An open source user space fast path TCP/IP stack

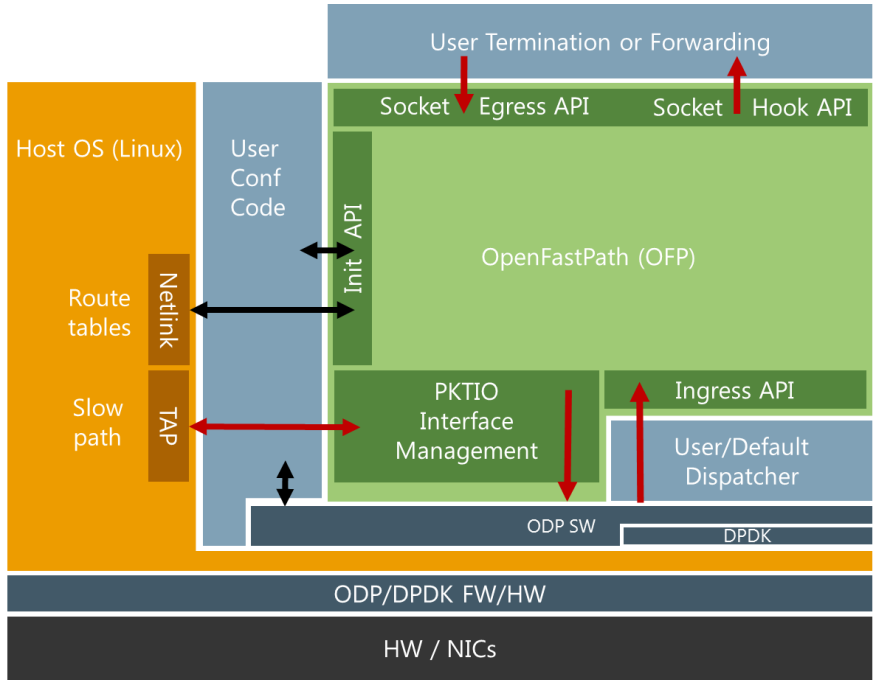
Industry network challenges

- **Growth in data traffic** means that even small network nodes needs a fast path
  - The Linux IP stack is **slow** and **does not scale**
- High **throughput IP processing** solutions has been around for a **number of years**
  - Why this now?
- Most existing implementations are either **hardware specific** or **proprietary closed source**
  - SoC vendor solutions and for example 6Wind
- Developing this **basic building** block from scratch in-house does **not make sense**
  - Not even for the big network equipment providers

Enter OpenFastPath!

A TCP/IP stack

- lives in **user space**
- is optimized for **scalability** and **throughput**
- uses Data Plane Development Kit (**DPDK**) and
- Open Data Plane (**ODP**) to access network hardware
- runs on **ARM, x86, MIPS, PPC** hardware
- runs **natively**, in a **guest** or in the **host** platform



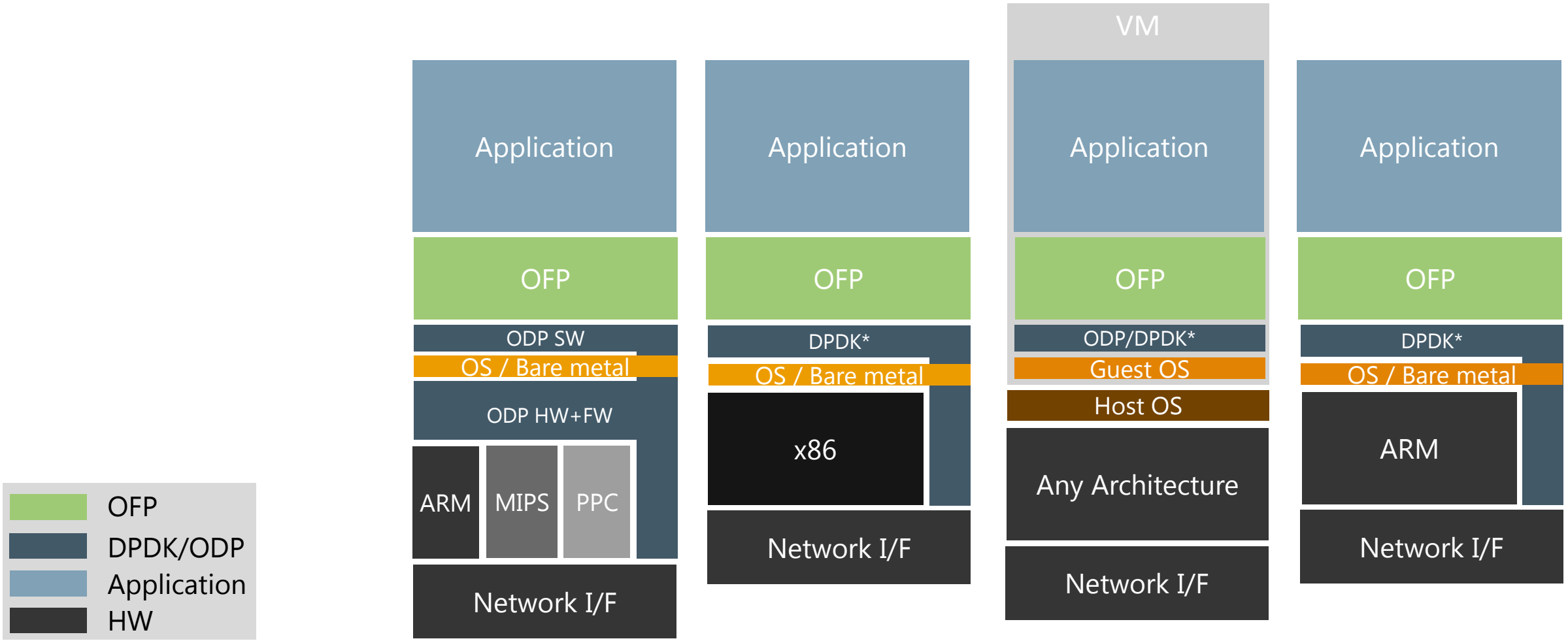
The OpenFastPath project

- is a **true open source project**
- uses well known **open source components**
- open for all to participate – **no lock-in** to HW or SW
- **Nokia**, **ARM** and **Enea** key contributors

ENEA ARM

NOKIA

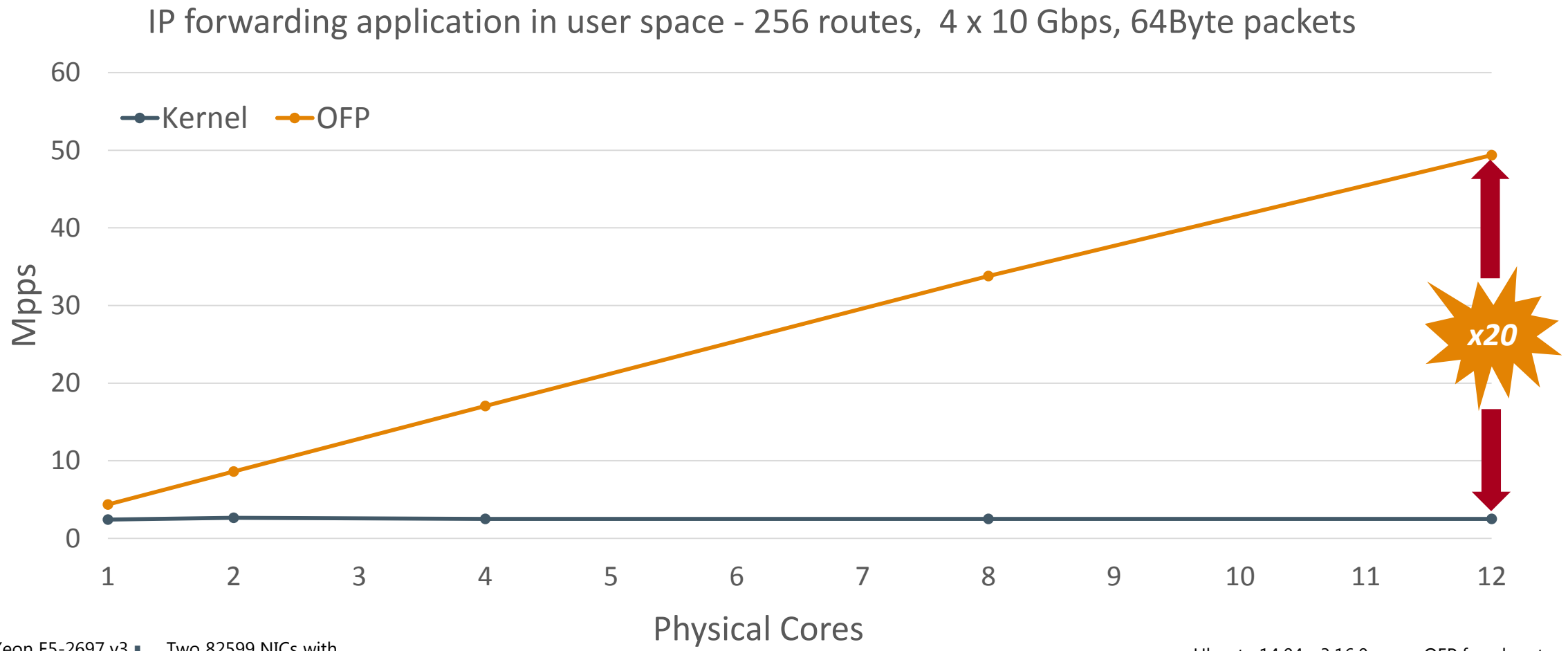
A main benefit with OFP is portability….



* Native support in execution

…. AND

…performance - OFP is 20x Linux TCP/IP stack!



- Intel Xeon E5-2697 v3 processor (turbo disabled)
- Two 82599 NICs with modified netmap ixgbe 4.1.5 driver (12 rx/tx queue pairs) totaling 4x10Gbps ports
- Ubuntu 14.04 - 3.16.0-53-generic. CPU isolation used to test kernel IP forwarding.
- OFP fpm_burstmode example application
- ODP 1.4.1.0 ext. with multi queue packet I/O support

Features implemented

Fast path protocols processing:

- Layer 4: UDP termination, TCP termination, ICMP protocol
- Layer 3
  - ARP/NDP
  - IPv4 and IPv6 forwarding and routing
  - IPv4 fragmentation and reassembly
  - VRF for IPv4
  - IGMP and multicast
- Layer 2: Ethernet, VLAN
- GRE and VXLAN Tunneling

Routes and MACs are in sync with Linux

Integration with Linux Slow path IP stack through TAP interface

Command line interface

- Packet dumping and other debugging
- Statistics, ARP, routes, and interface printing
- Configuration of routes and interfaces with VRF support

OFP IP and ICMP implementations passing Ixia conformance tests

IP and UDP implementations has been optimized for performance

- TCP implementation is functional but not performance optimized

Integrated with NGiNX webserver

OpenFastPath

OpenFastPath Source code

New open-source code

- Developed by partners during the incubation stage

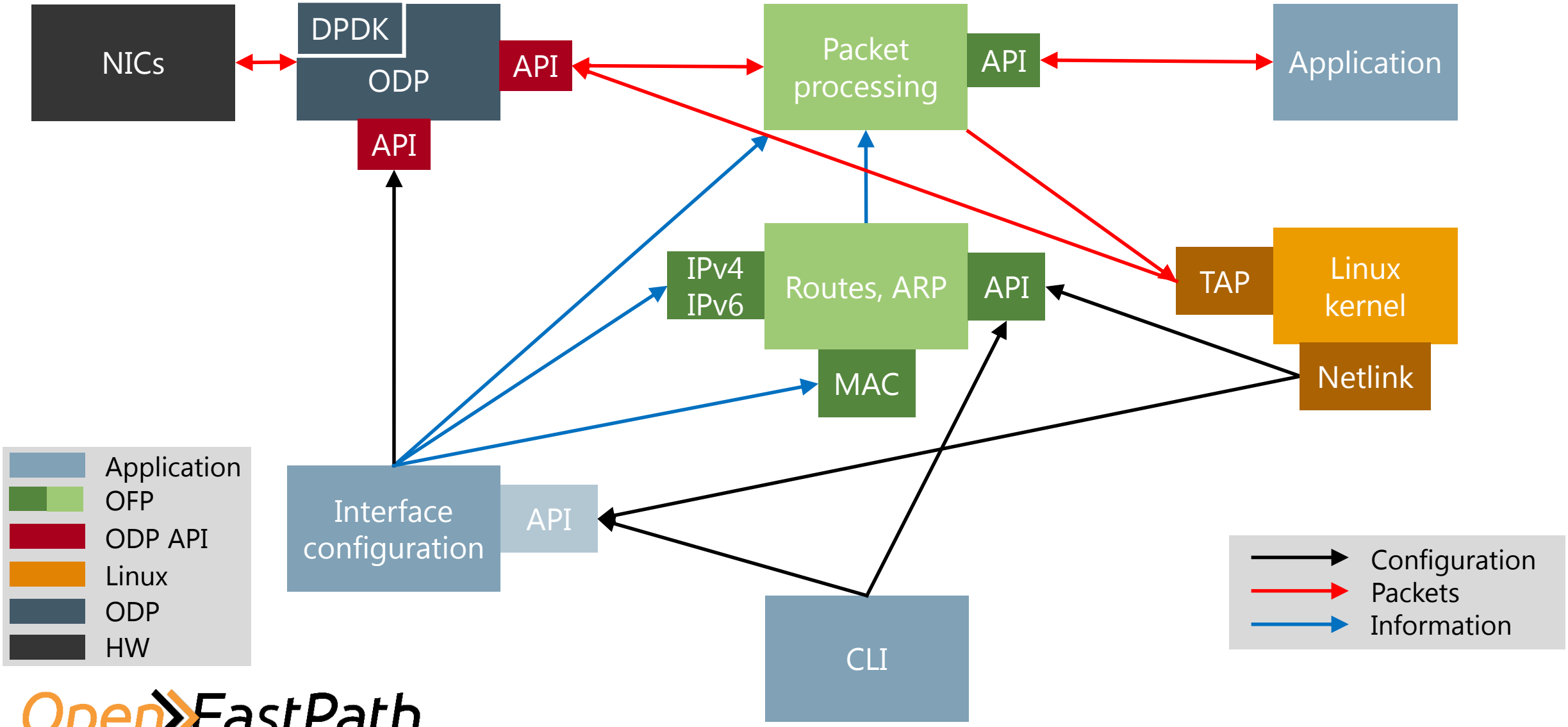
UDP, TCP, ICMP code was ported from libuinet (User space FreeBSD port)

- Non-blocking event based socket API
- Modular, multithreaded design focused on performance and scalability
- Tightly coupled to application, linked in as a library
- Maintainability – Tracks evolution of FreeBSD

High performance and scalable implementation for MAC and Route tables

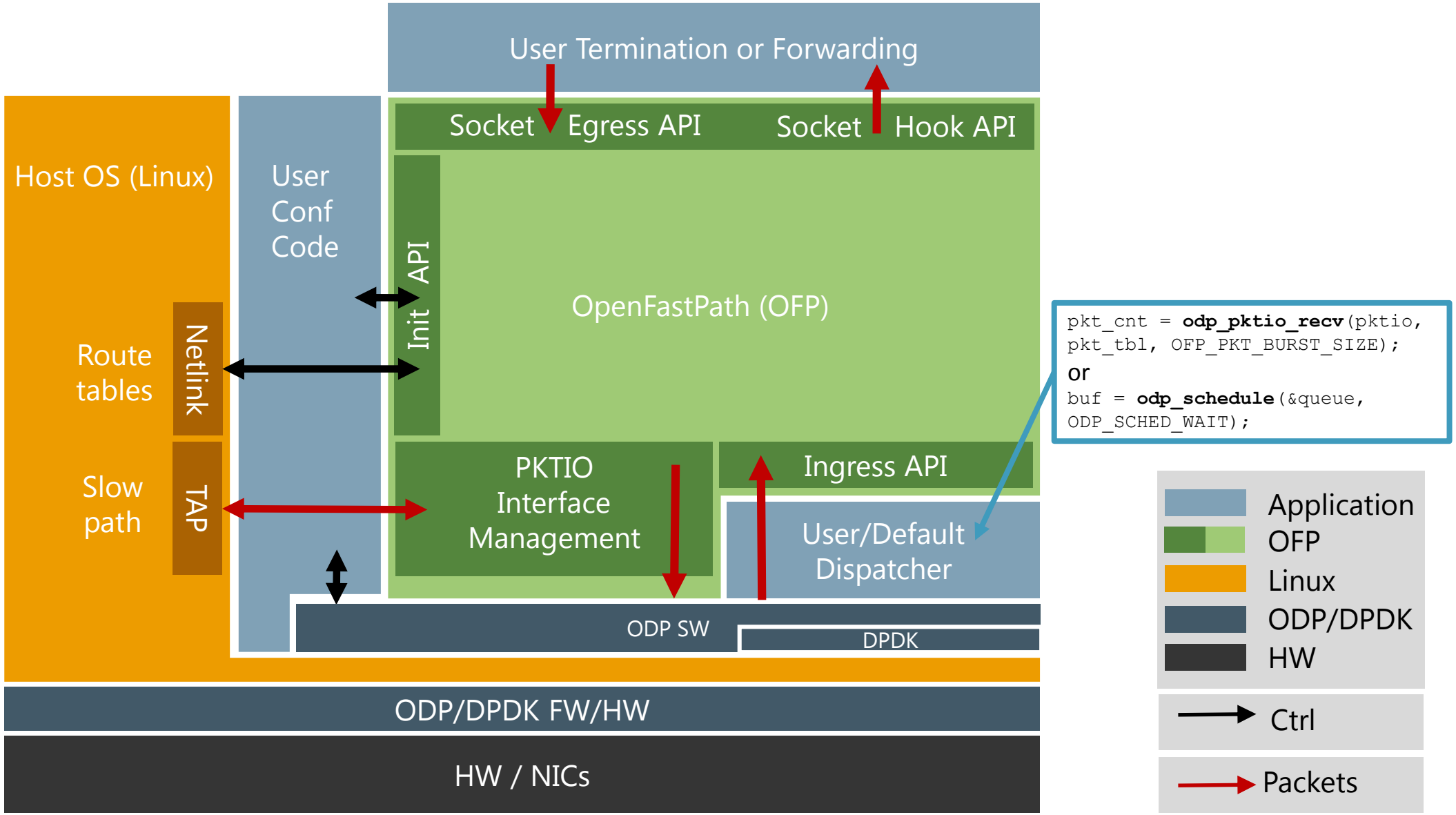
- Lockless synchronization

OpenFastPath system components

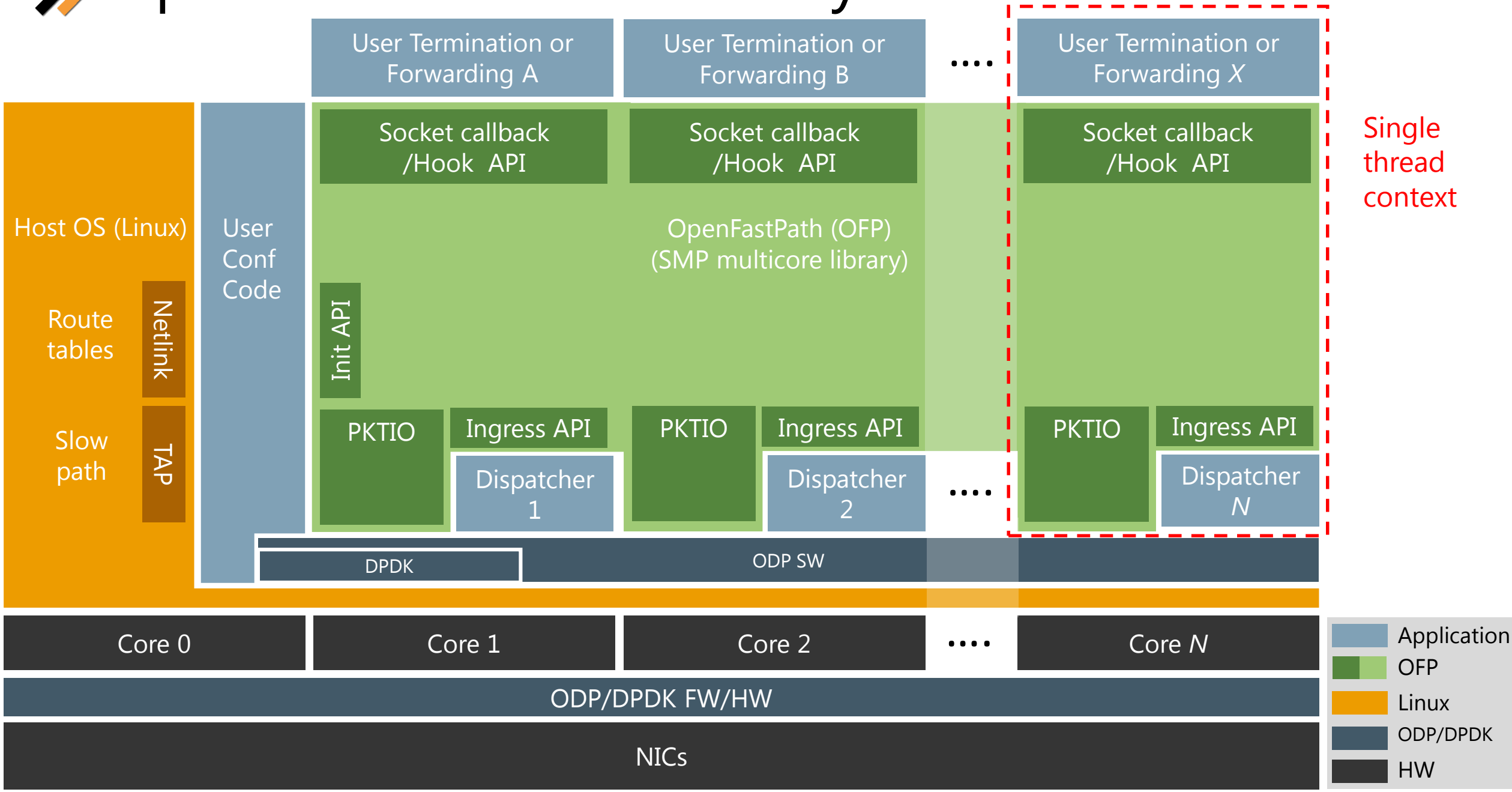


OpenFastPath

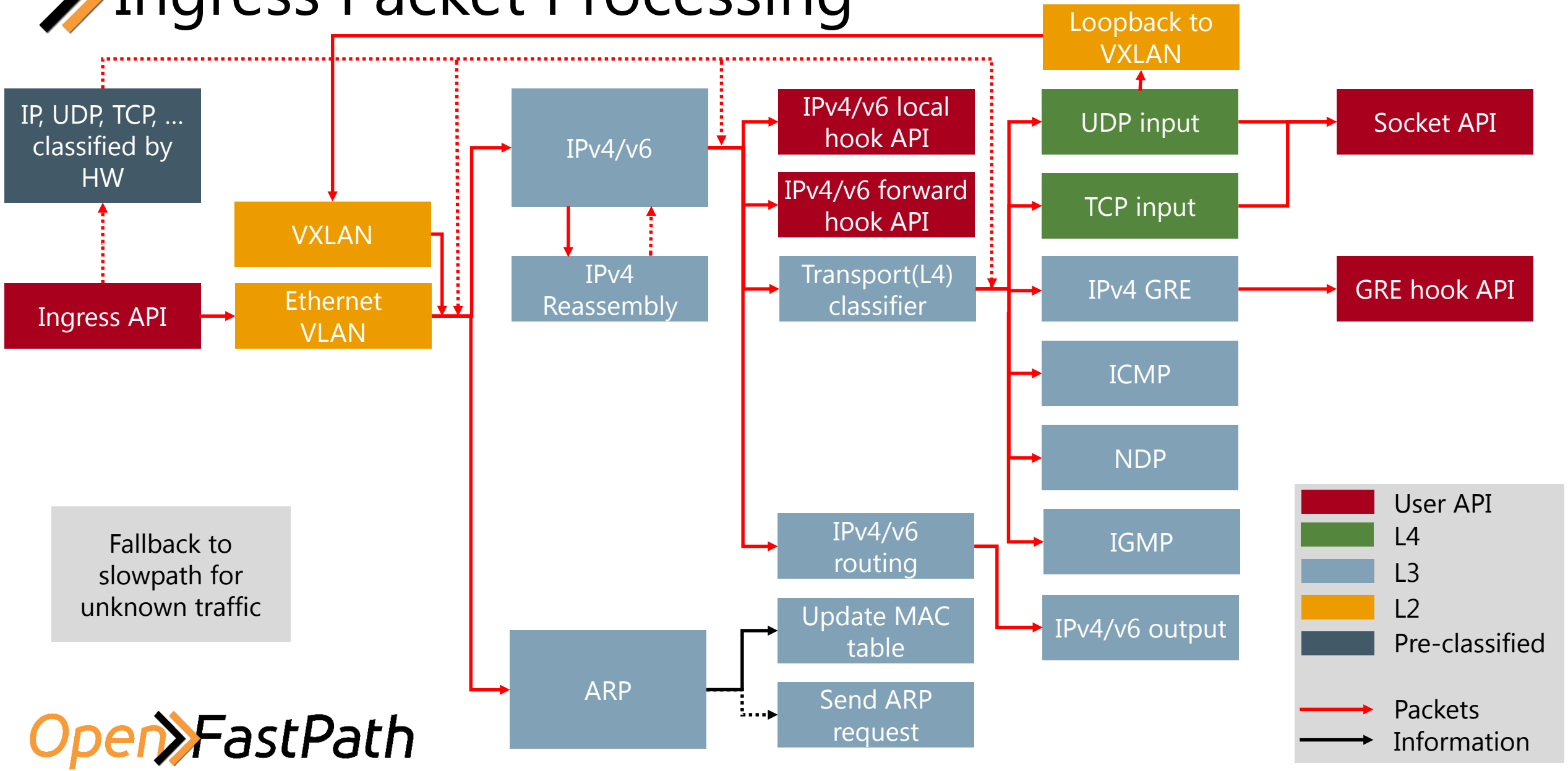
OpenFastPath System View



OpenFastPath multicore System View

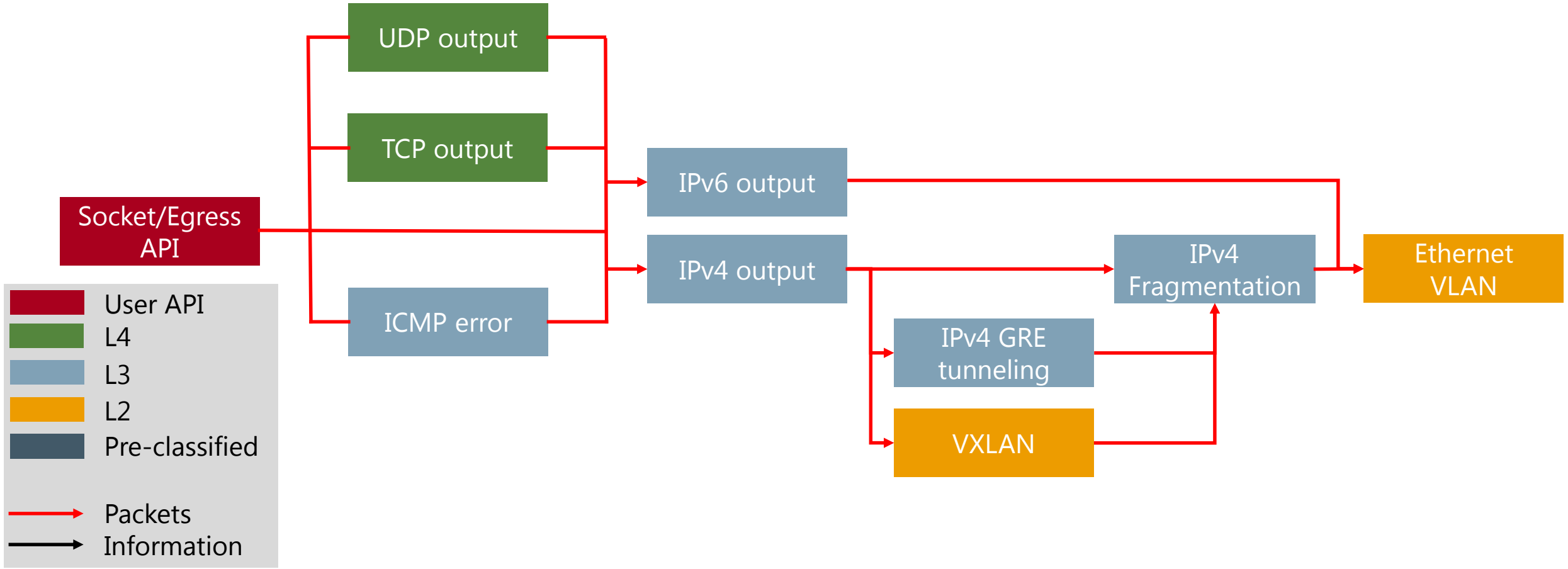


Ingress Packet Processing



OpenFastPath

Egress Packet Processing



OpenFastPath

Optimized OpenFastPath socket APIs

New zero-copy APIs optimized for single thread run-to-completion environments

- UDP
  - Send: Optimized send function with a packet container (packet + meta-data)
  - Receive: A function callback can be registered to read on a socket. Receives a packet container and socket handle
- TCP
  - Accept event: A function callback can be registered for TCP accept event. Receives socket handle.
  - Receive: A function callback can be registered to read on socket. Receives a packet container and a socket handle

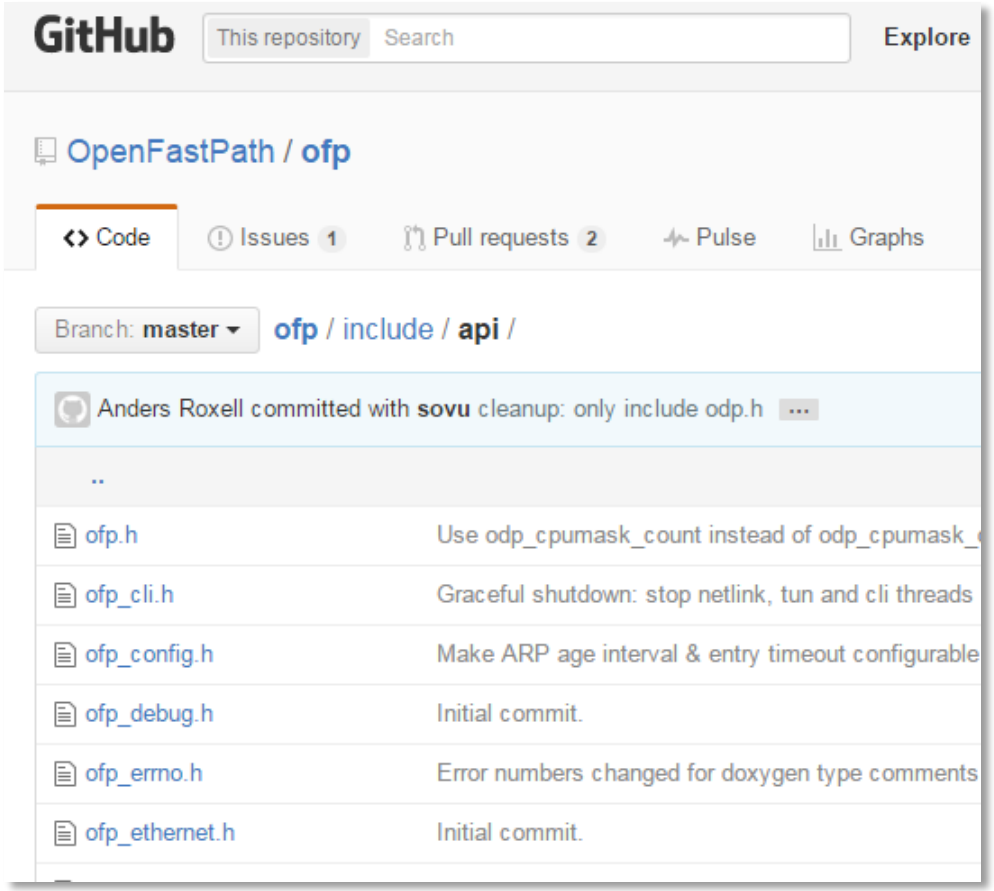
Standard BSD Socket interface

- For compatibility with legacy applications

OpenFastPath

Other OpenFastPath user application APIs

- Initiation of Open Fast Path
- Interface configuration
- Route and MAC table access
- Packet Ingress and Egress processing
- Hooks for IP local, IP forwarding and GRE
- Timer callbacks
- Statistics
- Packet capture



OpenFastPath

Code examples

```
76      /* PER CORE DISPATCHER */
77      while (1) {
78          event_cnt = odp_schedule_multi(&in_queue, ODP_SCHED_WAIT,
79                                         events, OFP_EVENT_BURST_SIZE);
80          for (event_idx = 0; event_idx < event_cnt; event_idx++) {
81              ev = events[event_idx];
82
83              if (ev == ODP_EVENT_INVALID)
84                  continue;
85
86              if (odp_event_type(ev) == ODP_EVENT_TIMEOUT) {
87                  ofp_timer_handle(ev);
88                  continue;
89              }
90
91              if (odp_event_type(ev) == ODP_EVENT_PACKET) {
92                  pkt = odp_packet_from_event(ev);
93  #if 0
100                 ofp_packet_input(pkt, in_queue, pkt_func);
101                 continue;
102             }
103
104             OFP_ERR("Unexpected event type: %u", odp_event_type(ev));
105
106             /* Free events by type */
107             if (odp_event_type(ev) == ODP_EVENT_BUFFER) {
108                 odp_buffer_free(odp_buffer_from_event(ev));
109                 continue;
110             }
111
112             if (odp_event_type(ev) == ODP_EVENT_CRYPTO_COMPL) {
113                 odp_crypto_compl_free(
114                     odp_crypto_compl_from_event(ev));
115                 continue;
116             }
117         }
118     }
119
```

```
181     /*
182      * Create and launch dataplane dispatcher worker threads to be placed
183      * according to the cpumask, thread_tbl will be populated with the
184      * created pthread IDs.
185      *
186      * In this case, all threads will run the default_event_dispatcher
187      * function with ofp_eth_vlan_processing as argument.
188      *
189      * If different dispatchers should run, or the same be run with differnt
190      * input arguments, the cpumask is used to control this.
191      */
192     memset(thread_tbl, 0, sizeof(thread_tbl));
193     ret_val = odph_linux_pthread_create(thread_tbl,
194                                         &cpumask,
195                                         default_event_dispatcher,
196                                         ofp_eth_vlan_processing);
197     if (ret_val != num_workers) {
198         OFP_ERR("Error: Failed to create worker threads, " \
199             "expected %d, got %d\n",
200             num_workers, ret_val);
201         odp_term_global();
202         return EXIT_FAILURE;
203     }
```

ODP thread creation above

OFP default dispatcher to the left

Why should someone use OpenFastPath?

Portable high performance solution supporting multiple HW platforms

- Functionality verified on ARM, MIPS and x86 HW

Highly optimized and scalable solution

- Non-blocking event based API focused on performance and scalability

User space implementation

- Simplifies maintenance and maximizes throughput and scalability by minimizing Linux kernel dependency

Very flexible deployment scenarios

- Embedded, virtualized, servers, edge nodes, etc.

OpenFastPath

Why engage in the OpenFastPath project?

OpenFastPath is designed as an open source project from the start

- Based on known open source code like libuinet
- Not an old proprietary code base turned open source….

The framework is highly modular, adaptable and lightweight.

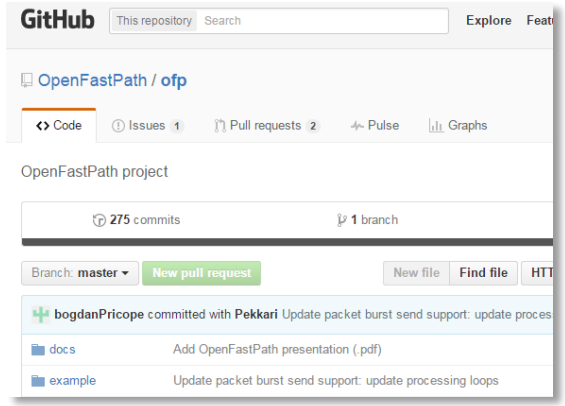
- Not restricted to plug-ins

Membership is cheap and open for all

- Potential to impact is high

Very high interest from major industry players

What's next? - Get involved!



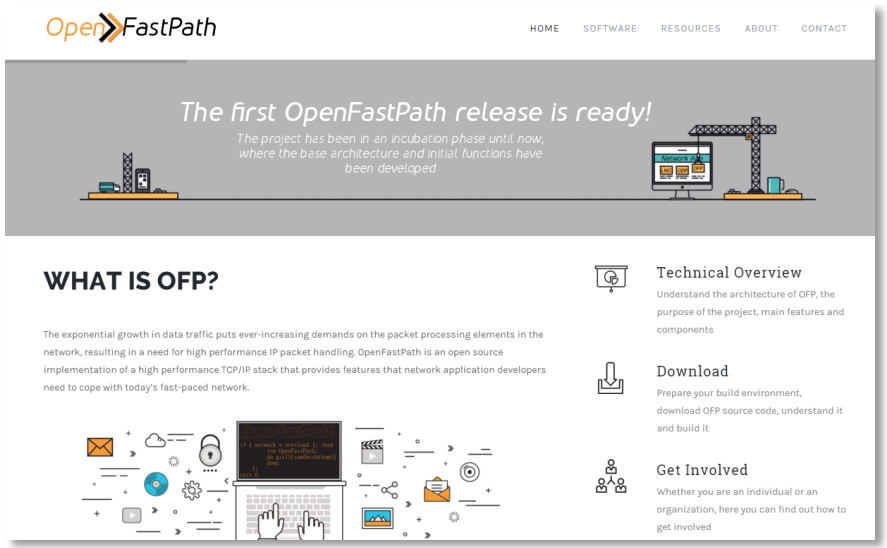
Download the source code from: *https://github.com/OpenFastPath/ofp*

Check us out at *www.openfastpath.org* **to get more information about the project**

Subscribe to Mailing-list: *http://www.openfastpath.org/mailman/listinfo*

Ping us on our freenode chat: *#OpenFastPath*

Membership is cheap and open to all!



Enea services offering on OFP

Integration services

- Integration of OFP in customer hardware and software system.

Hardware porting and optimization services

- Test, verification and optimization of silicon vendor ODP implementation together with OFP

Feature development services

- Pre-studying, specifying and implementing new OFP features and protocols.

Production test, maintenance and support services

- Production testing, release management and support.

Thank You

For additional information, please visit
www.openfastpath.org

OpenFastPath