Adapting HW to SW for the Cloud

Han Dong

Mass Open Cloud Workshop

10/30/2018

http://sesa.github.io/

Addressing Performance of Datacenter Workloads

- Kernel Bypass (Unikernels, User-level network stacks, DPDK, FaRM, …)

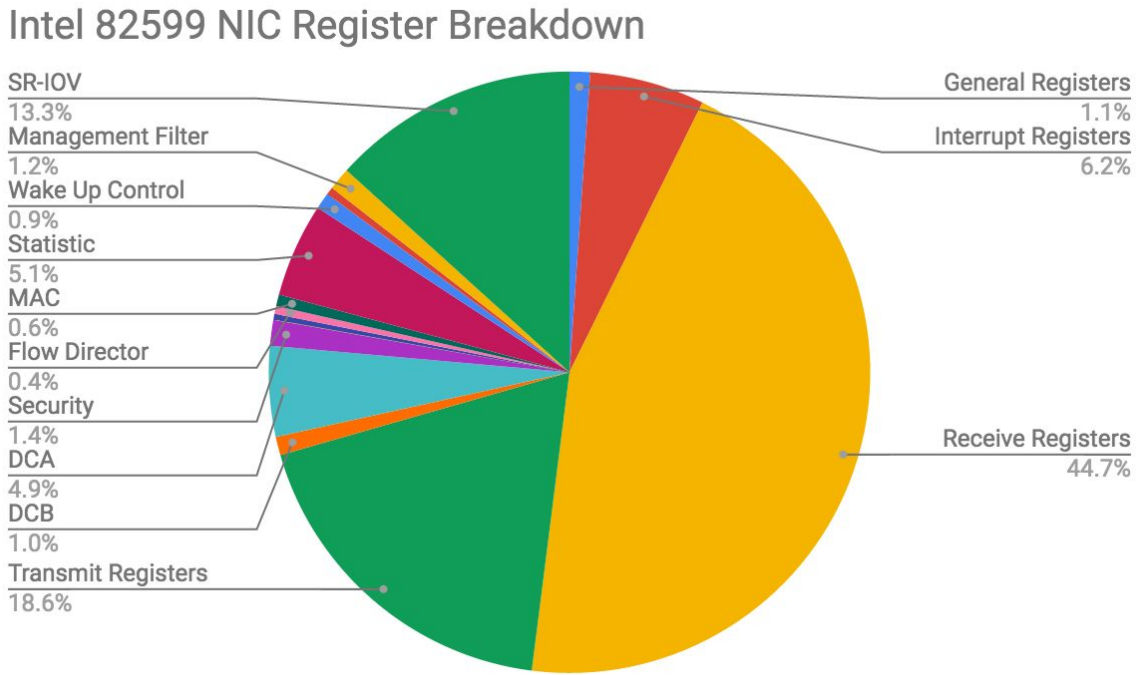
- Hardware Offload (Near-data Processing, Accelerators, FPGAs, Disaggregated Datacenters, …)

Opportunity to dynamically adapt the HW to the SW

Intel 82599 Datasheet

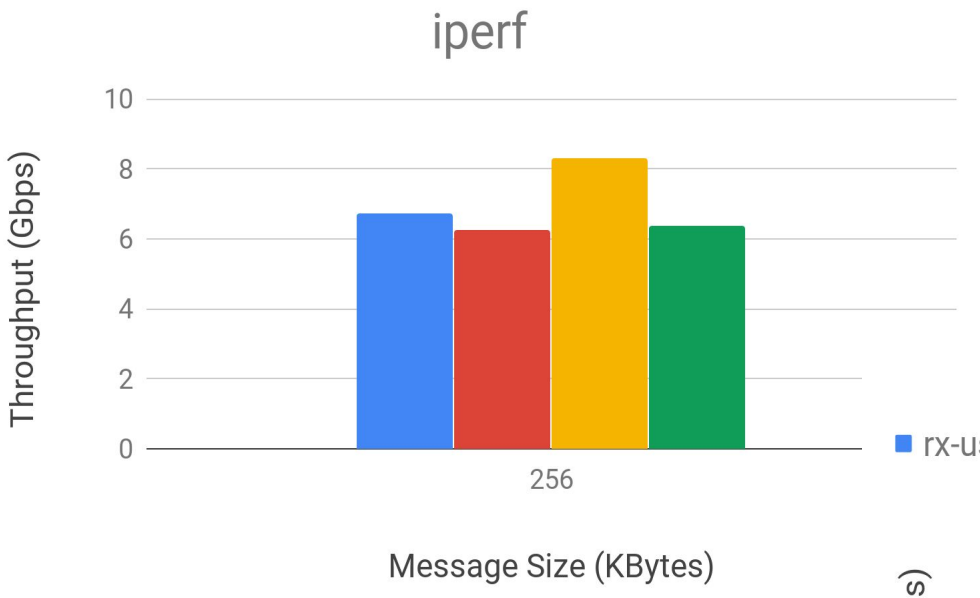
Total Registers:
~5630

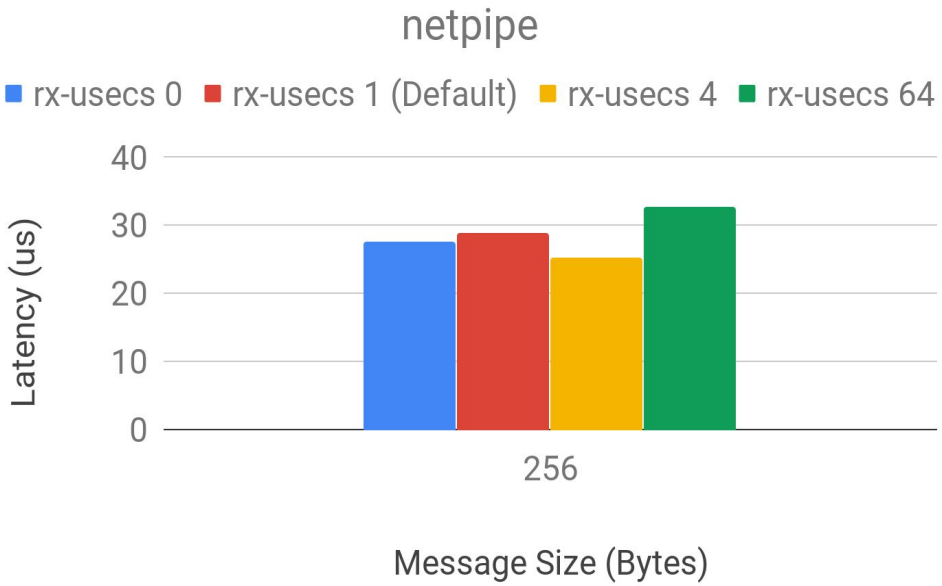
Linux Device Register Initialization: ~1360

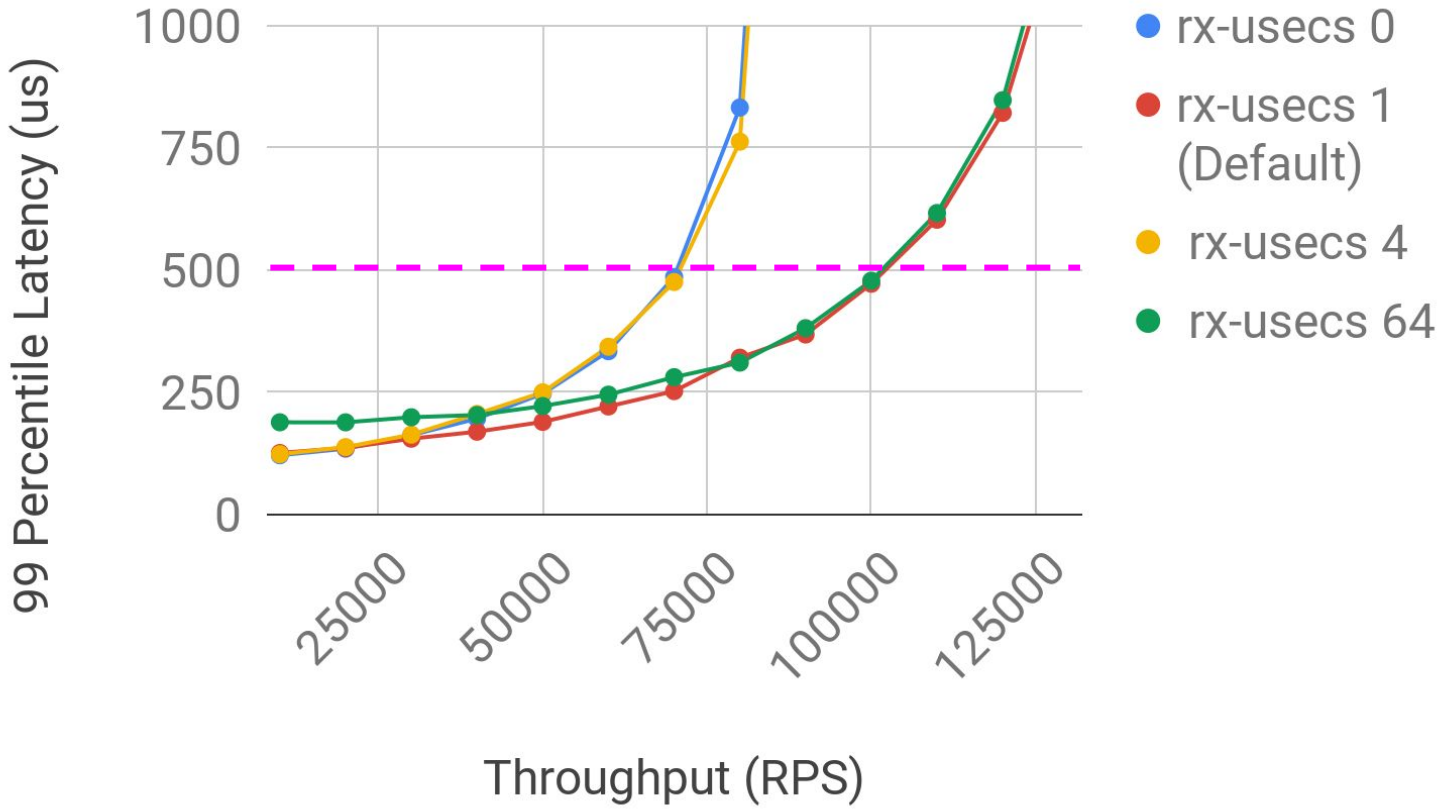


Experimental Setup

1. Two MOC machines:
    a. CPU: 2 * 8 Core Intel(R) Xeon(R) CPU E5-2650
    b. NIC: Intel Corporation Ethernet X520 10GbE
2. Linux ethtool:
    a. rx-usecs - how many usecs to delay an RX interrupt after a packet arrives
        i. 0us, 1us (Linux Default), 4us, and 64us
3. Benchmarks:
    a. iperf, netpipe, memcache (Facebook ETC workload)


iperf
Throughput (Gbps)
10
8
6
4
2
0
256
Message Size (KBytes)


netpipe
rx-usecs 0
rx-usecs 1 (Default)
rx-usecs 4
rx-usecs 64
Latency (us)
40
30
20
10
0
256
Message Size (Bytes)


memcache
99 Percentile Latency (us)
1000
750
500
250
0
25000
50000
75000
100000
125000
Throughput (RPS)
rx-usecs 0
rx-usecs 1 (Default)
rx-usecs 4
rx-usecs 64

Questions

- No single configuration works for all workloads.
- Which registers have an impact on network benchmarks?
  - Which registers have more “weight”?
- How to begin discovery of said registers?
- Is there a way to expose settings up to application?
- What about other devices?

Future Directions

1. Frame configuration problem as a learning problem:
    a. NIC contains ~5k unique registers, ~80 statistic registers
    b. ethtool contains ~100 device features that are tunable in Linux
2. Dynamically modify device features as workload changes:
    a. Cache injection, interrupts, packet filter
3. Per application optimization of HW and SW via unikernels or kernel bypass