Goals

- Network packets can be large, zero-copy wherever possible
- Support all hardware, don’t make design decisions that rule out a class of devices
- Avoid runtime overheads
  - Both copying (compute) and spurious memory allocations
- Think about dynamic allocations
  - Tock does not have precedent for dynamic (heap) allocations
    - Avoids runtime exhaustion
  - Current dynamic design uses grants, which ties it to userspace

Hardware Expectations: LiteEth MAC

- Uses “DMA”: exposes two SRAM ring buffers in the address space
    - Packets can’t wrap in the buffers
- Requires contiguous allocations for TX
- Provides memory-mapped contiguous buffers for RX

Hardware Expectations: VirtIO MAC

- Hardware & software share “descriptor” ring-buffers
  - Each descriptor points to a buffer at an arbitrary location in memory
  - Reception works by putting empty buffers into a descriptor ring
    - Hardware will return filled buffers to the software
  - Transmission works by putting full buffers into a descriptor ring
    - Hardware returns sent buffers to software
- Received & transmitted packets can be arbitrarily split across multiple non-contiguous memory allocations
  - Put all buffers into a single descriptor to mark them as one “virtual” buffer
- Packets require a VirtIO-Net specific header

Hardware Expectations: STM32 / NXP iMX.RT1060 MAC

- Similar descriptor ring-buffers
  - Much more constrained: minimum (& maximum?) buffer sizes
  - Packets can’t be split into arbitrarily small buffers: DMA bandwidth & latency
- Alignment constraints on buffers
- Packets require special MAC-specific header

Hardware Expectations: external MAC (e.g., ENC28J60)

- Connected using some serial bus
- Typically small internal ring-buffers which require packets to be contiguous
- Transmission / reception buffer copying are asynchronous operations in Tock

Hardware Expectations: USB EEM

- *Defer to Amit*

Hardware Expectations: CAN

- *Defer to OxidOS folks*
- *Fixed length frames, so slots (similar to LiteEth)*
- *FIFO registers*
- *Big chunk of memory reserved for CAN*

Hardware Expectations: 15.4 / 6LoWPAN / Thread

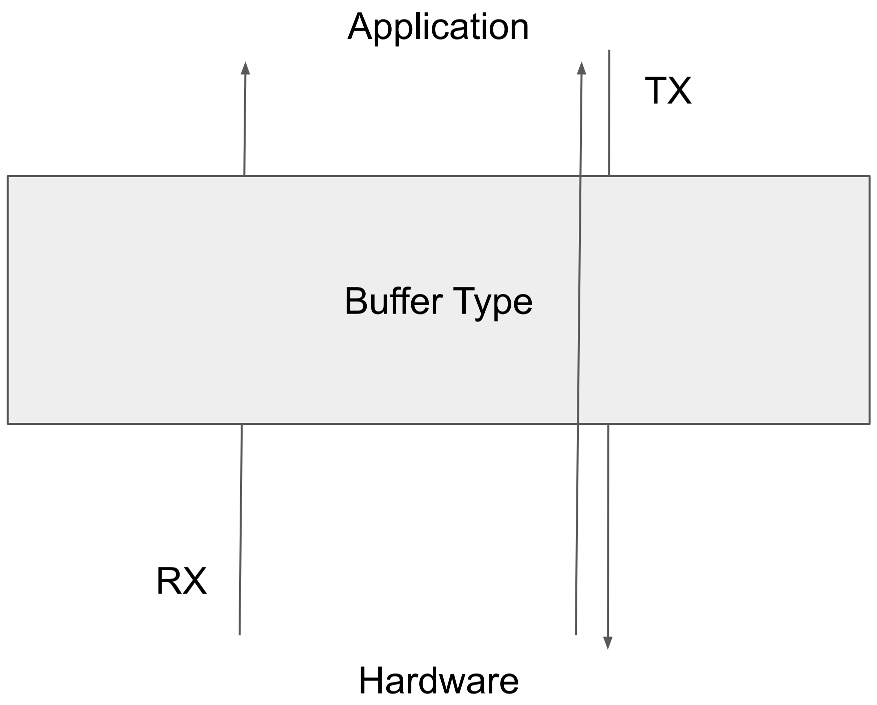
- *Defer to Tyler*
- *Hardware-expectations stem from the 15.4 layer*
    - *DMA, Nordic: pointer to location in memory, radio copies into buffer in RAM*
    - *Branden: Concept of FIFO or double buffering? -> Set DMA pointer after reading a packet on Nordic*
        - *Received buffers are passed to the client, copied to its own internal buffer*
    - *Branden: RF233?*
        - *Can't speak to that*
- *Memory is passed (copied) from layer to layer*

The current “Ethernet HIL”

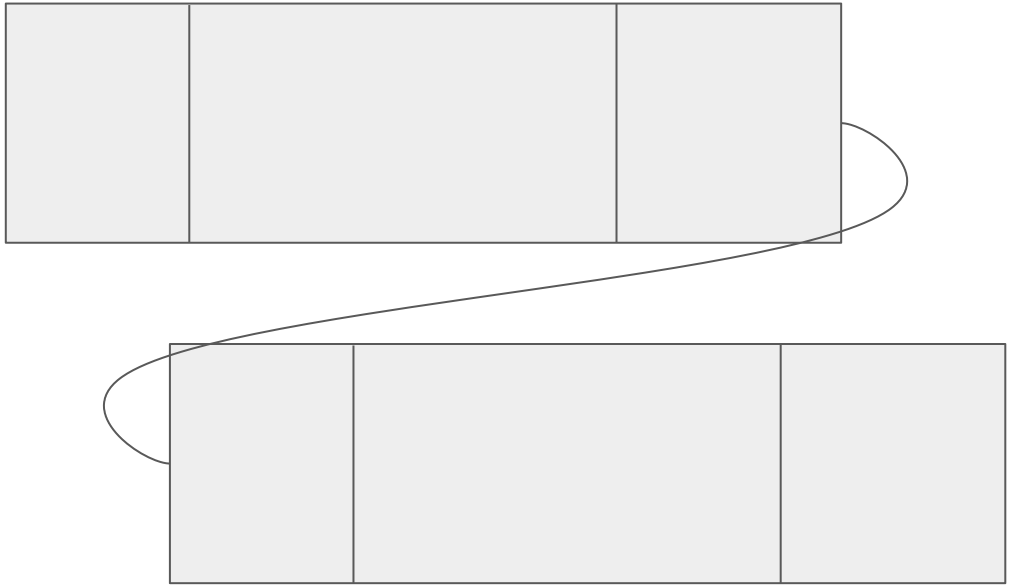
```
pub trait EthernetAdapterClient {
      fn tx_done(
            &self,
            err: Result<(), ErrorCode>,
            packet_buffer: &'static mut [u8],
            len: u16,
            packet_identifier: usize,
            timestamp: Option<u64>,
      );
      fn rx_packet(&self, packet: &[u8], timestamp: Option<u64>);
}

pub trait EthernetAdapter<'a> {
      fn set_client(&self, client: &'a dyn EthernetAdapterClient);
      fn transmit(
            &self,
            packet: &'static mut [u8],
            len: u16,
            packet_identifier: usize,
      ) -> Result<(), (ErrorCode, &'static mut [u8])>;
}
```

“PacketBuffer” Proposal

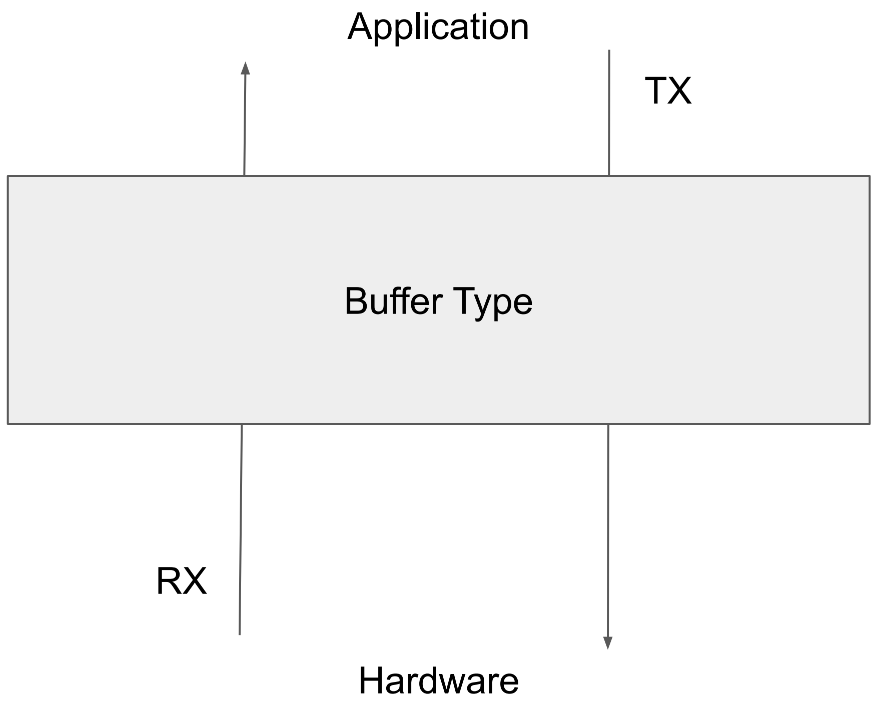


PacketBuffer<16, true>



Mbuf: linked list of $things
Mpullup: make this a contiguous allocation

“PacketBuffer” Proposal


Application
TX
Buffer Type
RX
Hardware