



OpenOnload

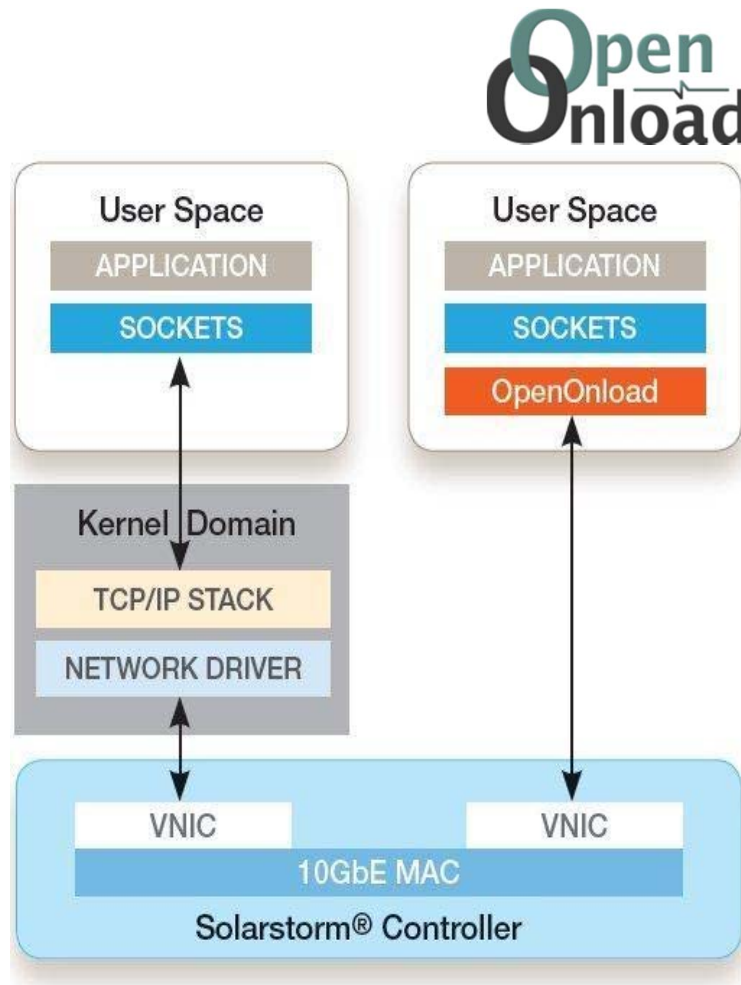
Dave Parry – VP of Engineering

Steve Pope – CTO

Dave Riddoch – Chief Software Architect



OpenOnload® Application Acceleration Software

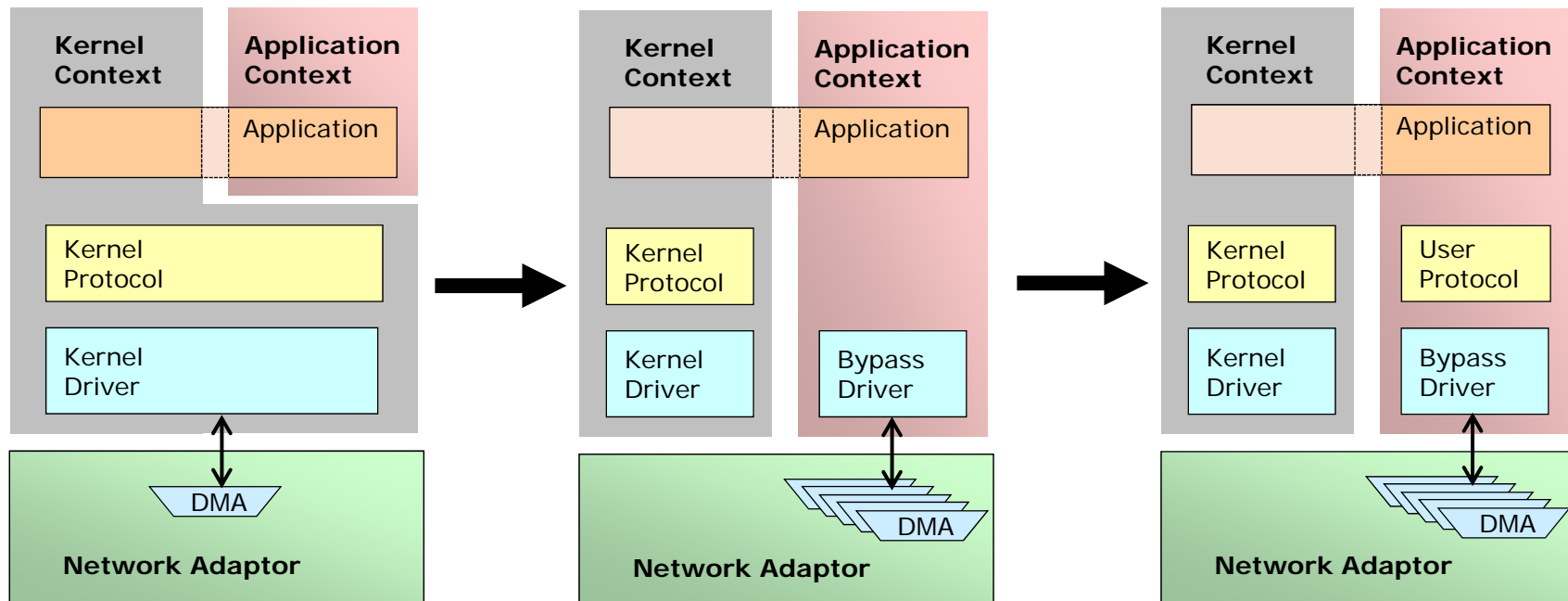


- Accelerated performance
 - TCP/IP, UDP and multicast acceleration
 - Streamlines and reduces interrupts, context switches and data copies
 - Reduces latency by 50%, increases message rates 3x or more
- Seamlessly integrates into existing infrastructure
 - Binary compatible with industry standard APIs
 - No software modifications are needed
 - Standards-based solution uses TCP/IP and UDP
 - No specialized protocols needed
 - Compatible with existing Ethernet infrastructure
- Open source, available with bundled support

OS Bypass Architectures

Offer great performance because network access is made in the context of the application process

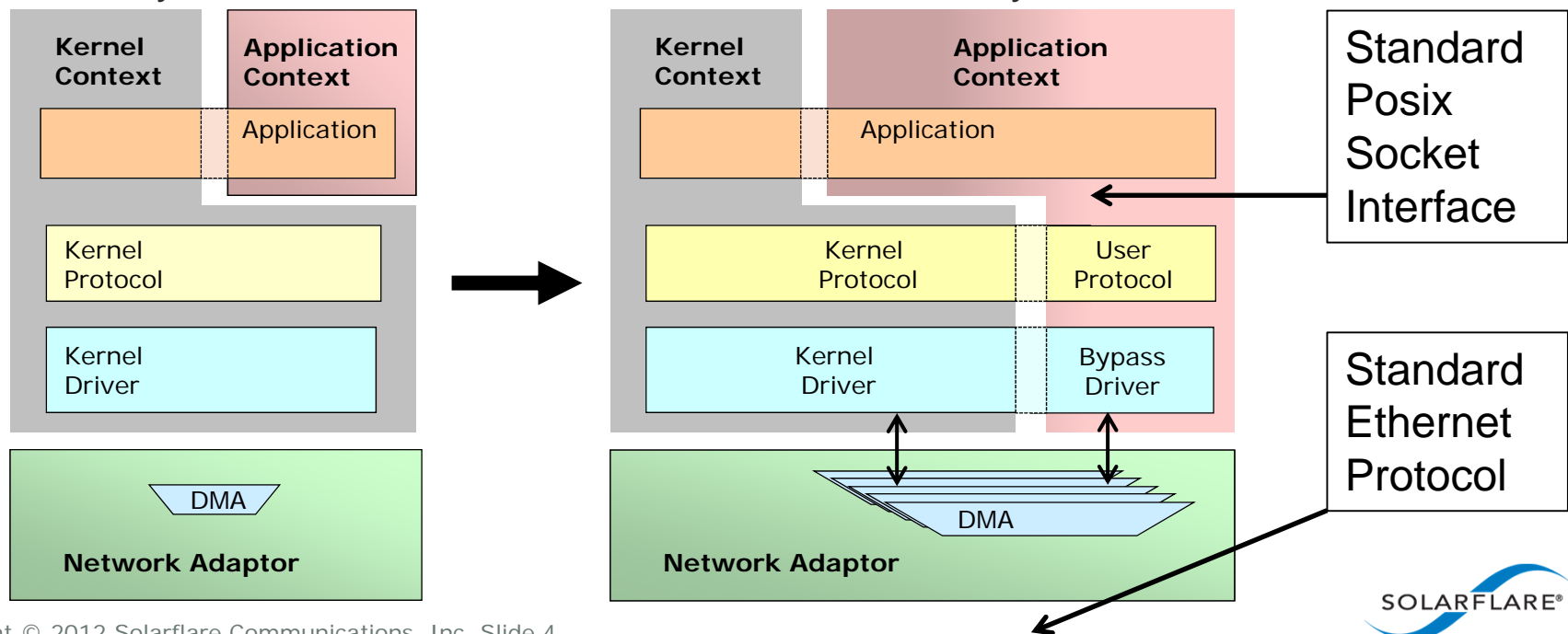
- Reduces overheads, but requires user-safe hardware and special libraries
- Increased performance and parallelism, but at the cost of specialized APIs requiring application changes or specialized middleware
- Application compatibility achievable via standard APIs (eg. BSD sockets) , but at the cost of API completeness and versatility



OpenOnload

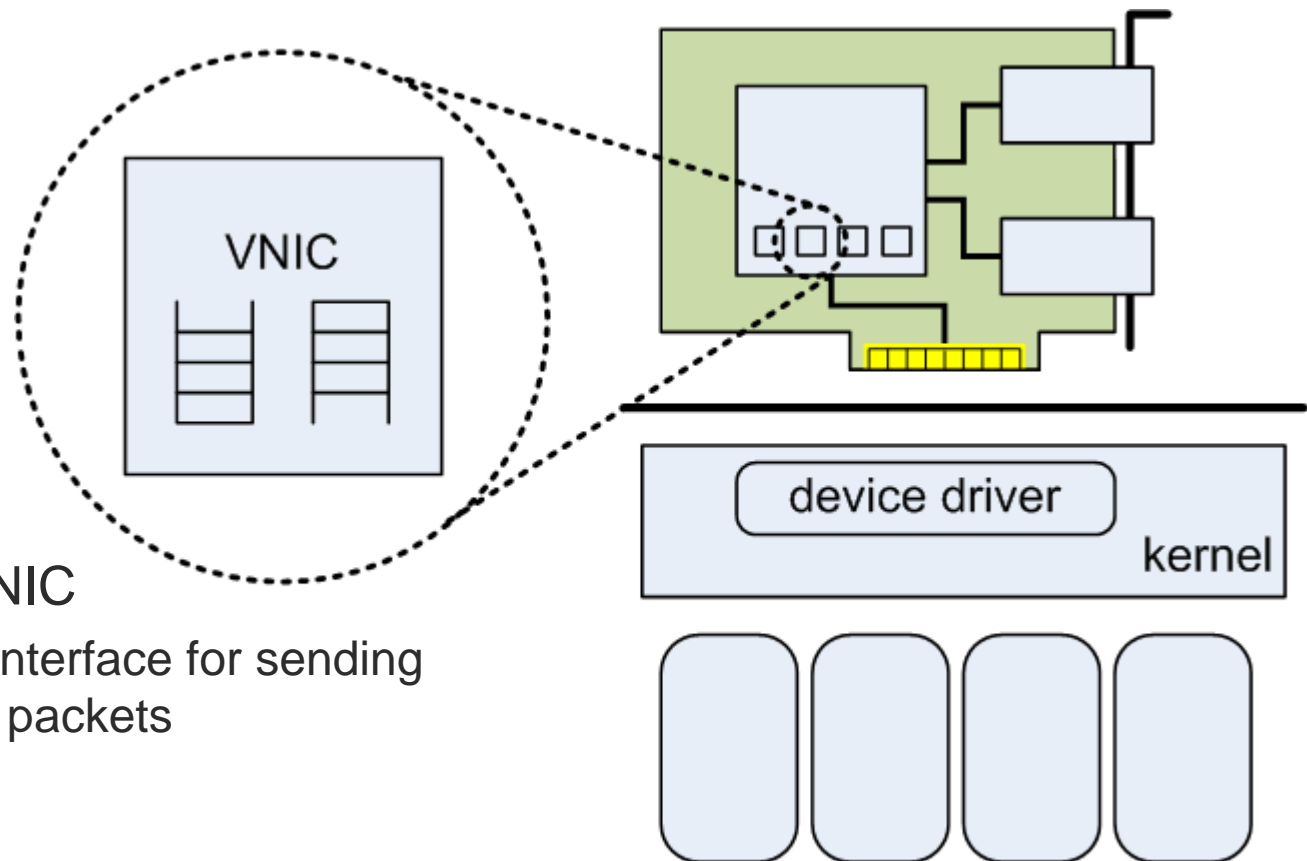
Provides the benefits of a bypass architecture without the sacrifices or compromises, by implementing a complete **hybrid model**

- Supports both high-performance, and fully conformant operation in congested environments
- Supports mixed operation of accelerated and non-accelerated interfaces
- Protected, shared state allows sharing of sockets between processes (eg. via fork()), and supports persistence of state through process creation.
- Provides compliance *and completeness* via rich implementation backed by ability to fall-back to kernel stack when necessary

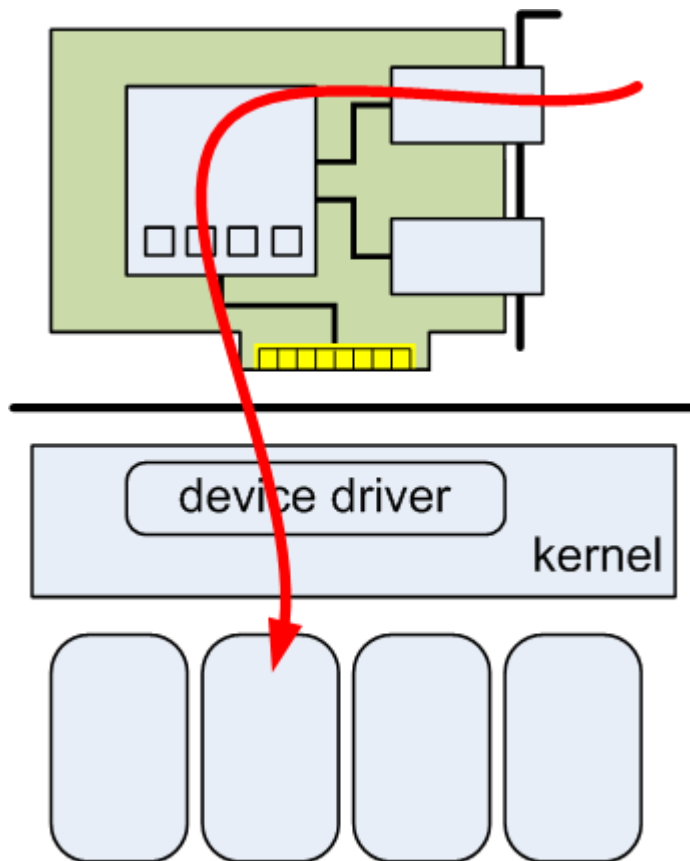


How Does the NIC Help?

- Low latency cut-through design
- 1024 VNICs per port
- VNIC == Virtual NIC
 - Independent interface for sending and receiving packets
- Flow steering
 - Direct individual flows to specific VNICs
 - Supports scaling and NUMA locality



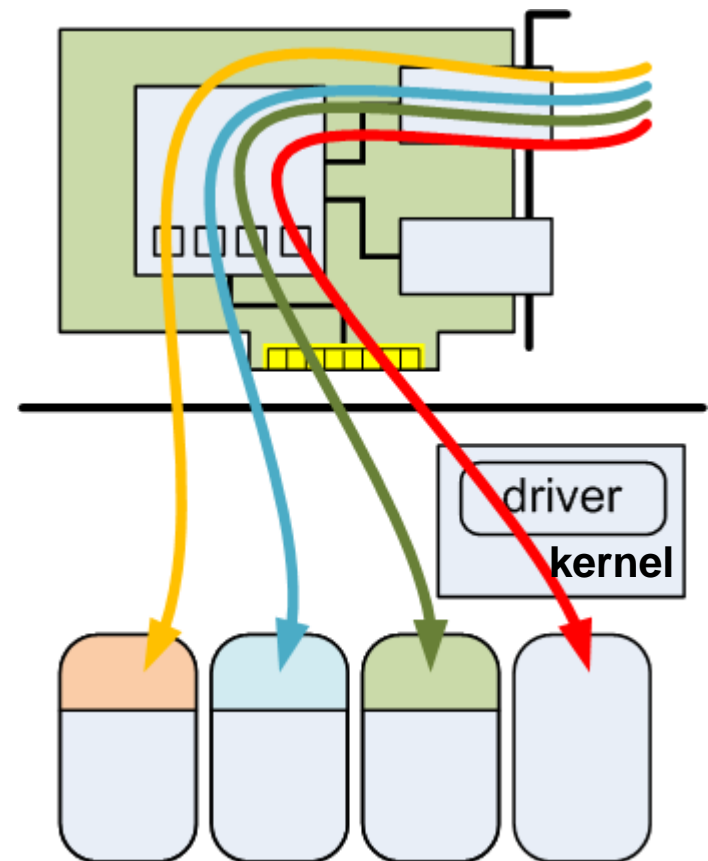
Kernel Networking



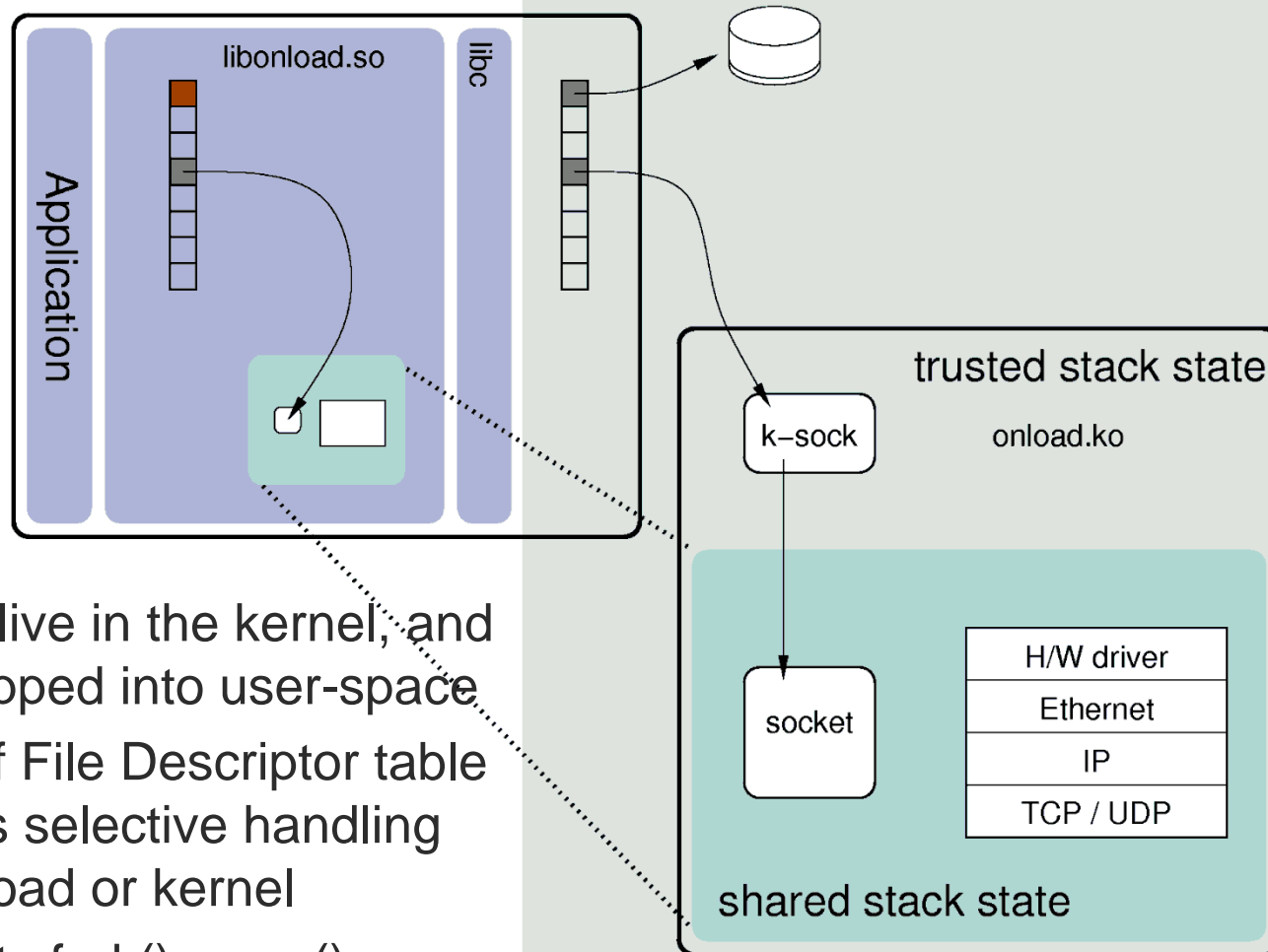
- Traditionally the network stack executes in the OS kernel
- Received packets are processed in response to interrupts
- Applications invoke the network via the BSD sockets interface by making system calls

Kernel Bypass – OpenOnload

- Dedicate a VNIC per application or thread
- TCP/UDP stack as user-level library
- Critical path entirely at user-level
- Reduces per-message CPU time
 - Cuts latency in half
 - Increases message rate by 5x per core
 - Improves scaling
- Fully compatible – no changes to applications needed



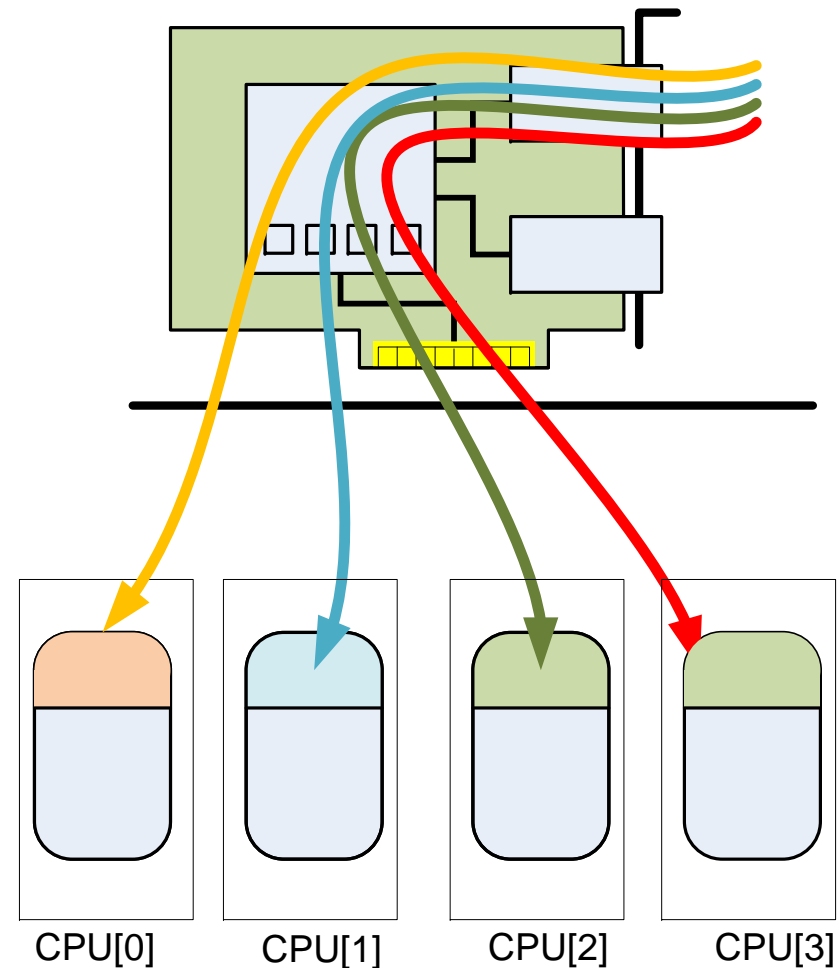
OpenOnload Architecture



- Stacks live in the kernel, and are mapped into user-space
- Copy of File Descriptor table enables selective handling via Onload or kernel
- Supports `fork()`, `exec()`, ...

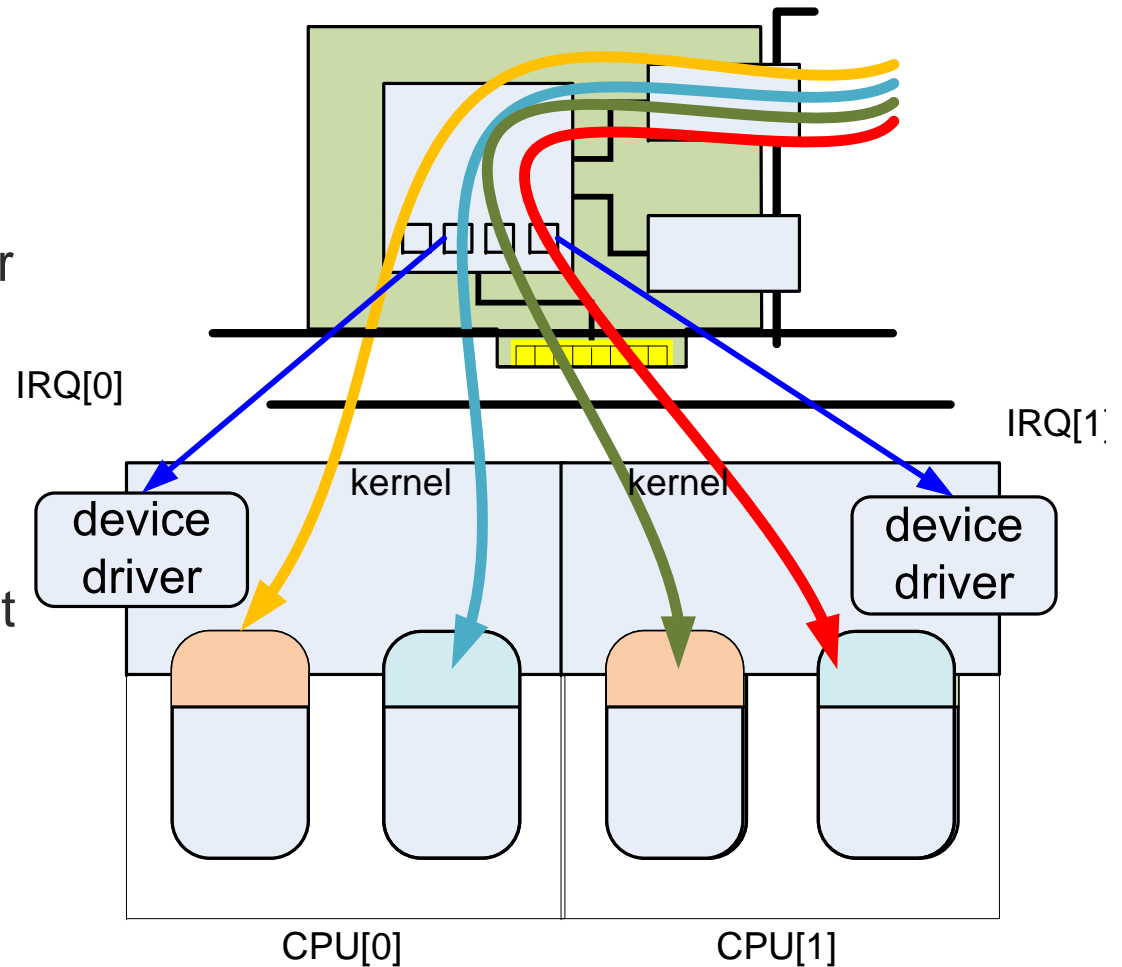
OpenOnload – CPU Core for Each Thread

- Sufficient CPU cores to dedicate one to each thread / process
- Configure Onload to spin waiting for network events rather than block in system calls such as `recv()`, `poll()`, `epoll()`
- Result is that no interrupts are required or generated, latency and jitter are minimised
- Not suitable for applications where there are more threads than CPU cores



OpenOnload – with Interrupts

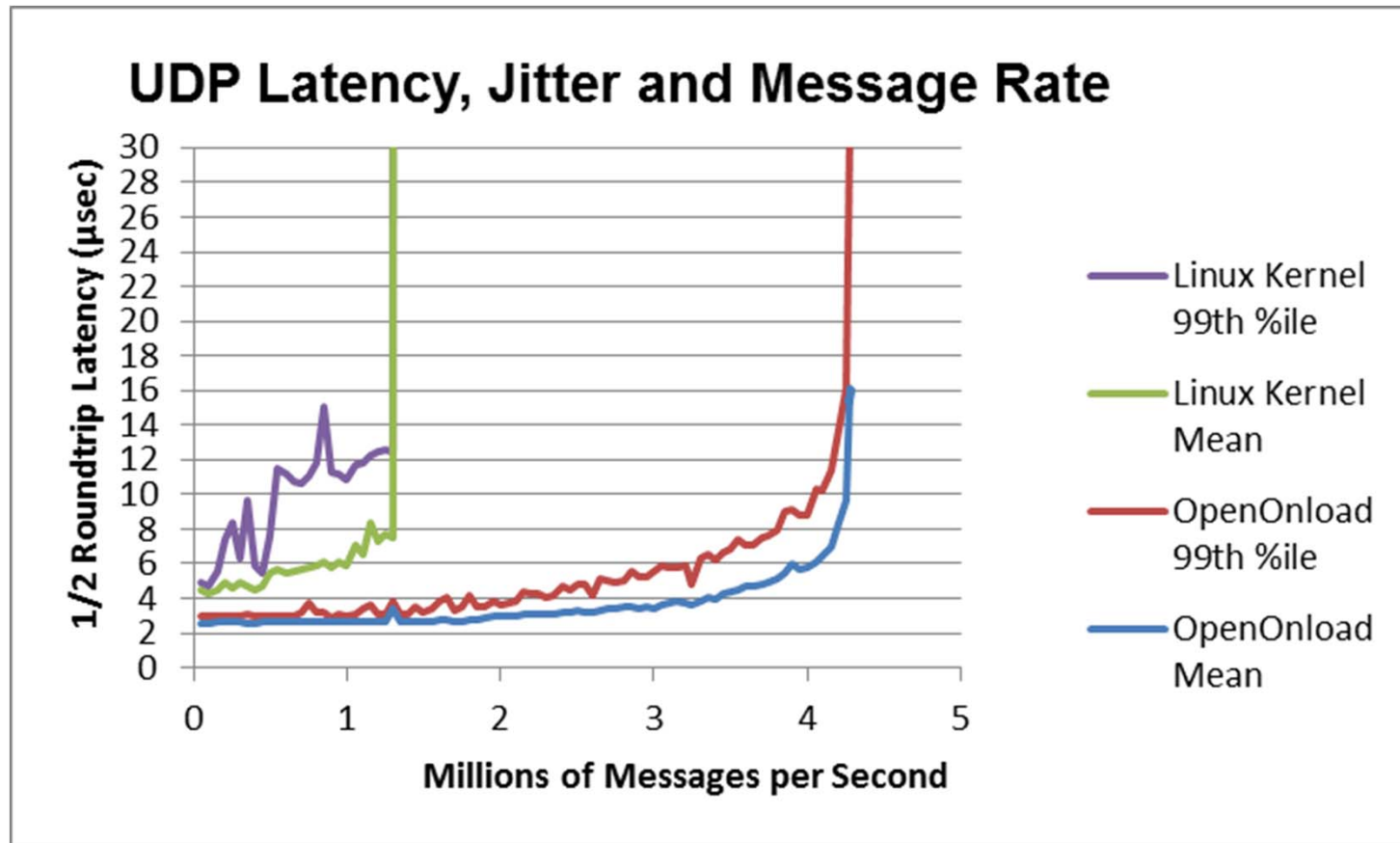
- If multiple Onload threads / processes per CPU core
- Configure Onload to block or spin-block
- Interrupts generated on per-application (onload stack basis). Can be affinitized just like the application thread
- TCP state shared between kernel and user mode. Enables protocol to be executed by kernel on behalf on a de-scheduled or unresponsive thread



OpenOnload – to Spin or Not to Spin? - Hybrid

- Onload spinning without interrupts for lowest latency / jitter in configurations where CPU resources are not contended
- Onload with interrupts enables threads to block and be scheduled
- In both cases, protocol processing often achieved in the context of the application with reduced overhead
- In both cases, TCP processing can take place in either user-mode or kernel mode
- This hybrid architecture also necessary to support the full POSIX API – fork(), exec() and also debugging / diagnostics

The Result...Performance



- SFN6x22F the choice for low-latency / highest message rates with ultra-low jitter
- Also, Maximum multi-stream packet rate of **20M pps**

Tests run using sfnt-stream (single stream data) / openonload-201109-u1
Intel i5-2500K CPU @ 3.30GHz, 8GB mem

Conclusions

OpenOnload's hybrid kernel/user architecture provides for

- High performance with application compatibility
- Full sockets API compliance via mixed OpenOnload/Kernel processing and unique shared-stack model
- Arbitrary mixing of accelerated and non-accelerated interfaces
- Efficiency via steering and affinitization improving locality
- Hybrid busy-wait/interrupt model enables timely processing in user or kernel mode as appropriate for optimal performance and efficiency
- Single or dual-ended usage – fully standard at the application interface, fully standard on the wire