OPENFLOW SOFTWARE SWITCH & INTEL® DPDK PERFORMANCE ANALYSIS
AGENDA

› Background
  – Intel® DPDK
  – OpenFlow 1.3 implementation sketch

› Prototype design and setup

› Results

› Future work, optimization ideas

- kernel space implementation
  - is more restricted
  - harder to develop and debug
  - interrupts are still needed → performance issues
- user space implementation over normal Linux kernel is slow
  - user – kernel memory separation, copy is slow
    - some workarounds exist (e.g. pcap mmap), but they are still not fast enough
  - a similar, but less widespread solution: [http://info.i et.unipi.it/~luigi/netmap/](http://info.i et.unipi.it/~luigi/netmap/)

Main features

- poll mode driver: avoid using interrupts and scheduling
- direct I/O: packet – or first X bytes – is copied to L1 cache directly
- some details from the Intel® DPDK tutorial will follow
Intel® DPDK Basic Design

Designed to run on any Intel® architecture CPU
- Intel® Atom™ to client cores to Sandy Bridge
- Essential to the IA value proposition

PThread to bind h/w thread to s/w task
- Literally no scheduler overhead

User Level Polled Mode Driver
- No Kernel Context / Interrupt Context Switching Overhead

Huge Pages To Improve Performance
- 1Gig Huge as well as 2 Meg Page support
- Co-exists with Linux’s 4 K Page

Low Latency Cache and Memory Access
- DDIO - Cache
- Prefetch and rte_cacheAligned - memory
Understanding the Choices & Performance
Setting the Direction for the Intel® DPDK

Scheduler (or why not)
• Hardware threads only
• No scheduler/task switcher - typical task switch time is between 200+ processor cycles (varies depending on processor architecture)

Process bunch of packets at a time
• Cores process a bunch of packets at a time to amortize some latencies

Prefetch
• Critical to latency hiding since we don’t have software threads. Stalls on hardware threads are costly
• The queue-based model is key to making prefetch effective

Locks
• Generally lockless implementations where-ever possible. A spinlock-unlock pair costs between 60-90 cycles.
• Queues are lockless (single producer & multi-producer, single consumer)
Scheduler ... or Why not?

Primary reason was performance:
- Task-switch overhead is typically a “few” hundred cycles
- FXSAVE/FXRSTOR are 100 and 150 cycles respectively (on Intel NetBurst®)
- Faster on recent processors, but not significantly
- Need to add cost of interrupt if pre-emptive

To put that in perspective in a 10 GbE environment
- On a 3 GHz processor, for small (64B) packets, a packet arrives every 67.2 ns = 201 cycles

For lower bandwidth environments, an essential thing to think about is the added CPU bandwidth consumed ...
Packet Bunching ...

Done on the NIC today

- NIC Receive descriptors are bunched four to a cache line
- Writing back “partial” descriptors has a severe performance penalty
  - Conflicts between CPU and I/O device on the same cache line
  - Increases memory & PCI-E bandwidth usage
  - Needed to overcome PCI-Express latencies

- All Intel Ethernet* controllers have settings that can be tweaked to control descriptor write-back
  - Coalesce as many descriptors as possible on Receive
  - Transmit side coalescing done as well (software controlled)
  - Timer values can be set to control latency (EITR)

Took the paradigm to the next level in having the fast-path process bunches of packets
- Facilitated by the queue abstraction

* Other names and brands may be claimed as the property of others.
Prefetch

Two types of prefetch – hardware & software

• Hardware prefetch is issued by the core
  – L1 DCU prefetcher: Streaming prefetcher – triggered by ascending access to recently loaded data
  – L1 IP-based strided prefetcher: triggered on individual loads with a stride
  – L2 DPL: Prefetches data into L2 cache based on DCU requests
    – Adjacent cache line (n, n+2, prefetch n+2)
    – Strided prefetcher (e.g. skipped cache lines)

• Software prefetch needs to be issued “appropriately” ahead of time to be effective
  – Too early could cause eviction before use
  – Multiple types of software prefetch
Paging

• 1GB super-pages & 2 Meg Huge Page Support

Performance implications ...

• Primarily due to D-TLB thrashing/replacement
• Paging performance drop is difficult to gauge ... really dependent on application
• Gets significantly worse as memory footprint increases –
• Varies by architecture, but initial measurements suggested ~30% on L3 forwarding
  – Quite often 2-3 D-TLB replacements per packet
Intel® Data Direct I/O Technology (Intel® DDIO)

L3 Forwarding Throughput

Intel® Data Direct I/O Technology

Packet Throughput

Configuration:
- L3 Forwarding Benchmark
- DPCD get-20110211
- Rose City CRB, 8x4GB DDR3-1333MHz
- 2xSNB-EP 8C B0, 2.0GHz
- Linux 2.6.33.6
- 4x X520-SR2 Dual 10GbE Ethernet NICs
- ECG Labs, February 2011

1x SNB-EP 8C B0, 2.0GHz

Intel® Data Direct I/O Technology demonstrates near linear performance
**Intel® DPDK Performance IPv4 Layer 3 Forwarding on an IA Server Platform**

![Graph showing performance improvements over time](graph)

**Massive IA Performance Improvements since 2009, PCIe Gen3 will Offer Even Better Performance....!**

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.
OPENFLOW BASICS

› Main idea: programmable networking
  – flexibility, programmability – together with high performance

› Problem: OF is either flexible OR fast today
  – flexible rules with many tuples: use TCAM or slow lookup
    › TCAM is expensive and uses a lot of power
  – complex instructions and actions: high overhead for software implementations
  – some solutions limit flexibility to increase performance (e.g. TTP)

› In theory performance should only depend on the data plane functions the node is implementing in the given scenario
  – it should be irrelevant whether the device is executing a native implementation of the use case, or is executing OF rules programmed by a controller for the same purpose
AGENDA

› Background
  - OpenFlow 1.3 implementation sketch
  - Intel DPDK

› Prototype design and setup

› Results

› Future work, optimization ideas
WHY NEW PROTOTYPE

› Software prototypes investigated and not selected
  – OVS: well-established, open source
    › mainly for virtual environment, performance issues
      - OVS on Intel® DPDK (OVDK) is an ongoing activity
  – CPqD softswitch: used by ONF for prototyping new features, open source
    › serious performance limitations
  – Linc: Erlang based softswitch, open source
    › runs in a VM environment, while we are primarily interested in close to the hardware solutions

› Hardware based prototypes / products
  – they have serious limitations in terms of number of rules
    › usually OF implementations use TCAM which has limited capacity
  – usually hard to program / modify / add new features
Simple MAC based forwarding
- 1, 10, 100, 1000, 2000 and 5000 DMAC rules
  - currently with linear search
  - always the last rule will match → caching is not easy…
- instruction = write action
  - action set = Output (egress port)

Intel® DPDK based generator station (tgen)
- generates 15 Mpps (@ 64 Bytes / pkt) on one core
MEASUREMENT SETUP

3Com (mgmt) 172.31.32.0/24

1G

Generator

Intel Xeon E5-2630
2x6 cores @ 2.3 GHz
8x4 GB DDR3 SDRAM

10G

Intel Niantic (82599EB)
2x10 GbE

1G

OF-SW

Intel Xeon E5-2630
2x6 cores @ 2.3 GHz
8x4 GB DDR3 SDRAM
Main results: 25% overhead vs. L2FWD (Intel’s example)
- it was more without highly optimizing the software
- linear with nr. of rules (not surprisingly)

So we began some investigation…
Processing time per number of rules
- at small number of rules cache(s) are effectively used
- note that real traffic would behave better
Preliminary results of current code: overhead was completely removed
AND FURTHER...

› Current status: basically removed “static” OF overhead
› It’s time for improving rule processing speed and implement control plane

› Basic ideas under discussion
  – high-performance southbound interface
    › minimize the need for locking, timeouts, etc.
  – fast data plane execution
    › flow caching
    › lookup algorithm selection, “selective TTP”
    › usage prediction