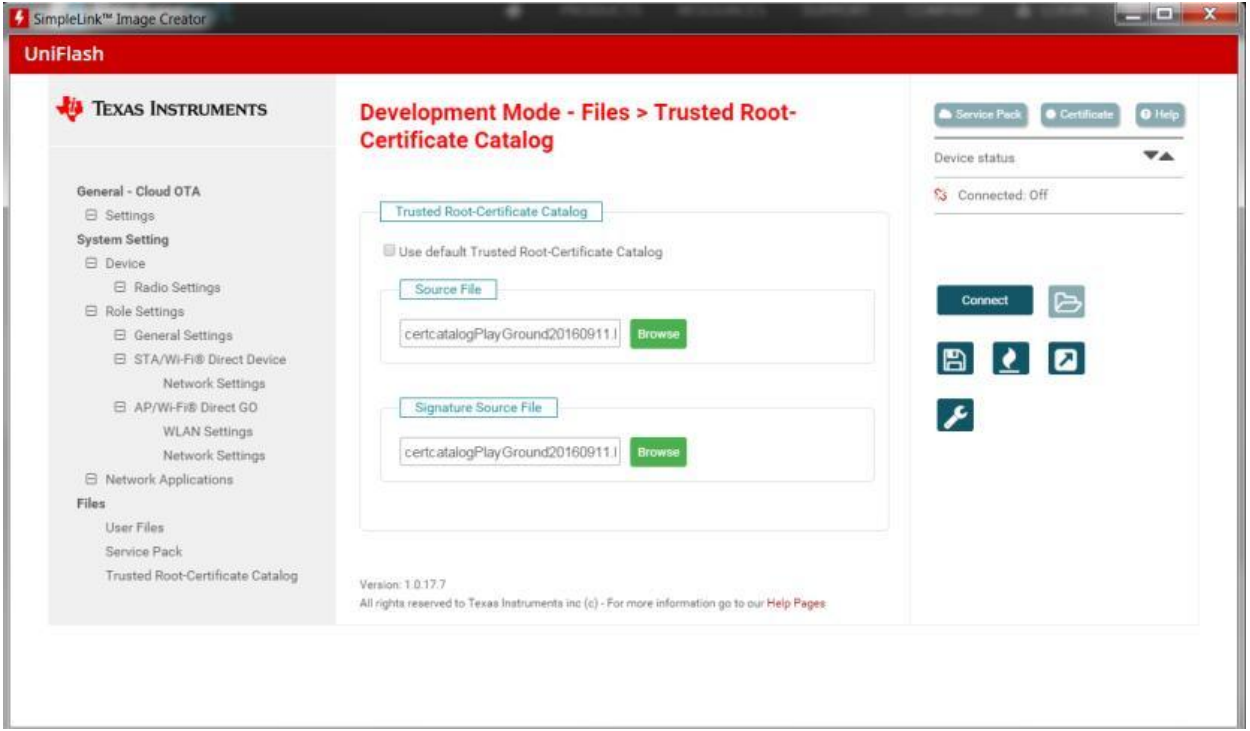
Idea: Encapsulate TLS under POSIX Socket APIs

```
#include <net/socket.h>

hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(HTTP_HOST, HTTP_PORT, &hints, &res);

#if defined(CONFIG_NET_SOCKETS_SOCKOPT_TLS)
sock = socket(res->ai_family, res->ai_socktype, IPPROTO_TLS_1_2);

setsockopt(sock, SOL_TLS, TLS_SEC_TAG_LIST, sec_tag_opt, sizeof(sec_tag_opt));
setsockopt(sock, SOL_TLS, TLS_HOSTNAME, HTTP_HOST, sizeof(HTTP_HOST)))
#endif /* CONFIG_NET_SOCKETS_SOCKOPT_TLS */

connect(sock, res->ai_addr, res->ai_addrlen);
send(sock, REQUEST, SSTRLEN(REQUEST), 0);
recv(sock, response, sizeof(response) - 1, 0);
close(sock);
```

Also, need to “provision” certificates/keys up front

```
/* In main() somewhere at program initialization: */

#if defined(CONFIG_NET_SOCKETS_SOCKOPT_TLS)
#include <net/tls_credentials.h>

#define CA_CERTIFICATE_TAG 1
/* GlobalSign Root CA - R2 for https://google.com */
static const unsigned char ca_certificate[] = {
#include "globalsign_r2.der.inc"
};

tls_credential_add(CA_CERTIFICATE_TAG, TLS_CREDENTIAL_CA_CERTIFICATE,
                   ca_certificate, sizeof(ca_certificate));
sec_tag_t sec_tag_opt[] = {
     CA_CERTIFICATE_TAG,
};

#endif /* CONFIG_NET_SOCKETS_SOCKOPT_TLS */
```

Provisioning Certificates/Keys on CC3220SF



TI UniFlash Tool:

- Burns certificates/private keys into secure flash.
- Keeps a catalog of Trusted Root Certificates
- Eg: Add the cloud server's root certificate to secure flash.

At runtime:

- bind certificate's **filename** to client socket using `setsockopt()`
- MCU apps have no access to the "secrets".

For CC3220SF, only need provide cert/key filenames

```
#include <net/tls_credentials.h>

#define CA_CERTIFICATE_TAG 1
/* GlobalSign Root CA - R2 for https://google.com */
static const unsigned char ca_certificate[] = "global_sign_root_ca_cert"

/* Rest of networking application is the same: */
tls_credential_add(CA_CERTIFICATE_TAG, TLS_CREDENTIAL_CA_CERTIFICATE,
                   ca_certificate, sizeof(ca_certificate));
sec_tag_t sec_tag_opt[] = {
     CA_CERTIFICATE_TAG,
};

...
```

Summary

- The **TI SimpleLink CC3220SF SoC** allows the TCP/IP stack, WiFi, secure communications, encryption, secrets storage and power management to be offloaded from the MCU (Zephyr) to an integrated network coprocessor (NWP).

- The SimpleLink SDK's "NWP driver" is ported to Zephyr via a thin OSAL.
- A SimpleLink Zephyr WiFi driver implements the wifi_mgmt API, and sends connect/disconnect/scan notifications back to the network event manager.
- Certificates are provisioned to CC3220SF **secure flash** offline via TI UniFlash.
- The SimpleLink Zephyr WiFi driver hooks into the (new) **Zephyr socket offload tap**, and with the help of Zephyr's TLS addition to the BSD APIs, can achieve **fully secure socket offload**, available to Zephyr **socket-based** net protocols.

Thank You!

THE LINUX FOUNDATION