Native IP Stack For Zephyr™ OS

Why a native IP stack?

- Features missing in current Contiki based uIP stack
  - No dual IPv6 & IPv4 stack
  - Only one bearer (BLE / 802.15.4) usable at a time
  - One adapter only (like no 2 x 802.15.4 device)
  - Memory management issues (too big buffers for small amount of networking data)
  - Difficult to adapt uIP to multithreaded Zephyr OS
- Automatic testing of the network stack

Why not port 3rd party IP stack?

- Use native stack instead of embedding existing stack like lwip or FNET

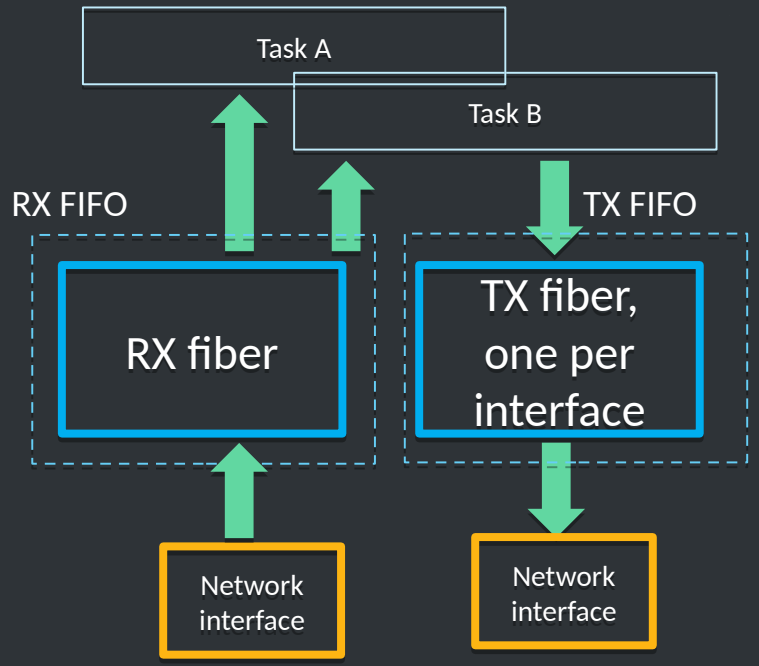
  > avoid adaptation layers which cause overhead
  > use same network buffers everywhere to utilize memory efficiently
  > unified look and feel of the APIs and code (helps maintenance)
  > creating testing harness easier with new stack

How to do it?

- Re-write IPv6 and IPv4 components to enable dual stack
- Avoid any adaptation layers by utilizing Zephyr OS services directly
- Utilize memory better by enabling IP stack to use smaller buffers that can be linked together
- Creating a testing framework to test the stack automatically

Key Features

- Native dual IPv6 and IPv4 stack
- Built around network buffer pools concept, efficient memory management
- Possibility to have Zero/minimal copy data path from user space to device driver
- Supports Thread IP requirements
- Native KConfig integration
- Uses automated testing harness for running network stack tests



One RX fiber

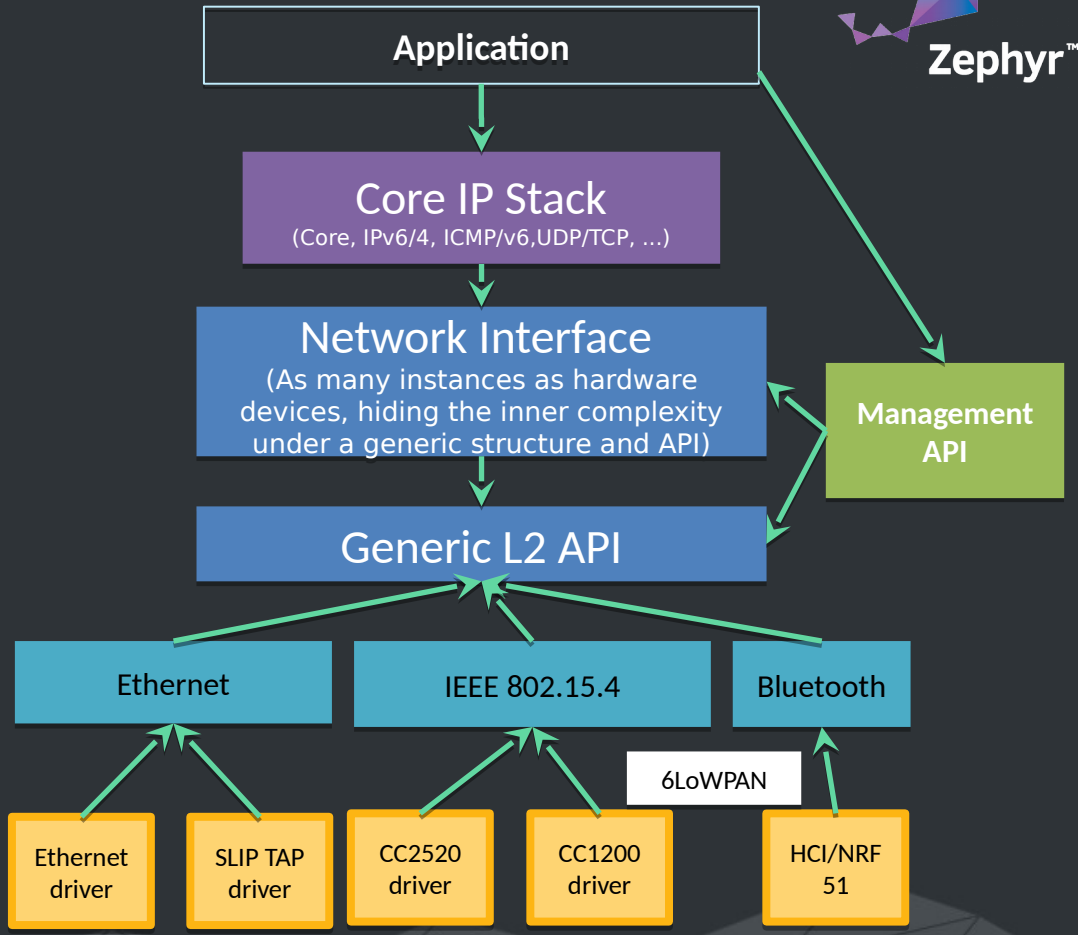
- Global RX FIFO for receiving
- array of network contexts (“sockets”)
- Look up Tasks’ Rx FIFOs

Multiple TX fibers

- Each network device has its own TX fiber
- Write to device drivers TX FIFO

Key L2 features

- Dedicated OSI L2 abstraction
- Supports multiple bearers and interfaces such as IEEE 802.15.4 and Bluetooth*
- Concurrent TX/RX on all interfaces
- Spec-compliant 802.15.4 implementation
- Supports current Zephyr OS network drivers with small modification

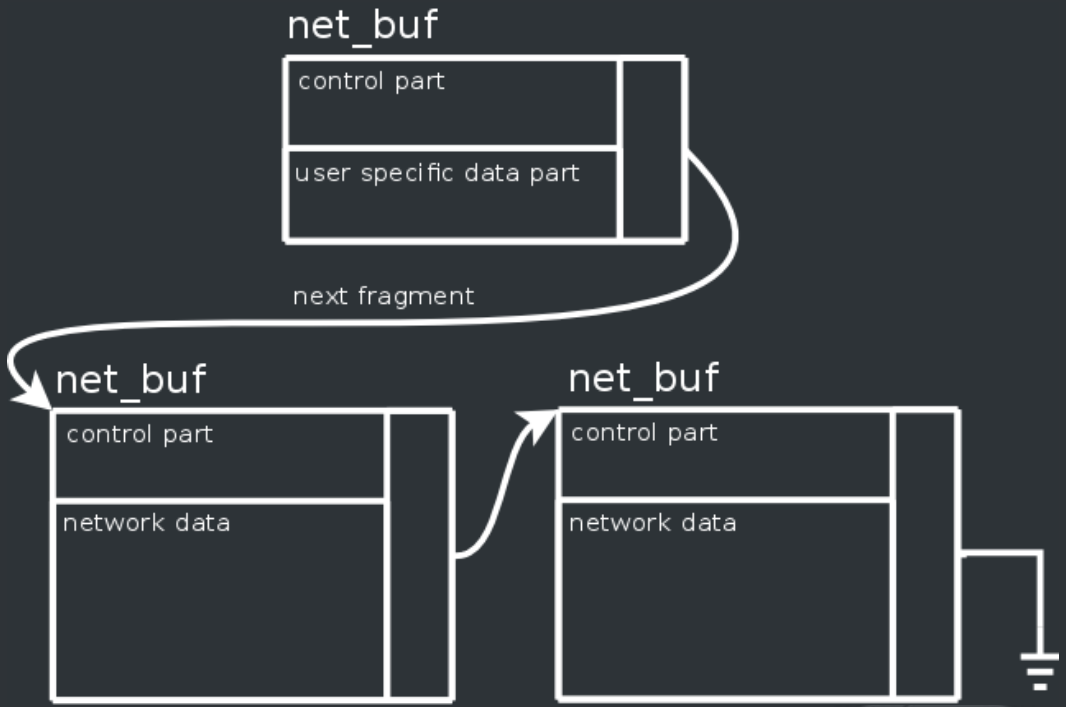


Automatic Testing

- Component tests created that will test some specific part of the networking stack (example: 6lo, neighbor mgmt, ip-addr mgmt etc.)
- These tests are executed automatically for each commit
- Full network stack conformance testing run periodically

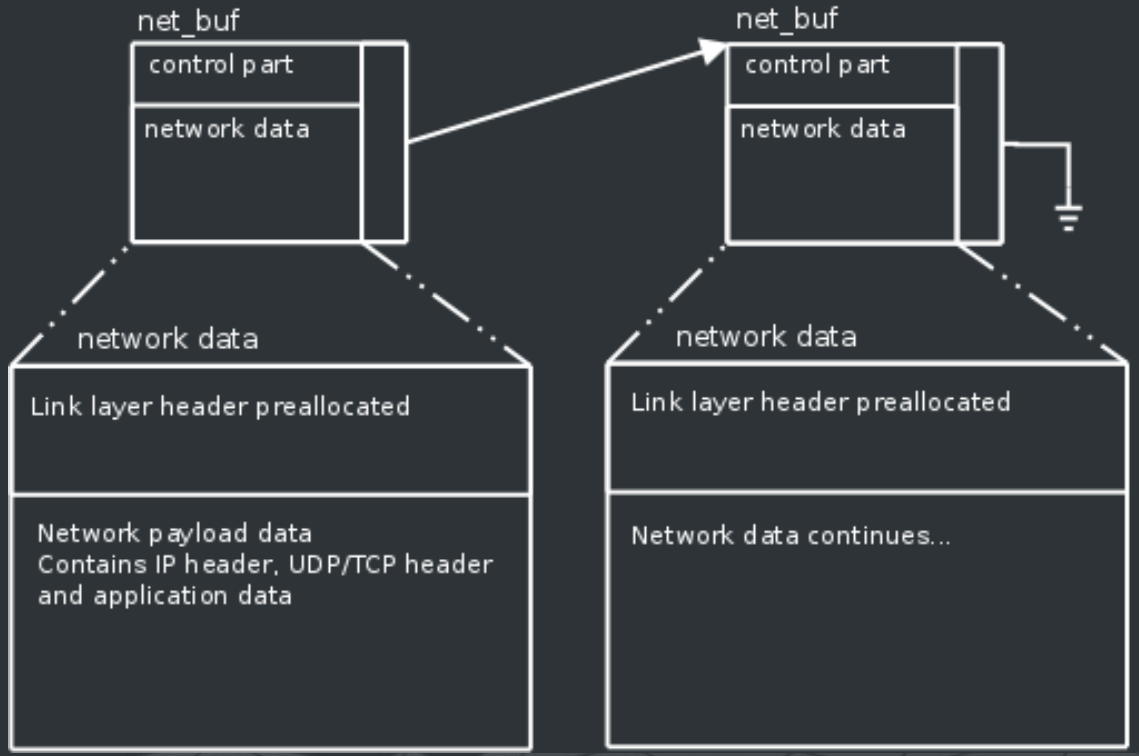
Network buffer overview

- net_buf's can be chained together to allow sending or receiving bigger amount of data
- Amount of fragments is configurable by Kconfig option
- Size of the network data part of the fragment can be configured via Kconfig
- User specific data part in the first fragment can contain protocol specific information
- Example: typical size of the network data part is 128 bytes for 802.15.4



Network buffer details

- Link layer header pre-allocated in the net_buf
- Requires application data partitioning when sending the data
- When receiving, the application needs to read the data in chunks as the data is not continuous
- The L2 layer will insert link layer data to the start of the net_buf



Release Plan

- Currently native IP stack code under development can be found at “net” branch in Zephyr OS git [1]
- Merge native IP stack to Zephyr OS 1.7 release

[1] https://gerrit.zephyrproject.org/r/gitweb?p=zephyr.git;a=tree;h=refs/heads/net;hb=refs/heads/net


Zephyr™
Thank you!