Verified IPv6 Network Stack

Felix Gibeault, *Alain Kägi*, Wade McDermott,
Daniel Neshyba-Rowe,
Ellen Whalen, Elias Williams

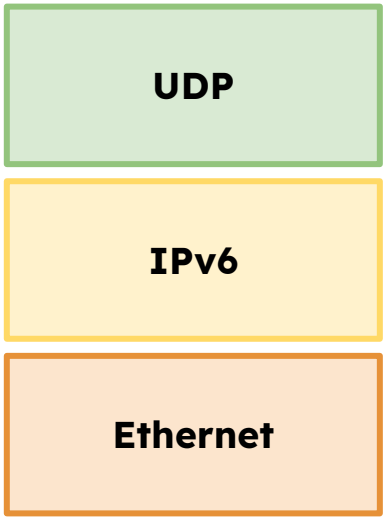
Lewis & Clark College

The Vision

- Completely verified single-function, networked devices
  - From the gate-level design to the application
  - Similar to Deep Specification project but applied cyber-physical systems
- Opportunities for optimization
  - Performance
  - Energy
  - Cost
  - Balanced system

A Start

- IPv6 network stack
  - Geared towards end-points
  - High-performance
  - Functionally verified
- Opportunities
  - Responsible for both code & proofs
  - Network layers are best effort
  - Optimizations



Big Picture

```
User Datagram Header Format
---------------------------

0       7 8     15 16    23 2431
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
...
```

```
udp_unwrap ∘ udp_wrap ≡ id
...
```

Invariants

RFCs, IEEE stds

Manual

Abstract Specification

```
definition wrap_buf ::
  ‹sk_buf
   ⇒ (unit, sk_buf, 's) spec_monad›
  where ‹wrap_buf sk ≡
    do {
      msg    ← return (read_buf sk);
      header ← return (...);
      new_sk ← return (sk ⟨ ... ⟩);
      return new_sk
    }›
```

Refinement

```
void wrap(struct sk_buf *skb) {
  size_t sz = size_of(struct header);
  struct header *hd = skb->first - sz;
  ...
```

C Code

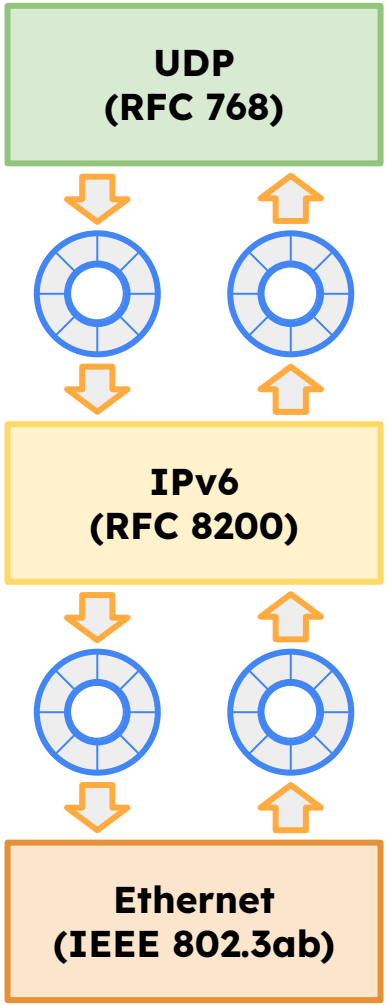
Automatic

C Specification

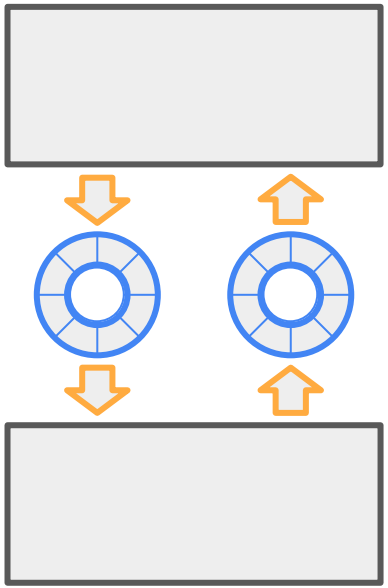
```
wrap' ?skb ≡
do {
  guard (λs. IS_VALID(sk_buf_C) s ?skb);
  modify (heap_sk_buf_C_update ...);
  p ← gets (λs. PTR_COERCE(8 word → header_C)(first_C (heap_sk_buf_C s ?skb)));
  ...
```

Software Architecture

- Communicating sequential processes [Hoare 1978]
  - Erlang, Limbo, Go, …
  - Non-zero fixed-length queues
  - Separate transmit/receive queues
  - Reduced verification burden
  - Great match with seL4 microkit
- Zero copy
  - Socket buffers
  - Performance
- Best effort

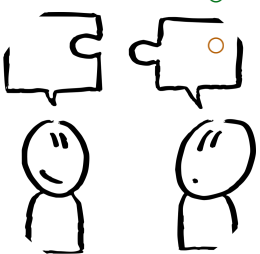


Communicating Sequential Processes

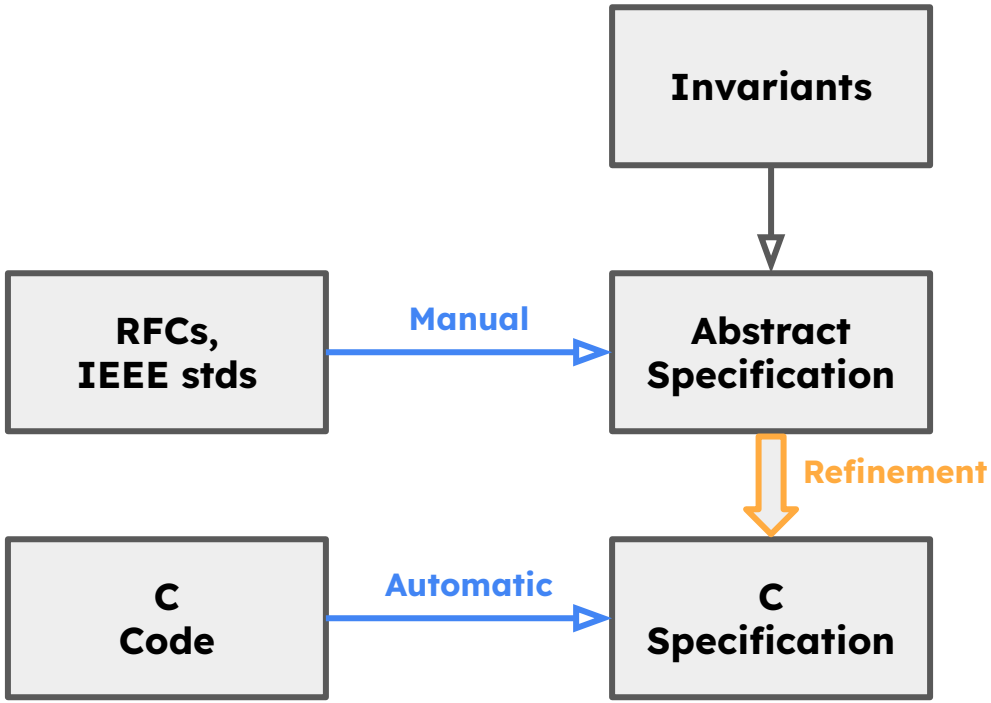


- Structure, Performance, Verification

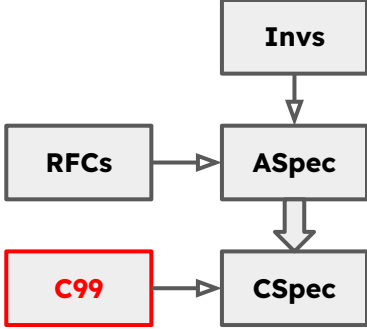
- *Independent*
  - Structure concurrent processes
  - Execute processes concurrently
  - Ease verification
- *Asynchronous send/synchronous receive*
  - Coordination
  - Resource management
  - Reason about processes separately until explicit receive



Verification

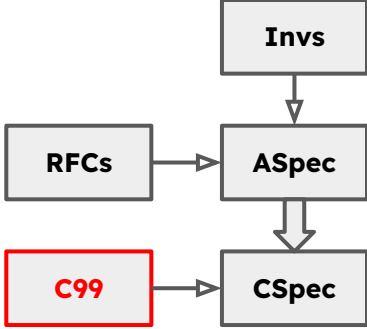


Socket Buffer



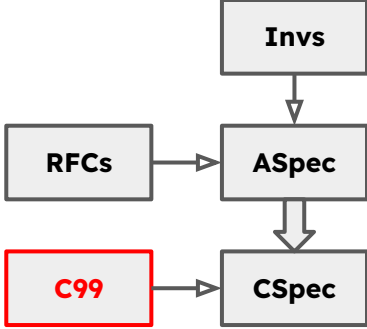
Retired electrician, CC0, via Wikimedia Commons

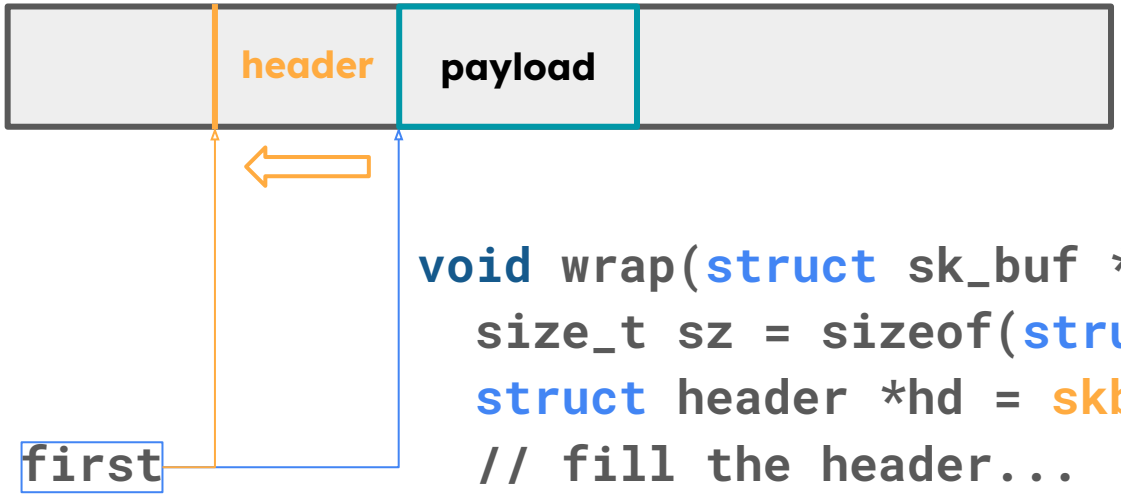
Socket Buffer




```
struct sk_buf {
  uint8 *begin;
  uint8 *end;
  uint8 *first;
  uint8 *last;
  struct socket *src;
  struct socket *dst;
};
```

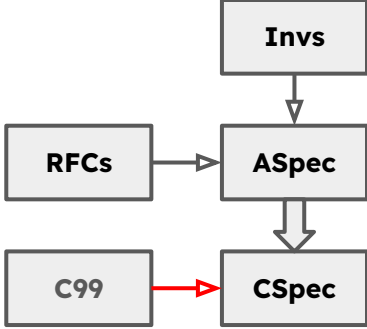
Socket Buffer






```
void wrap(struct sk_buf *skb) {
  size_t sz = sizeof(struct header)
  struct header *hd = skb->first - size;
  // fill the header...
  skb->first = hd;
}
```

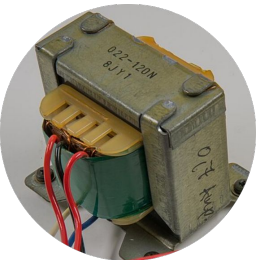
AutoCorres2



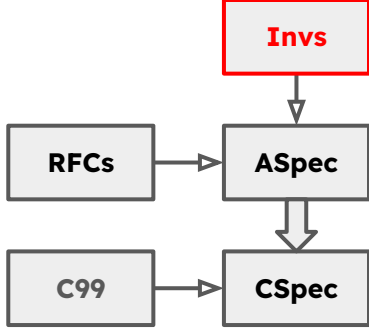
[Greenaway 2014]

```
wrap' ?skb ≡
do {
  guard (λs. IS_VALID(sk_buf_C) s ?skb);
  modify (heap_sk_buf_C_update ...);
  p ← gets (λs. PTR_COERCE(8 word → header_C)(first_C (heap_sk_buf_C s ?skb)));
  ...
}
```

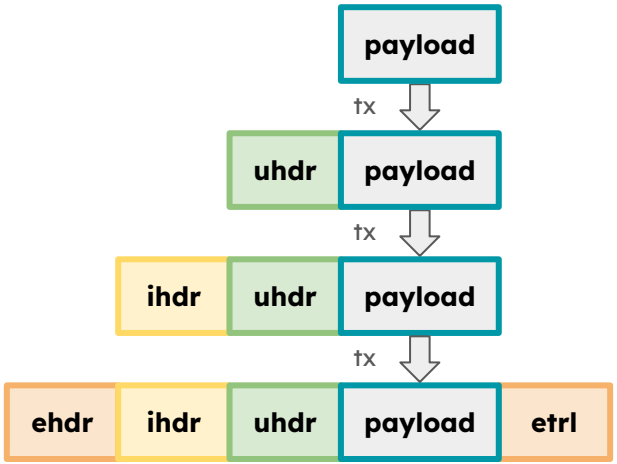
```
void wrap(struct sk_buf *skb) {
  size_t sz = sizeof(struct header)
  struct header *hd = skb->first - sz;
  ...
}
```

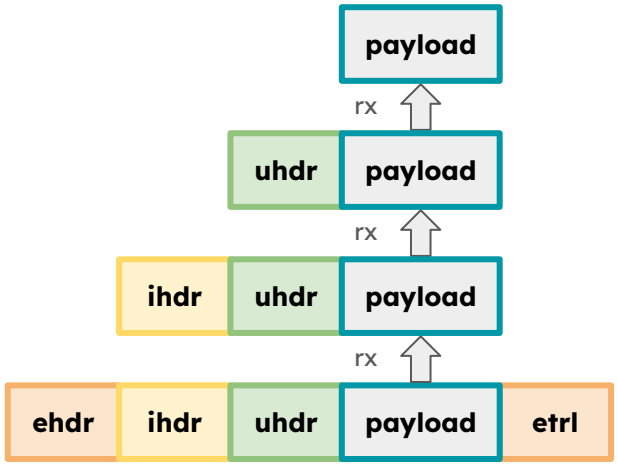


Invariants

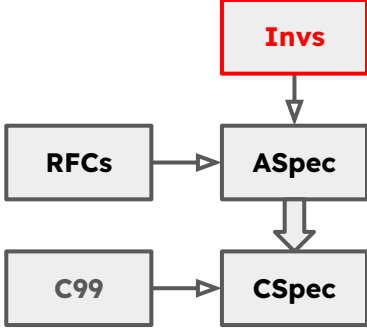


- $f^{-1} \circ f = id$
  - `udp_unwrap` ∘ `udp_wrap` = *id*
  - `ip_unwrap` ∘ `ip_wrap` = *id*
  - `eth_unwrap` ∘ `eth_wrap` = *id*

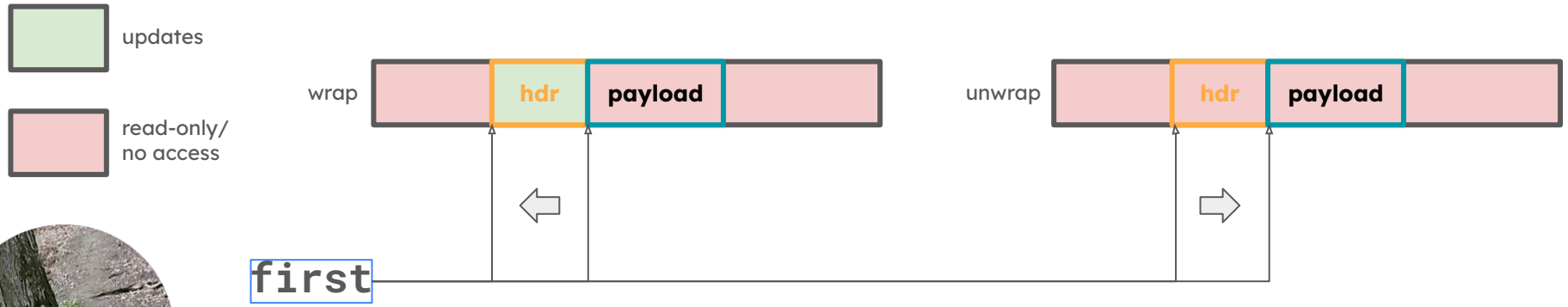




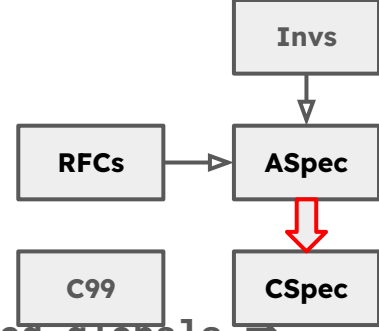
Invariants



- buffer
  - wrap
    - Only `first` and the content of the header change
  - unwrap
    - Only `first` changes



Refinement: Relation




```
definition udp_wrap_R :: ‹sk_buf_C ptr ⇒
                          (unit, unit) exception_or_result × lifted_globals ⇒
                          (unit, sk_buf) exception_or_result × 's ⇒
                          bool›
  where ‹udp_wrap_R sk_buf_c ≡ λ(er', s') (er, s). (
        case er of
          (Exception e) ⇒ True |
          (Result r)    ⇒ case er' of
            (Exception e') ⇒ False |
            (Result r')    ⇒ (sk_buf_c_valid sk_buf_c s') ∧
                             (sk_buf_equiv sk_buf_c s' r))›
```

Status

- UDP layer
  - Latest stumbling block: Pointer coercion
- Ethernet and IP layers
  - Sketch
  - Abstract UDP proof and reuse
- Communication
  - Framework in place

Future Plans

- ICMP and TCP protocols
  - Stateful
- Security extensions
  - Confidentiality, Integrity, Availability

Summary

All possible behaviors of the C code permitted by the specification, in particular,

- No no-null pointer dereference
- No buffer overflow
- No memory leaks
- No undefined behaviors (e.g., UINT_MAX + 1, etc.)
- No infinite loops/recursion

Acknowledgements

- John S. Rogers Science Program
- Margaret Salstrom, Amy Timmins
- Dylan Angel, Aubrey Birdwell, Linus Brogan, Christian Ermann, Michael Harper, Seth Leichsenring, Jorge Martinez, Levi Overcast, Sydney Parker, Wyeth Greenlaw Rollins, Caitlyn Wilde, Natalie Zoz
- Johannes Åman Pohjola, June Andronick, Matthew Brecknell, Gernot Heiser, Gerwin Klein, Lucy Parker, Ivan Velickovic
- Jens Mache, Richard Weiss