Networking Workloads on Intel Architecture

Communications, Storage and Infrastructure Group
September 2014

Legal Disclaimer

What does the Intel DPDK team do?

Implement new features/libraries to aid networking workloads

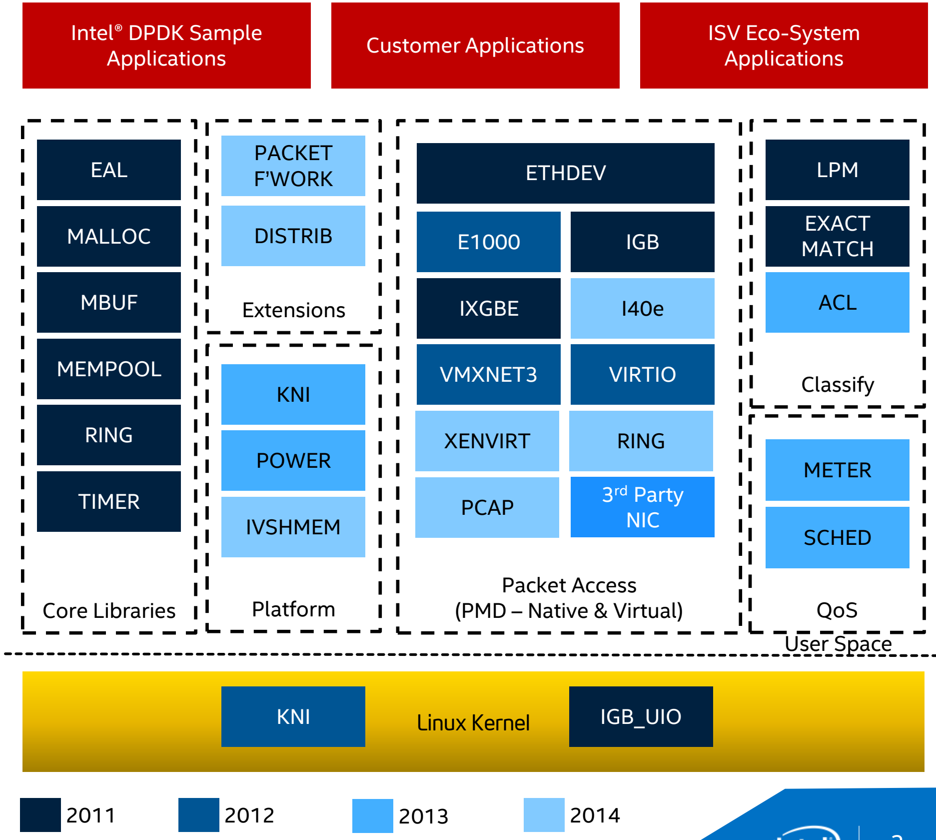
- Ongoing process as we discover bottlenecks/problems
- Customer feedback

Continuous performance improvement of existing libraries

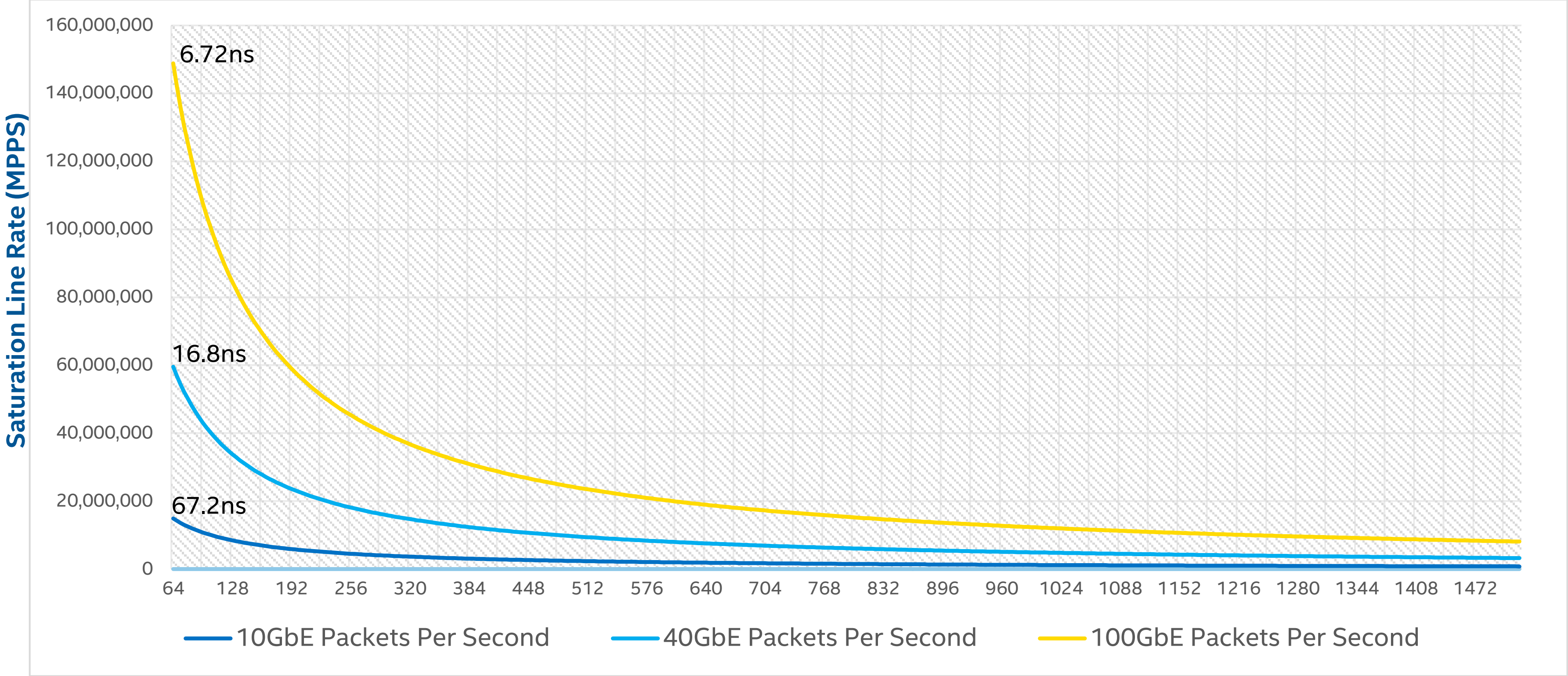
- Pure code optimizations on existing platforms
- New/improved algorithmic implementations
- With multiple CPU architectures (Intel® Xeon®, Atom)

Recommend & drive enhancements

- To further packet processing solutions on Intel Architecture platforms



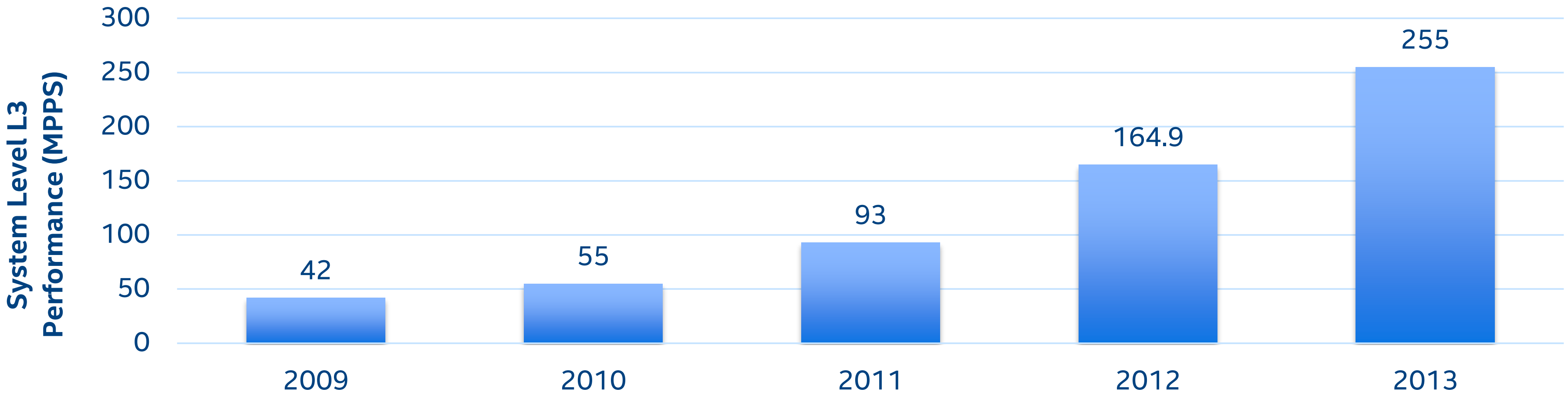
The Challenge



Intel® DPDK Performance

A snapshot of on different architectures

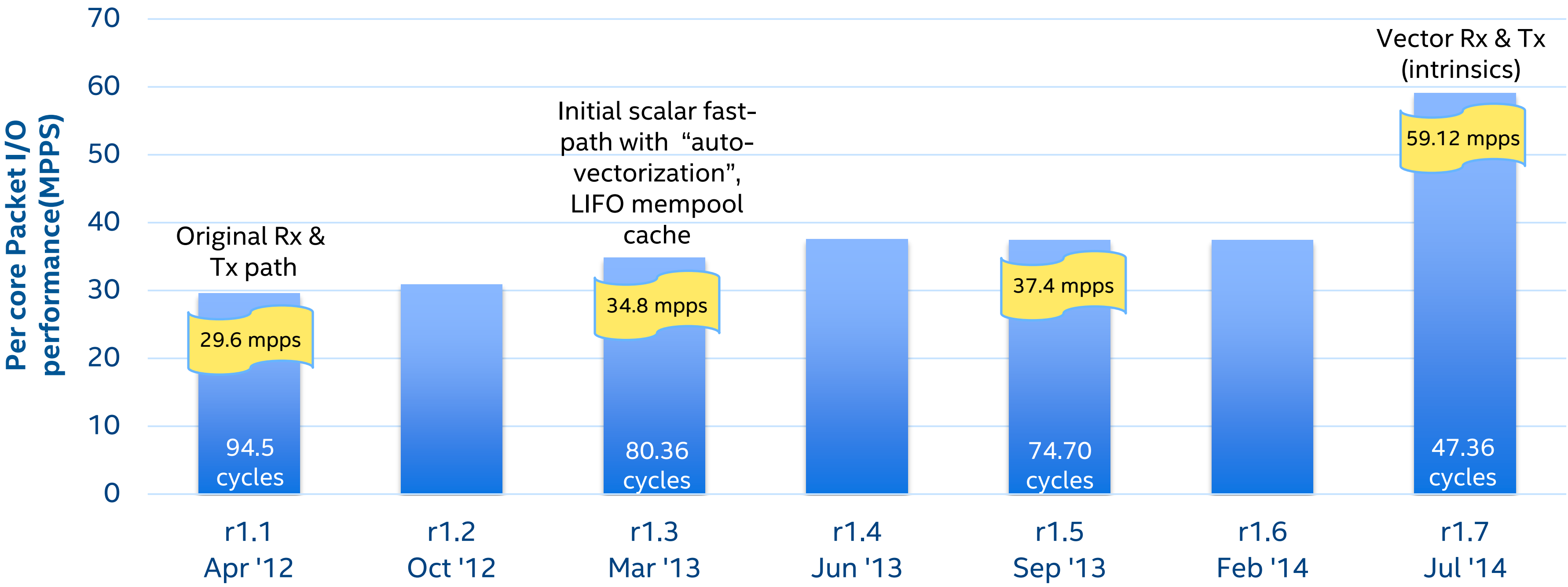
Platform Features			
	Integrated Memory Controller PCI-E Gen2	Data Direct I/O Integrated PCI-E Gen3 AVX1 (integer)	4x10 GbE NICs



Disclaimer: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Per core I/O performance …

Performance of DPDK releases on E5-2680v2 (2.80 GHz)



Disclaimer: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Changes under consideration …

Improving performance further …

- Patches to rework the original 1.1 “slow” path with a faster version
- Re-organized mbuf to carry more metadata from NIC
- Investigating implementation using AVX2
- Latency …

Performance improvements to the exact match library (rte_hash)

- Faster hash functions
- Higher flow count (16M, 32M flows)
- Characterize/limit memory bandwidth usage
- Different algorithmic implementation – cuckoo hash

DPDK and latency ….

L3fwd default tuning is for performance

- Coalesces packets up to 100us
- Receives and transmits at least 32 packets at a time
  - nb_rx = rte_eth_rx_burst(portid, queueid, pkts_burst, **MAX_PKT_BURST**);

Could bunch 8, 4, 2, or even 1 packet(s) and trade-off some performance

- Lower coalescing increases transmit cost per packet
- Even at ~200 cycles per packet on a CPU running at > 2 GHz software isn't a big adder to latency

Larger system-wide investigations

API versioning

Dynamic management of DPDK resources

- CPU/Threads, memory, network

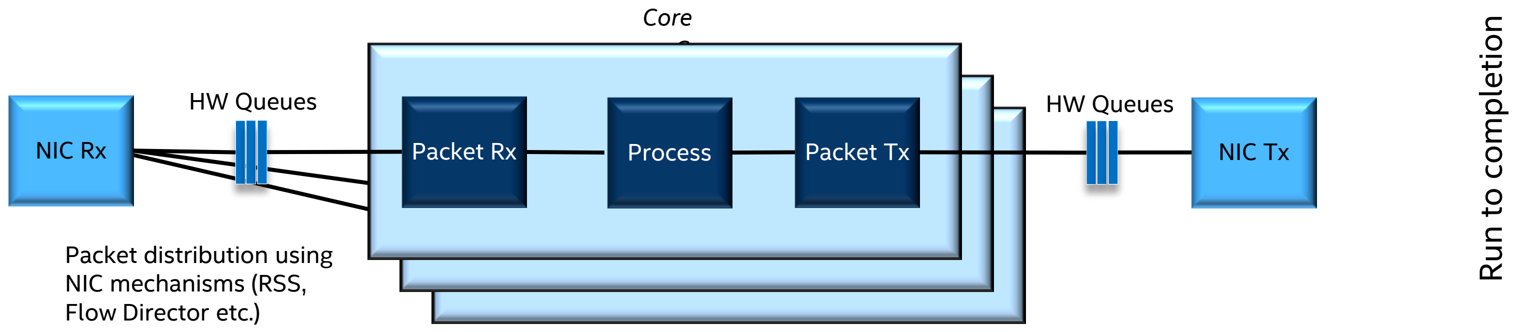
Sharing DPDK core with other pthreads

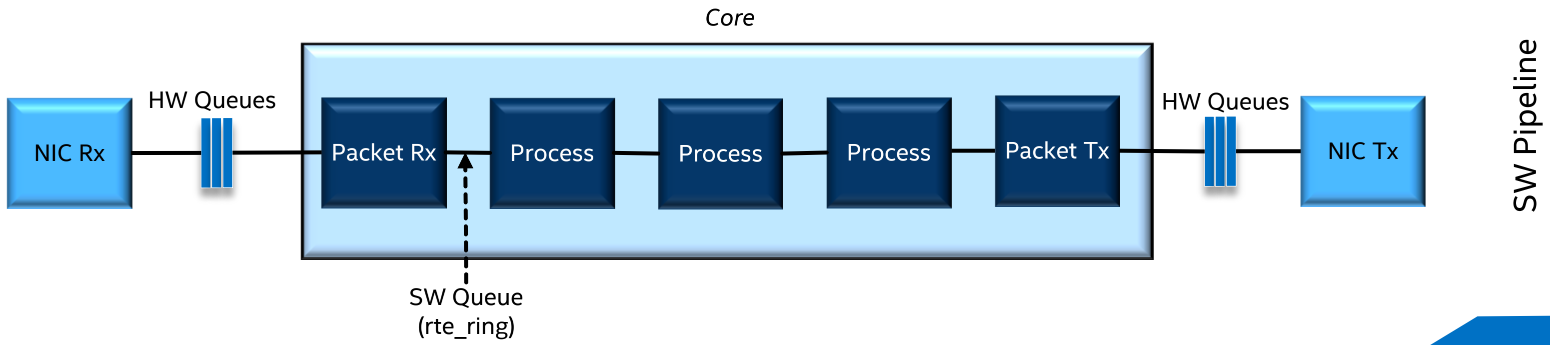
- Adding interrupt-based entry, cgroups controlled time-sharing

Sharing NIC port between DPDK and kernel

- VFIO, bifurcated driver

Building applications on top of DPDK





Run-to-completion or Pipeline?

DPDK doesn’t impose a model – both are supported, as are hybrid approaches

- Each has their advantages and disadvantages
- Have tools/benchmarks to evaluate either approach

Direction should be dictated by

- Legacy code & TTM considerations
- Performance requirements
- NIC limitations (e.g. RSS is purely IP packets only)

Assessing basic performance

app/test: Implements unit-tests and micro-benchmarks

- Micro-benchmarks exist for a number of libraries – labeled *_perf_autotest – mempool, hash, ring, timer, memcpy, distributor *[adding nic in 1.8?]*

app/testpmd: NIC I/O benchmark

- Benchmarks the network I/O pipe (NIC hardware + PMD)

examples/l3fwd: An example L3 forwarder

- Increased CPU processing – NIC hardware + PMD + hash/lpm

examples/load_balance: a simple load balancer in software

examples/ip_pipeline: Using the packet framework to build a pipeline

Intel uses these micro-benchmarks to drive performance

Summary

Multiple areas of improvement in the pipeline

- Focus is on performance, functionality, usability

Looking for community input/participation

- What are the problems that we need to solve?

Let's discuss …

Backup

The libraries/components (1)

Library	
librte_eal	Environment Abstraction Layer. Meant to hide system/OS specifics from “common” upper layers
librte_malloc	rte_malloc() - replacement for malloc(). Allows allocation of data structures backed by huge pages
librte_mempool librte_mbuf	Memory management: DPDK buffer pool management and packet buffer implementations
librte_ring	High speed ring for inter-core/process pointer passing
librte_timer	Timer routines
librte_lpm	Accelerated longest prefix match
librte_hash	Hash driven key-value exact match for tuple matching
librte_acl	Accelerated implementation of an Access Control List

The libraries/components (2)

Library	
librte_meter	Meter/mark library: Implements srTCM (RFC 2697) and trTCM RFC 2698)
librte_sched	Hierarchical traffic shaper in software
librte_pmd*	Packet Access “Poll” mode drivers
librte_ether	Generic Ethernet device abstraction – the DPDK PMD API
librte_cmdline	Command line parser library
librte_distributor	A work queue distributor
librte_power	Power management primitives
librte_ivshmem	Shared memory implementation for inter-VM communication
KNI, librte_kni	Kernel Network Interface – implements a kernel netdev for passing packets into the kernel from DPDK