Linux and NFV Testing and Tuning

Document History

Version	Date Updated	Updated By	Comments
1.0	Jan 2017	James Bensley	Initial draft

Document Distribution

Contents

References

AF_PACKET / PACKET_MMAP / PACKET_FANOUT:
https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt
http://man7.org/linux/man-pages/man7/packet.7.html
https://github.com/JustinAzoff/can-i-use-afpacket-fanout
http://yusufonlinux.blogspot.de/2010/11/data-link-access-and-zero-copy.html?m=1
https://github.com/jwbensley/EtherateMT/wiki/EtherateMT-PACKET_MMAP-Mode
https://github.com/jwbensley/EtherateMT/wiki/EtherateMT-Transmit-Overview
https://github.com/jwbensley/EtherateMT/wiki/Linux-Kernel-tracing-for-sendto()-using-AF_PACKET,-PACKET_MMAP-and-PACKET_FANOUT
http://kukuruku.co/hub/nix/capturing-packets-in-linux-at-a-speed-of-millions-of-packets-per-second-without-using-third-party-libraries
https://gist.github.com/pavel-odintsov/c2154f7799325aed46ae
https://gist.github.com/pavel-odintsov/15b7435e484134650f20
https://git.zx2c4.com/linux/plain/tools/testing/selftests/net/psock_fanout.c
https://home.regit.org/2012/07/suricata-to-10gbps-and-beyond/

Cisco CSR1000v:
Cisco Virtual Routers, CSR 1000V and ISRv - The Impact of Configuration Changes On Throughput Performance An Independent Assessment
Troubleshooting packet flow in CSRv

CPU & NUMA:
http://www.breakage.org/2013/08/30/oh-did-you-expect-the-cpu/
http://docs.openvswitch.org/en/latest/intro/install/dpdk/#performance-tuning
http://dpdk.org/doc/guides-2.0/rel_notes/faq.html#without-numa-enabled-my-network-throughput-is-low-why
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-network-packet-reception.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_for_Real_Time/7/html/Tuning_Guide/Offloading_RCU_Callbacks.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_for_Real_Time/7/html/Tuning_Guide/Isolating_CPUs_Using_tuned-profiles-realtime.html
http://www.breakage.org/2013/11/15/nohz_fullgodmode/
https://01.org/linuxgraphics/gfx-docs/drm/admin-guide/pm/intel_pstate.html
https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt
https://www.kernel.org/doc/Documentation/kernel-per-CPU-kthreads.txt

DMA:
https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#dma-engines
https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#adjusting-the-netrxaction-budget

https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#procnetsoftnetstat

DPDK:
DPDK Intel NIC Performance Report Release 17.02
https://software.intel.com/en-us/articles/dpdk-performance-optimization-guidelines-white-paper

Hardware & NICs:
DPDK: How to get best performance with NICs on Intel platforms
https://sandilands.info/sgordon/segmentation-offloading-with-wireshark-and-ethtool
http://people.binf.ku.dk/~hanne/b2evolution/blogs/index.php/2015/03/04/ixgbe-failed-to-load-because

Huge Pages:
https://wiki.debian.org/Hugepages
http://dpdk.org/doc/guides/linux_gsg/sys_reqs.html#use-of-hugepages-in-the-linux-environment
http://unix.stackexchange.com/questions/66134/huge-page-and-performance-improvemnt
http://dpdk.org/doc/guides/linux_gsg/sys_reqs.html
http://dpdk-guide.gitlab.io/dpdk-guide/setup/hugepages.html
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/main-memory

HugePages on VMs:
http://dpdk.org/ml/archives/users/2016-December/001301.html
http://dpdk.org/ml/archives/users/2017-January/001395.html

Interrupts:
https://github.com/torvalds/linux/blob/master/Documentation/networking/scaling.txt
https://github.com/torvalds/linux/blob/master/Documentation/IRQ-affinity.txt
https://github.com/pavel-odintsov/fastnetmon/wiki/Traffic-filtration-using-NIC-capabilities-on-wire-speed-(10GE,-14Mpps)

Linux Networking Stack:
https://wiki.fd.io/view/File:FD.io_mini-summit_916_Vhost_Performance_and_Optimization.pptx
https://wiki.linuxfoundation.org/networking/kernel_flow
https://fasterdata.es.net/host-tuning/linux/

Performance Network Coding in Linux:
http://netoptimizer.blogspot.co.uk/2014/10/unlocked-10gbps-tx-wirespeed-smallest.html
http://stackoverflow.com/questions/20008707/minimizing-copies-when-writing-large-data-to-a-socket
http://www.linuxjournal.com/article/6345?page=0,1

http://stackoverflow.com/questions/9770125/zero-copy-with-and-without-scatter-gather-operations?rq=1
Network Buffers And Memory Management

Receive-Side Scaling (RSS) / Receive Packet Steering (RPS) / Receive Flow Steering (RFS):
https://lwn.net/Articles/362339/
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/main-network.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/network-rss.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/network-rps.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/network-rfs.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/network-acc-rfs.html
https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#procsoftirqs

Sockets:
https://linux.die.net/man/7/socket
http://man7.org/linux/man-pages/man2/setsockopt.2.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-network-commonque-soft.html
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-network-dont-adjust-defaults.html
https://github.com/jwbensley/EtherateMT/wiki/EtherateMT-Socket-Overview

SR-IOV:
http://events.linuxfoundation.org/sites/events/files/slides/20160715_LinuxCon_sriov_final.pdf
https://en.wikipedia.org/wiki/PCI-SIG
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Host_Configuration_and_Guest_Installation_Guide/sect-Virtualization_Host_Configuration_and_Guest_Installation_Guide-SR_IOV-How_SR_IOV_Libvirt_Works.html
http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf
http://wiki.libvirt.org/page/Networking#PCI_Passthrough_of_host_network_devices

Standards:
RFC2544 - Benchmarking Methodology for Network Interconnect Devices (for packet sizes 64, 128, 256, 1024, 1280, 1518)
RFC3393 - IP Packet Delay Variation Metric for IP Performance Metrics (IPPM) (for PDV/packet delay variation info)

Threading:
https://computing.llnl.gov/tutorials/pthreads/
http://man7.org/linux/man-pages/man3/pthread_attr_setaffinity_np.3.html
http://man7.org/linux/man-pages/man3/pthread_setaffinity_np.3.html
http://man7.org/linux/man-pages/man2/sched_setaffinity.2.html

Virtualisation / OVS / KVM / NFV:
Intel NFV Performance Optimization for Virtualized Customer Premises Equipment
Intel® Open Network Platform_Release 2.0 Performance Test Report
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/ch33s08.html

Introduction

Caveat: This document is made from the notes that I have collected during personal research projects and doesn't contain tuning tips and testing methodologies for every scenario. It focuses on the lower layers of the Linux networking stack (Ethernet down to the physical NIC) for both bare metal systems and virtual systems. This document is not a how-to or step-by-step guide on performance tuning or writing high performance code for Linux.

This document contains details about performance testing and tuning for a Linux host that runs high performance network functions (either in terms of high throughput or low latency) as either virtual functions or directly on bare metal. It also contains notes on different Linux features for writing high performance network functions and applications.

Lots of the text in the document is not written by me, as stated this document isn't a "how to" guide and it is expected that the reader already has a understanding of the technologies discussed. The document is made mostly from notes and snippets I have compiled to build a quick reference guide / cheat sheet / reminder.

I mainly work on CentOS and Ubuntu so they are the focus of the document. I'm not an expert on writing and testing networking performance functions, please sent corrections to jwbensley@gmail.com.

I use a bunch of scripts to help profile a box under load from a high level before using tools like `valgrind`, `cachegrind`, `perf` or `systemtap` to then drill deeper into application, kernel or network performance. They can all be downloaded using this wrapper script: download.sh and they are all available individually here: https://gist.github.com/jwbensley/ and referenced throughout this document.

The document starts with a high level view / quick reference, of different testing and perf tools, to provide a testing cheat sheet and then goes on to provide the background info.

Before doing anything though, as a general best practice tip, when testing in the lab it is always recommended to run the latest Kernel version that you can (that is still suitably stable for your needs). Bugs are constantly being fixed and performance improvements are being released in every single Linux version more or less. This also extends to devices drivers and firmware updates.

Testing and Monitoring

Below are some quick references for monitoring and testing, these work won't on ever system or for every workload. YMMV.

CPU Stats and Layout

Check number of CPU sockets, core, threads etc: cpu_layout.sh or cpu_layout.py

Also numactl will show NUMA nodes and cores per node:
`$ numactl -H`

`nproc` will incorrectly show the number of CPU cores in a system if isolcpus is being used.

To check which HT/logical cores share a physical core:
`$ cat /sys/devices/system/cpu/cpu0/topology/thread_siblings_list`

The following command shows the PCI bus for a NIC:
`$ ethtool -i eth0 | grep bus`

Then check:
`$ cat /sys/bus/pci/devices/<pci_bus_info>/numa_node`

Check the following to see softirqs balanced across CPUs:
`$ cat /proc/softirqs`
irq_display.sh
Irq_balance.sh

NIC Stats & Settings

As a general good note, ethtool queries a NIC's driver for stats which provide accurate stats at the NIC/PHY level. Some Linux commands are showing stats from higher up in the network stack and packets maybe have been dropped due to the socket SO_SNDBUF/SO_RCVBUF queues overflowing, or a QDISC queue overflow, or a lack of Tx descriptors, or all of the above. Using ethtool lets on focus accurately on just one area, the NIC.

Try disabling adaptive interrupt coalescing and set the Tx/Rx delay between interrupts to 20 microseconds. This stops the frame count being used as the interrupt delay and will regularly empty the Tx/Rx queue (good for high throughput, maybe bad for low latency)
`$ sudo ethtool -C eth0 adaptive-rx off`
`$ sudo ethtool -C eth0 adaptive-tx off`
`$ sudo ethtool -C eth0 rx-usecs 20`
`$ sudo ethtool -C eth0 tx-usecs 20`

Increasing the Tx/Rx queue length may be good for throughput and bad for latency
`$ sudo ethtool -G eth0 rx 4096`
`$ sudo ethtool -G eth0 tx 4096`

One can see CPU utilisation for interrupts using:
`$ mpstat -P ALL`

One can see interface stats using:
`$ sudo ethtool -S eth0`
`$ cat /sys/class/net/eth0/statistics/tx_aborted_errors`
`$ cat /proc/net/dev`
`$ sar -n EDEV 1`
`$ netstat -i`

Some NICs have LLDP enabled (such as Intel X710), disable per port with:
`$ sudo bash -c “echo lldp stop > /sys/kernel/debug/i40e/0000:09:00.0/command”`
`$ sudo bash -c “echo lldp stop > /sys/kernel/debug/i40e/0000:09:00.1/command”`

Queue Stats

Check device queue stats reading the files stored in
/sys/class/net/NIC/queues/tx-QUEUE_NUMBER/byte_queue_limits/
`$ cat /sys/class/net/wlp3s0/queues/tx-0/byte_queue_limits/*`

Show queueing disciplines:
`$ tc qdisc`

Show QDISC statistics (check if packets are bypassing the QDISC):
`$ tc -s qdisc show dev eth0`

The tx_inflight.sh script can be used to check the number of packets in the Tx queue.

One can see higher level stats for TCP/UDP/ICMP with netstat:
`$ netstat -s`

System Stats

The soft_irqs.sh script shows interrupt counts per CPU core.

The soft_net.sh script shows softIRQ back-pressure stats .

The Processor Counter Monitor (originally by Intel): https://github.com/opcm/pcm can be used to monitor many things, just some of them: instructions per cycle, core frequency QPI bandwidth, local and remote memory bandwidth, cache misses, core and CPU package

sleep C-state residency, core and CPU package thermal headroom, cache utilization, CPU and memory energy consumption, PCIe bandwidth per-socket/per PCIe device, local and remote NUMA memory accesses.

Open Source Tools

There are many great open source tools which have a range or use cases. Some can be used for generating and receiving various IP flows quickly with minimal install effort, but at lower throughputs or with less granularity. Some tools are more complex to set up but provided very specific traffic generation features and/or at high throughput or low latency. Some are very granular allowing for the crafting of any packet type (any value possible) but at a lower speed etc.

Some of these tools fit into multiple categories, this is just how I use them

Packet Crafting:
http://ostinato.org/
http://secdev.org/projects/scapy/
http://packeth.sourceforge.net/packeth/Home.html
http://netsniff-ng.org/ (mausezahn)

Layer 4 TCP/UPD/ICMP testing (easy to install easy to use):
https://github.com/facebook/UdpPinger
https://github.com/esnet/iperf

Layer 3 IP (more complex to install and use, higher performance)
https://trex-tgn.cisco.com/
http://pktgen-dpdk.readthedocs.io/en/latest/
https://github.com/pktgen/Pktgen-DPDK
https://github.com/emmericp/MoonGen

Layer 2-2.5 Ethernet and MPLS:
https://github.com/emmericp/MoonGen
https://github.com/jwbensley/Etherate

Application Tuning

AF_PACKET / PACKET_MMAP / PACKET_FANOUT

This page provides an overview of how PACKET_MMAP works in Linux using EtherateMT as the example program. This page discusses the Tx path through the Kernel when using PACKET_MMAP/AF_PACKET. This page provides a walkthrough of AF_PACKET socket Tx code in the Kernel.

The high level view of the Kernel PACKET_MMAP, AF_PACKET, FANOUT_FANOUT features are as follows:

Consider an example scenario in which XPS/RSS has been configured on a host/NIC; A system has at least 16 CPU cores and a NIC with 8 RX queues and 8 TX queues, each RX queue 0-7 sends an interrupt which has been mapped to CPU cores 0-7 respectively (using MSI-X) and each TX queue 0-7 has an interrupt which has been mapped to each CPU core 8-15 respectively (using MSI-X).

Rx path:

1. 8 Rx worker threads 0-7 are started, each one is pinned to a CPU core 0-7 respectively.

2. Each Rx worker thread creates a socket of type SOCK_RAW using the AF_PACKET address family, all bound to the same physical NIC, in promisc mode, and all part of the same PACKET_FANOUT group.

3. Each Rx worker thread creates an RX_RING buffer using MMAP and associates it with it’s unique socket.

4. At this point for example, worker thread 0 is bound to CPU core 0 which created socket 0, and the NIC Rx queue 0 interrupt is mapped to CPU core 0 also (MSI-X). This thread now enters an infinite loop calling read() or using poll()/select() against socket 0 only.

5. When a packet comes into NIC Rx queue 0, the NIC places the packet into DMA memory in RAM.

6. An interrupt is sent to CPU 0 (MSI-X mapping). CPU 0 copies the packet into Kernel receive buffer into an SKB.

7. Because the interrupt was processed on CPU 0 the Kernel uses that CPU ID as a key into a hash which ultimately returns from the FANOUT socket group that socket 0 is the destination socket for the packet.

8. The Kernel copies the packet from the SKB into the RX_RING associated with socket 0 in the FANOUT group and marks the packet status in the ring buffer indicating there is a packet ready to be read.

9. The Rx worker thread on CPU 0 can poll() the RX_RING and once it sees a block in the ring with TP_STATUS_USER set it can access the packet inside the block (inside MMAP'ed RX_RING).

That explanation skipped over the Kernel receive queue (QDISC bypass is used in EtherateMT).

The Tx path is similar but in reverse:

1. Each Tx worker thread 0-7 creates a socket, all in the same FANOUT group, all bound to the same physical NIC, all in promisc mode, all raw sockets, all using AF_PACKET, each one has MMAPed a TX_RING, etc. Just like the Rx worker threads.

2. Each worker thread 0-7 is bound to CPU cores 8-15 respectively.

3. A worker thread can fill it's TX_RING blocks with frames and mark each block as ready for Tx.

4. Next worker thread 0 for example calls send() only once, and the Kernel will dequeue all packets in the Tx thread 0 / socket 0 Tx ring in one context switch instead of once per packet.

5. The Kernel will copy each packet/frame from the TX_RING into individual SKBs in Kernel memory.

6. The Kernel thread now running on the CPU core that the Tx worker thread was assigned to, CPU core 8, will signal to the NIC that packets are ready to be sent.

7. This will generate an interrupt by the NIC to get CPU time to process the pending Tx packets. This interrupt will be processed on CPU core 8 due to the MSI-X mapping of NIC Tx queue 0 to CPU core 8, mentioned earlier.

8. After signalling the NIC that packets are ready to be send, the NIC will copy each packet from DMA memory space to it's hardware Tx queue. So at least two copies are required to send a packet.

Note: The only way PACKET_MMAP saves on copy overheads if it the userland application builds and modifies packets inside the TX_RING directly. It does save on contexts switches though as one syscall can be used to train a RING. This in turn reduced cache misses. Also using raw sockets skips a big chunk of the Kernel networking stack when compared using higher level TCP sockets for example.

That explanation skipped over the Kernel transmit queue (QDISC bypass is used in EtherateMT).

TPACKET v2 (used for Tx and Rx rings in AF_PACKET sockets with PACKET_MMAP) fully supports Tx and Rx queues however TPACKET v3 originally only support Rx. A recent Kernel is required for TPACKET v3 Tx support!

AF_PACKET vs DPDK

Despite what has been said above about AF_PACKET, DPDK is still much faster.

AF_PACKET has no support for HugePages, separate TLB entries are used for both the userland and Kernel land processes involved.

DPDK also reduces the number of copies; the NIC is disconnected from the Kernel and the userland application uses a custom driving to connect to the NIC which is then able to allocate and write to / read from the DMA ranges directly.

When using VLANs with AF_PACKET; on packet Rx the VLAN is stripped by the Kernel and written as metadata into an SKB, AF_PACKET then copies the VLAN ID from the SKB metadata into the AF_PACKET TPACKET header again to present to the userland process, which is wasteful.

AF_PACKET runs in the Kernel which means no changes can be made to its code without a recompile and reboot and a bug in AF_PACKET would likely crash the Kernel. DPDK applications are userland applications and can crash or be restarted without impact on the wider system.

DMA

"...when exiting the net_rx_action loop and when additional work could have been done, but either the budget or the time limit for the softirq was hit. This statistic is tracked as part of the struct softnet_data associated with the CPU. These statistics are output to /proc/net/softnet_stat…"

This script can be used to track softnet_stats during network activity: soft_net.sh

One must check the Linux source code to find out exactly what each fields means in softnet_stat as it can change between Linux versions:
https://github.com/torvalds/linux/blob/v3.13/net/core/net-procfs.c#L161-L165

From the above article about 3.13.0:

Each line of /proc/net/softnet_stat corresponds to a struct softnet_data structure, of which there is 1 per CPU. The values are separated by a single space and are displayed in hexadecimal:

- The first value, sd->processed, is the number of network frames processed. This can be more than the total number of network frames received if you are using ethernet bonding. There are cases where the ethernet bonding driver will trigger network data to be re-processed, which would increment the sd->processed count more than once for the same packet.
- The second value, sd->dropped, is the number of network frames dropped because there was no room on the processing queue.
- The third value, sd->time_squeeze, is the number of times the net_rx_action loop terminated because the budget was consumed or the time limit was reached, but more work could have been. Increasing the budget can help reduce this.
- The next 5 values are always 0.
- The ninth value, sd->cpu_collision, is a count of the number of times a collision occurred when trying to obtain a device lock when transmitting packets.
- The tenth value, sd->received_rps, is a count of the number of times this CPU has been woken up to process packets via an Inter-processor Interrupt.
- The last value, flow_limit_count, is a count of the number of times the flow limit has been reached. Flow limiting is an optional Receive Packet Steering feature.

We can increase the packet processing budget for all NAPI structures on a CPU, default is 300 on Linux 3.13.0:

```
$ sudo sysctl -w net.core.netdev_budget=600
# This has no effect on Etherate Tx, does it help Rx?
```

When incoming packets are copied by the NIC into the DMA space in RAM and the Kernel is signalled that new packets have arrived, each packet is copied into an SKB in Kernel memory space. This can be upto a few kilobytes even for 64 byte packets (in a worst case scenario).

In the Tx path SKBs add in a reasonable amount of memory overhead per packet. When using a raw socket in the Tx path part of that overhead is removed. In the Rx path if the incoming packet is a valid IP/TCP/UDP packet the Kernel will add in the extra memory overhead (building up that higher layer meta data) in the SKB. Etherate and EtherateMT send “junk” data (random data this isn’t valid IP or TCP/UDP) to prevent and higher layer interaction.

Sockets & Socket Settings

Like the NIC hardware queue, the socket queue is filled by the network stack from the softirq context. Applications then drain the queues of their corresponding sockets via calls to read, recvfrom, and the like.

rmem_default/wmem_default:

A kernel parameter that controls the *default* size of receive buffers used by a socket...To determine the value for this kernel parameter, view /proc/sys/net/core/rmem_default. Bear in mind that the value of rmem_default should be no greater than rmem_max (/proc/sys/net/core/rmem_max); if need be, increase the value of rmem_max. Equally wmem_default and wmem_max control the socket send buffer size.

One can try to enable busy polling (requires driver support and a socket option on the application socket) to reduce latency at the potentially cost of increased power requirements:
`$ sysctl -w net.core.busy_poll = 50`
`$ sysctl -w net.core.busy_read = 50`

This page discusses how sockets work when using AF_PACKET using EtherateMT as the example application.

Once can check for hardware timestamping capabilities on a NIC using:
`$ sudo ethtool -T eth0`

One can disable packet timestamps using the following:
`$ sudo sysctl -w net.core.netdev_tstamp_prequeue=0`

There are various options that can be used on Linux socket() and setsockopt() to improve performance if they are suitable for the application using the socket:

SO_RCVBUF/SO_RCVBUFFORCE/SO_SNDBUF/SO_SNDBUFFORCE - One can increase the size of the socket tx queue so that the application doesn't block or drop packets during tx if the Kernel thread draining the queue is busy. Vice versa with the rx queue, so that the Kernel thread doesn't block when receiving packets because the application thread is too busy to drain the queue.

PACKET_FANOUT - To scale processing across threads, packet sockets can form a fanout group. In this mode, each matching packet is enqueued onto only one socket in the group...Fanout supports multiple algorithms to spread traffic between sockets.

PACKET_LOSS (with PACKET_TX_RING) - When a malformed packet is encountered on a transmit ring, the default is to reset its tp_status to TP_STATUS_WRONG_FORMAT and abort the transmission immediately. The malformed packet blocks itself and subsequently enqueued packets from being sent. The format error must be fixed, the associated tp_status reset to TP_STATUS_SEND_REQUEST, and the transmission process restarted via send(). However, if PACKET_LOSS is set, any malformed packet will be skipped, its tp_status reset to TP_STATUS_AVAILABLE, and the transmission process continued.

PACKET_RX_RING - Create a memory-mapped ring buffer for asynchronous packet reception. The packet socket reserves a contiguous region of application address space, lays it out into an array of packet slots and copies packets (up to tp_snaplen) into subsequent slots.

PACKET_TX_RING - Create a memory-mapped ring buffer for packet transmission. This option is similar to PACKET_RX_RING and takes the same arguments.

When using PACKET_RX_RING and PACKET_TX_RING the userland application can send or receive a batch of packets (up to the Tx/Rx ring size or socket Tx/Rx queue size, whichever is smaller) in a single syscall and single context switch.

PACKET_TIMESTAMP (with PACKET_RX_RING) - The packet receive ring always stores a timestamp in the metadata header. By default, this is a software generated timestamp generated when the packet is copied into the ring. This integer option selects the type of timestamp. Besides the default, it support the two hardware formats.

When using PACKET_TX_RING and PACKET_RX_RING, for the Tx ring by default no timestamp is generated. For the Rx ring if neither TP_STATUS_TS_RAW_HARDWARE or TP_STATUS_TS_SOFTWARE are set then the software fallback is invoked *within* PF_PACKET's processing code, which is less precise and more overhead that if hardware timestamps were enabled. Try to off load the Rx timestamp to hardware.

PACKET_QDISC_BYPASS - By default, packets sent through packet sockets pass through the kernel's qdisc (queuing discipline / traffic control) layer, which is fine for the vast majority of use cases. For traffic generator appliances using packet sockets that intend to brute-force flood the network - for example, to test devices under load in a similar fashion to pktgen - this layer can be bypassed by setting this integer option to 1. A side effect is that packet buffering in the qdisc layer is avoided, which will lead to increased drops when network device transmit queues are busy.

Threading

Note: The Pthreads API does not provide routines for binding threads to specific cpus/cores. However, local implementations may include this functionality - such as providing the non-standard pthread_setaffinity_np routine. Note that "_np" in the name stands for "non-portable".

“I think it goes without saying (so why am I saying it?) a single threaded scalar approach to sending or receiving data and processing it in today's world of multi-core CPUs, multi-channel RAM, and multi-queue NICs isn’t going to stress test networking throughput and latency of a host device or network device to its full potential. Threads are used in EtherateMT instead of multiple processes using MPI (message-passing interface) to give higher memory throughput for (NUMA) on-node communications. Also pthread_create() is much more lightweight than fork(). Additionally the EtherateMT author is lazy and pedantic at the same time so the pthread library is used instead of boost (as well as various other idiosyncrasies throughout the code).”

DPDK

DPDK CLI Arguments

One can set up separate hugepage allocations per DPDK application running on the same device.

First mount unique hugepage allocations in the fstab:
nodev /mnt/huge_app1_1G hugetlbfs rw,relatime,pagesize=1G,size=1G 0 0
nodev /mnt/huge_app2_1G hugetlbfs rw,relatime,pagesize=1G 0 0,size=2G 0 0

When starting a DPDK application use the following CLI option to specify which huge page allocation to use:

--huge-dir Directory where hugetlbfs is mounted

Also use the following CLI option to specify which NUMA node the allocation should be made against, specified as a comma separated list (as the fstab feature is not able to specify this):

--socket-mem Memory to allocate on sockets (comma separated values)

```
$ sudo ./my_dpdp_app --huge-dir /mnt/huge_app1_1G --socket-mem 1024,0
```

CPU Tuning

CPU Isolating/Pinning/Scheduling

Note: the irqbalance service should be disabled when using isolcpus/nohz_full and tasks should be manually allocated to the appropriate CPU core for best performance.

Note: Hyperthreading is generally advised to be disabled so that only “real” cores are used for workloads. Hyperthreading can incur latency from switching out CPU registers to switch between processing threads for example. This depends on the workload though. Pushing line rate data transfers for example or zero loss packet tests benefit from disabling hyperthreading, lower bandwidth NFV CPEs however with oversubscribed vCPUs hyperthreading may be useful for scaling the number of VMs.

The isolcpus Kernel option can be used to isolate cores from the Linux scheduler. The isolated cores can then be used to dedicatedly run HPC applications or threads (reducing interrupt contention). This helps application performance due to zero/minimal context switching and minimal cache thrashing. To run platform logic on core 0 and isolate cores between 1 and 19 from scheduler, add isolcpus=1-19 to GRUB cmdline. NOTE: It has been verified that core isolation has minimal advantage due to mature Linux scheduler in some circumstances.

nohz_full=*cpulist*
The *nohz_full* Kernel parameter is used to treat a list of CPUs differently, with respect to timer ticks. If a CPU is listed as a nohz_full CPU and there is only one runnable task on the CPU, then the kernel will stop sending timer ticks to that CPU, so more time may be spent running the application and less time spent servicing interrupts and context switching.

Note:

1. If you have a multithreaded application where threads need to communicate with one another by sharing cache, then they may need to be kept on the same NUMA node or physical socket.

2. If you run multiple unrelated real-time applications, then separating the CPUs by NUMA node or socket may be suitable.

Check NUMA layout on Ubuntu:
`$ sudo apt-get install hwloc-nox`
`$ lstopo-no-graphics --no-io --no-legend --of txt`
or
`$numactl -s`
or
`$ numactl --hardware`

Note: nohz_full and rcu_nocbs is to disable Linux Kernel interrupts, and it's important for zero-packet loss tests.

For example, to ensure CPU 0 will get (almost) all interrupts use the following Kernel boot parameters:
nohz_full=1-17 (No kernel interrupt)
isolcpus=1-17 (Only taskset & co. will assign a process to these corer, also nproc won't show these cores)
rcu_nocbs=1-17 (Offload rcu callbacks)

Add the following to the GRUB_CMDLINE_LINUX value in /etc/default/grub: "isolcpus=1-17 nohz_full=1-17 rcu_nocbs=1-17"

Then update grub and reboot:
```
$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```
Or for UEFI systems:
```
$ sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg
```

When performing zero packet loss tests one can disable the soft lockup detector watchdog process and hard lockup detector (Non-Maskable Interrupts) to reduce overheads:
```
$ echo 0 > /proc/sys/kernel/watchdog
$ echo 0 > /proc/sys/kernel/nmi_watchdog
```

Alternatively one can mask out certain cores from the watchdog processes using:
```
$ echo "0,18-23" > /proc/sys/kernel/watchdog_cpumask
```

When nohz_full is used those cores are masked out by default.

When using isolcpus/rcu_nocbs/nohz_full the default Linux scheduler won't allow tasks to be scheduled to those isolated cores. Even when using `taskset`. One must use `chrt` too. For example, the following will run EtherateMT with 3 worker threads on cores 1-4 and use scheduling policy SCHED_RR (round-robin):
```
$ sudo chrt -r 1 taskset -c 1-4 ./etherate_mt -i ens2f1 -c 3
```

isolcpus/nohz_full/rcu_nocbs are Kernel boot options, during run time though the following can be used to ensure that no IRQs are mapped to the CPU cores being used for the important network function:
```
$ for i in $(ls /proc/irq/ | grep [0-9]); do
    echo 1 > /proc/irq/$i/smp_affinity ;
done
```

Disable Intel's processor turbo-boost: sudo wrmsr -p <cpu> 0x1a0 0x4000850089
I haven't tested this yet

CPU Frequency and Scaling

“The Linux scheduler is not optimized for VM scheduling and doesn’t understand the relationship between the multiple vCPUs in a virtual machine. The Linux scheduler can schedule the vCPU of a CSR VM across the different cores in the system. This will negatively impact the CSR forwarding performance as the context switching thrashes the processor instruction and data caches. To overcome this, the vCPUs should be manually pinned to the underlying physical cores...When pinning vCPUs to physical cores, the processor NUMA node configuration should also be taken into consideration. The vCPUs should be pinned to the processor cores from the same NUMA node...Assigning the physical cores from the same NUMA node as the physical NICs helps in increased throughput.”

“We ran most tests at the nominal CPU speed, 2.3 GHz, and saw better performance with intel_pstate=disabled. Later, we ran a few additional tests on an x86 server with a 2.6 GHz CPU and saw some performance improvement with the faster CPU. All our test configurations used CPU pinning to designate CPUs for command and data and 1 GB hugepages to better handle the data operations.”

The Intel PSTATE CPU governor seems to have been quite buggy over the years, some people recommend to disable this and use a different CPU governor, for some people it works fine, for some it works fine for a period of time then stops working (it’s overall benefit also depends on which CPU model one has). The following details how to disable it, if it is causing problems:

To set the Intel P state to disabled add the following Kernel boot argument in /etc/default/grub:
`GRUB_CMDLINE_LINUX="intel_pstate=disable”`
Update grub with:
`$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg`
or
`$ sudo grub2-mkconfig -o /etc/grub2-efi.cfg`
Or for UEFI systems:
`$ sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg`

After a reboot, verify with:
`$ cat /proc/cmdline`

CPU scaling will use less power but it means that when a stream of packets comes into a machine there might be a delay whilst the CPU ramps up to full clock speed (a “wake” CPU latency). Some people set their CPUs to “performance” mode to improve network latency at the cost of extra power, for some the latency increase is negligible. To force the CPU to performance mode (this is not power efficient!) in real time one can use the below commands. Use the tool `turbostat` to measure the different C-states.

First verify the CPU frequency and scaling governor by using the following commands:

```
$ grep -E '^model name|^cpu MHz' /proc/cpuinfo
model name    : Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
cpu MHz       : 1200.000

$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do sudo cat $CPU;
done
conservative

$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq; do sudo cat $CPU;
done
1200000

$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_min_freq; do sudo cat $CPU;
done
1200000

$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_max_freq; do sudo cat $CPU;
done
2100000
```

Available CPU power state governors:

```
$ ls -1 /lib/modules/`uname -r`/kernel/drivers/cpufreq/
acpi-cpufreq.ko
amd_freq_sensitivity.ko
cpufreq_stats.ko
p4-clockmod.ko
pcc-cpufreq.ko
powernow-k8.ko
speedstep-lib.ko
```

Current CPU power state governor:

```
$ cpupower frequency-info | grep driver
  driver: pcc-cpufreq

$ lsmod | grep pcc
pcc_cpufreq            14082  0
```

Available CPU governor profiles:

```
$ cpupower frequency-info | grep governors
  available cpufreq governors: conservative userspace powersave ondemand performance
```

Use the below script to change the scaling governor setting to “performance” (this is not power efficient!):

```
$ for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do sudo bash -c "echo -n performance > $CPUFREQ"; done
```

The same thing happens when using:
```
$ tuned-adm profile network-latency
```
Another option is…
```
$ tuned-adm profile throughput-performance
```

```
$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do sudo cat $CPU; done
performance
```

```
$ for CPU in /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq; do sudo cat $CPU; done
2100000
```

One can also use the following commands, for RHEL/CentOS systems:
```
$ cpupower frequency-set -g performance
```

For Debian/Ubuntu systems:
```
$ cpufreq-set -r -g performance
```

To make the change permanent on the host across system reboot, for Ubuntu, modify the /etc/init.d/ondemand script. Change this:
```
echo -n ondemand > $CPUFREQ
```
to this:
```
echo -n performance > $CPUFREQ
```

Also stop any running daemons such as `cpuspeed`, `cpufreqd`, `powerd` that control the CPU stepping. For this execute the following command on Ubuntu:
```
$ service cpuspeed stop
```
On RHEL 6 run:
```
$ chkconfig cpuspeed off –level 1
```
On CentOS 7:

```
$ chkconfig cpuspeed off –level 1
```

NUMA and PCI Affinity

“With NUMA disabled in the BIOS the memory controller interleaved memory access across the multiple CPU sockets. A read operation may complete on the local or mote socket memory. A remote memory accesses will end up crossing the QPI link and incur latency.” - So, it's probably best to leave NUMA enabled but carefully manage the placement of hardware interrupts and application to CPU core mapping to ensure locality.

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/ch33s08.html

To take advantage of this, place process affinity of applications expected to receive the most data on the NIC that shares the same core as the L2 cache. This will maximize the chances of a cache hit, and thereby improve performance.

Below it can be seen that a dual port 10G NIC is present in a test system. It is a PCIe 3.0 card which shows support for 8GT/s and 8 PCIe lanes. PCIe 3.0 supports 984.6 MB/s per-lane. Sending a single packet requires at least two round trips across the PCIe bus: one RTT to notify the NIC about the packet descriptor waiting to be sent and one RTT for the NIC to retrieve the packet from the DMA region in RAM. For 10Gbps of unidirectional throughput, two PCIe 3.0 lanes worth of bandwidth are required (that provides 15,760Mbps of bandwidth). However, at 10 Gbps using 64 byte Ethernet frames (84 bytes at layer 1) one frame is transmitted every 67.2 nanoseconds. Each PCIe operation introduces latencies and jitter in the nanosecond-range so two RTTs across the PCI bus is not negligible.

```
$ lspci | grep Eth
09:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ (rev 01)
09:00.1 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ (rev 01)

$ lspci -s 09:00 -vv | grep LnkSta
                LnkSta: Speed 8GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
                LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete+, EqualizationPhase1+
                LnkSta: Speed 8GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
                LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete-, EqualizationPhase1-
```

One can use these scripts to check the CPU core and socket layout of a machine, the provide similar output, one in Bash and one in Python:
cpu _layout.sh
cpu_layout.py

The command `nproc` will provide the total number of CPU cores (however nproc won't show cores excluded by isolcpu) and the `lscpu` command will provide the total number of CPU cores, and the number of sockets, and NUMA nodes.

Also numactl will show NUMA nodes and cores per node:
`$ numactl -H`

To check which HT/logical cores share a physical core:
```
$ cat /sys/devices/system/cpu/cpu0/topology/thread_siblings_list
```

Once the CPU layout is known, it is usually beneficial to have any traffic heavy applications running on the same NUMA node as interrupts for the NIC(s) they use (to prevent memory accesses across the QPI link which incurs additional latency, improving cache locality). To see what NUMA node a NIC is bound to, first get the NIC PCI ID:

```
$ lspci -nn | grep Eth
09:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ [8086:1572] (rev 01)
09:00.1 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ [8086:1572] (rev 01)
```

Or, the following command shows the PCI bus for a NIC:
```
$ ethtool -i eth0 | grep bus
```

Adjusting the following file changes the NUMA node affinity for a NIC (this way one can align the NIC to the same NUMA node as the IRQ cores and application cores):

```
$ cat /sys/bus/pci/devices/0000\:09\:00.0/numa_node
```

NIC Settings

General NIC Settings

Just as a recent Kernel is advisable, recent NIC drivers and firmware beyond the stock versions are also recommended. For some Intel cards VLAN promiscuous mode and Unicast promiscuous mode features weren't added until more recent VF driver versions.

Disabling certain NIC hardware offload features can improve performance:
`$ sudo ethtool–K <vnetN> tso off gso off gro off`

Note that many PFs/NICs and/or their drivers have a limitation of only supporting 64 VLANs. Often the first VLAN (VLAN 0, as in index 0 not the actual VLAN tag 0!) is used by the PF meaning VFs can only use a maximum total aggregate of 63 VLANs.

Some NICs don't allow third party SFPs without setting a option in the Kernel module when it's loaded. For example, load the ixgbe driver using:

```
$ sudo modprobe ixgbe allow_unsupported_sfp=1
```

To make it permanent across reboots add "options ixgbe allow_unsupported_sfp=1" to /etc/modprobe.d/ixgbe.conf.

Another method to make the change permanent is to add this option to the kernel boot args in /etc/default/grub:
`GRUB_CMDLINE_LINUX="ixgbe.allow_unsupported_sfp=1"`

Then update grub:
`$ grub-mkconfig -o /boot/grub/grub.cfg`
or
`$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg`
or
`$ sudo grub2-mkconfig -o /etc/grub2-efi.cfg`
Or for UEFI systems:
`$ sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg`

And check after a reboot with
`$ cat /proc/cmdline`

This option is an array, so for each port on the card 1 or 0 must be specified, so for a 2 port card use "allow_unsupported_sfp=1,1".

For the i40e drivers (latest is 1.6.42 at the time of writing) no such option exists however third party SFPs do seem to be working with some cards, YMMV!

Interrupt Scaling

A hardware feature called “Interrupt Throttling” (also called “Interrupt Coalescing”) can be used to to pace the delivery of interrupts to the CPU. This can be controlled via ethtool. Fewer interrupts generated result in higher throughput, increased latency, and lower CPU usage. More interrupts generated result in the opposite: lower latency, lower throughput, but also increased CPU usage.

The following command will show the interrupt ID for each hardware device and how many times each CPU core has processed that interrupt:
`$cat /proc/interrupts`

NIC Interrupts and CPU Affinity

It is important to note that with some drivers such as the IXBGE driver for example, the same softIRQ NET_RX is used to process incoming packings but also the “completion” part of a transmission. NET_TX softIRQ is used to send packets, after they have been transmitted on the wire a softIRQ is raised to process their completion (clean up DMA addresses, Tx descriptors, free SKB’s etc.) and that softIRQ is the NET_RX softIRQ. This means when checking in /proc/softirqs and using EtherateMT in transmit mode, one may see a high number of NET_RX softIRQs even though data is being transmitted not received (because the NET_RX is running the post transmit actions in the Kernel).

The irq_display.sh script will show which CPU core is assigned to process the interrupt for each NIC queue. The irq_balance.sh balance script will spread each NIC queue (interrupt) for a given NIC across each CPU core in a “lazy” approach; if there are more NIC queues (interrupts) than there are CPU cores, interrupts are each mapped 1:1 to a CPU core up to the number of CPU cores, and for each additional remaining interrupt more than there are CPU cores, they are all lumped onto the final core.

These scripts provide a quick method to enable RSS/XPS.

Check the following to see softirqs balanced across CPUs:
`$ cat /proc/softirqs`

When changing interrupt to CPU mapping the irqbalance service should be disabled.

Receive-Side Scaling (RSS) / Receive Packet Steering (RPS) / Receive Flow Steering (RFS)

Check the following to see softirqs balanced across CPUs:
`$ cat /proc/softirqs`

"... Be careful using just this file when monitoring your performance: during periods of high network activity you would expect to see the rate NET_RX increments increase, but this isn't necessarily the case. It turns out that this is a bit nuanced, because there are additional tuning knobs in the network stack that can affect the rate at which NET_RX softirqs will fire…"

The file /proc/irq/IRQ_NUMBER/smp_affinity is used to set IRQ affinity to a specific CPU core.

Note: enabling RPS to distribute packet processing to CPUs which were previously not processing packets will cause the number of `NET_RX` softirqs to increase for that CPU, as well as the `si` or `sitime` in the CPU usage graph. You can compare before and after of your softirq and CPU usage graphs to confirm that RPS is configured properly to your liking.

The rxq_display.sh script can be used to check the current mapping between a NICs receive queue softIRQs and the CPU core that will process it.

Transmit Packet Steering (XPS)

Use XPS (Transmit Packet Steering) to assign a different CPU to each NIC TX queue by setting a bitmask in /sys/class/net/DEVICE_NAME/queues/QUEUE/xps_cpus. For example using eth0 and transmit queue 0, one would modify the file /sys/class/net/eth0/queues/tx-0/xps_cpus with a hexadecimal number indicating which CPUs should process transmit completions from eth0's transmit queue 0.

The txq_display.sh script can be used to check the current mapping between a NICs transmit queue softIRQs and the CPU core that will process it.

NIC Hardware Settings

It might be worth turning off some of the NIC hardware offload features for common traffic types to check for performance improvements/impairments (in case the NIC is trying to pro-actively interfere):

Disable rx-checksumming
`$ sudo ethtool -K eth0 rx off`

Disable tx-checksumming

$ sudo ethtool -K eth0 tx off

Disable tcp-segmentation-offload
$ sudo ethtool -K eth0 tso off

Disable udp-fragmentation-offload
$ sudo ethtool -K eth0 uso off

Disable generic-segmentation-offload
$ sudo ethtool -K eth0 gso off

Disable generic-receive-offload
$ sudo ethtool -K eth0 rso off

Disable rx-vlan-offload
$ sudo ethtool -K eth0 rxvlan off

Disable tx-vlan-offload
$ sudo ethtool -K eth0 txvlan off

Disable disable pause frames
$ sudo ethtool -A eth0 rx off tx off autoneg off

With some drivers such as the IXBGE driver for example, the cleanup of the Rx and Tx hardware buffers are both handled when the NET_RX softIRQ runs.

Disable adaptive interrupt coalescing (in production devices adaptive is probably wanted to better support a mixture of low latency and high throughput applications with the other settings providing the upper-bound limits):
$ sudo ethtool -C eth0 adaptive-rx off
$ sudo ethtool -C eth0 adaptive-tx off
This is the number of microseconds to wait before raising an RX interrupt after a packet has been received. When rx-usecs is set to 0 rx-frames is used. The default value "1" is some auto interrupt throttling:
$ sudo ethtool -C eth0 rx-usecs 30

This is the number of frames to queue up before raising an RX interrupt:
$ sudo ethtool -C eth0 rx-frames N

This is the number of microseconds to wait before raising an TX interrupt after a packet has been sent. When tx-usecs is set to 0 tx-frames is used:
$ sudo ethtool -C eth0 tx-usecs N

This is the number of frames to queue up before raising an TX interrupt:
$ sudo ethtool -C eth0 tx-frames N

The hardware buffer length can also be increased so that more frames are sent per NET_TX interrupt. The NIC fills its hardware buffer with frames; the buffer is then drained by the NET_RX softirq, which the NIC asserts via an interrupt. To interrogate the status of this queue, use the following command:
`$ ethtool -S eth0`

This will display how many frames have been dropped within eth0. Often, a drop occurs because the queue runs out of buffer space in which to store frames. To increase the hardware buffer length use:

`$ ethtool --set-ring eth0 tx 1024`
Or
`$ ethtool -G eth0 tx 1024`

Increasing the hardware queue size too much might induce bufferbloat. For this reason it may be best to set the rx-usecs value shown above to a low value causing the Tx queue to be regularly drained of “sent” packets.

Set the number of Tx/Rx queues, this has no/minimal benefit unless traffic is distributed across the queues and each queue is mapped to a separate CPU for processing:
`$ sudo ethtool -l eth0`
`$ sudo ethtool -L eth0 combined 8`
`$ sudo ethtool -L eth0 tx 8`

Set the size of the queue length:
`$ sudo ethtool -g eth0`
`$ sudo ethtool -G eth0 tx 4096`

Increasing the size of the Tx queue may not make a drastic difference because DQL is used to prevent higher layer networking code from queueing more data at times. Reducing the Tx queue size can have negative impacts if too small.

For example when testing with a default Tx queue size of 512 packets, using a single Tx queue and single CPU with 1500 byte frames, the Tx speed of EtherateMT 0.3.beta was a steady 6.7Gbps (circa 550k pps). NET_TX IRQs on that queue/CPU core were 7k p/s and NET_RX IRQs 5k p/s. The number of inflight packets (/sys/class/net/eth0/queues/tx-2/byte_queue_limits/) is fluctuating between 300k and 375k p/s.

`$ sudo ethtool -G eth0 tx 128`

This dropped the Tx rate to a steady 5.7Gbps (circa 470k pps). NET_TX IRQs on that queue/CPU core increased to 27k p/s and NET_RX IRQs increased to 18k p/s. The number of inflight packets (/sys/class/net/eth0/queues/tx-2/byte_queue_limits/) is fluctuating between 40k and 80k p/s.

Setting a larger queue size reduces the number of IRQs processed per second and slightly increased the throughput (although the latency will increase too) buy filling the queue with more packets/frames to be process in a single interrupt.

```
$ sudo ethtool -G eth0 rx 4096
$ sudo ethtool -G eth0 tx 4096
```

Applying the the above commands produced a steady 7.2Gbps (circa 595k pps) Tx rate with between 0 and 1 NET_TX IRQs per second and 5k NET_RX IRQs p/s. A point of note is that all the time the CPU core processing the NET_TX and NET_RX IRQs for this single NIC queue was running at 100% utilisation (whatever the queue size). The single CPU core running a single EtherateMT worker thread placed traffic into that single NIC TX queue wasn't running at 100% utilisation until the NIC queue size was increased to 576 or higher. A NIC queue size of 544 or lower left the EtherateMT worker thread core running at a sustained circa 45% utilisation. `$ sudo ethtool -G eth0 rx 576 && sudo ethtool -G eth0 tx 576` has a steady Tx rate of 7.3Gbps (600k pps) with 0 or 1 NET_TX IRQs p/s on average and 7k NET_RX IRQs p/s and an average inflight packet count for that NIC queue of 350k-420k.

^ This was all tested with Kernel 4.10. With 3.10 each scenario was about 1-1.5Gbps slower.

Linux Tuning

BIOS Settings

Disable cpu frequency scaling to make sure cpu runs at max speed (not power efficient!)
CPU Power and Performance Policy > Set to Performance
CPU C-state/P-state/C3-state/C6-state: Disabled
Enhanced Intel® Speedstep® Tech: Disabled
Turbo Boost: Disabled
MLC Streamer: Enabled
MLC Spatial Prefetcher: Enabled
DCU Data Prefetcher: Enabled
DCA: Enabled
Memory RAS and Performance Config -> NUMA optimized: Enabled

Hyperthreading should be disabled for zero packet loss or line rate transfers.

NIC Device weight

Once can increase the rate at which a NIC queue is drained. To do this, adjust the NIC's *device weight* accordingly. This attribute refers to the maximum number of frames that the NIC can receive before the softirq context has to yield the CPU and reschedule itself. It is controlled by the /proc/sys/net/core/dev_weight variable.

Kernel Tx Queue

The size of the Tx queue in the Kernel (the QDISC queue size, if QDISC bypass is not enabled) txqueuelen can be altered (increasing the QDISC length can help with NIC starvation). This is the number of *frames* in the queue!

```
$ sudo ifconfig eth0 txqueuelen 10000
```
Or
```
$ sudo ip link set txqueuelen 10000 dev eth0
```

^ This is having no effect with Etherate/EtherateMT because QDISC bypass is used.

The tx_inflight.sh script can be used to check the number of packets in the Tx queue.

Kernel Rx Queue

The kernel parameter “netdev_max_backlog” is the maximum size of the receive queue. The received frames will be stored in this queue after taking them from the ring buffer on the NIC. A higher value helps to prevent packet loss with high speed NICs but can add in latency.

To get the current value or change it:
```
$ cat /proc/sys/net/core/netdev_max_backlog
$ echo 250000 > /proc/sys/net/core/netdev_max_backlog
```

To permanently change the value add “net.core.netdev_max_backlog=NNN” to /etc/sysctl.conf

SELinux

On a system with SELinux enabled there is an auditing call for each syscall, disabling auditing (and/or) SELinux (if appropriate) can cut out some cycles-per-packet.

Note: Some guides explicitly recommend disabling SELinux. It is very likely to have a positive performance improvement however there are obvious security implications tied with that. One should disable SELinux at one's own risk.

Memory Tuning

Hugepages

“When a process uses some memory, the CPU is marking the RAM as used by that process. For efficiency, the CPU allocates RAM by chunks of 4K bytes (it's the default value on many platforms). Those chunks are named *pages*. Those pages can be swapped to disk, etc. Since the process address space is virtual, the CPU and the operating system have to remember which pages belong to which process, and where it is stored. Obviously, the more pages you have, the more time it takes to find where the memory is mapped. When a process uses 1GB of memory, that's 262144 entries to look up (1GB / 4K). If one Page Table Entry consume 8 bytes, that's 2MB (262144 * 8) to look-up.
Most current CPU architectures support bigger pages (so the CPU/OS have less entries to look-up), those are called *Hugepages* (on Linux), *Super Pages* (on BSD) or *Large Pages* (on Windows), but it all the same thing.”

“By using hugepage allocations, performance is increased since fewer pages are needed, and therefore less Translation Lookaside Buffers (TLBs, high speed translation caches), which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur with the standard 4k page size, slowing performance.”

“The TLB has a fixed number of slots. If a virtual address can be mapped to a physical address with information in the TLB, you avoid an expensive page table walk. But the TLB cannot cache mappings for all pages.
Therefore, if you use larger pages, that fixed number of virtual to physical mappings covers a greater overall address range, increasing the hit ratio of the TLB (which is a cached mapping).”

Note: Hugepages are not swapped.

Check for 2MB (pse) and 1GB (pdpe1gb) hugepage Intel x86 CPU flags:
```
$ grep -oE "pse |pdpe1gb" -m 1 /proc/cpuinfo
```

Check hugetable TLB is enabled in the current Kernel boot config:
```
$ grep CONFIG_HUGETLB /boot/config-4.11.0-041100-generic
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
```

Check hugepage usage:
```
$ grep Huge /proc/meminfo
AnonHugePages:       102400 kB
ShmemHugePages:           0 kB
HugePages_Total:     0
HugePages_Free:      0
```

```
HugePages_Rsvd:    0
HugePages_Surp:    0
Hugepagesize:      2048 kB
```

Hugepages are 2M in size by default. Allocate 2048 * 2M huge pages in real time (not possible with 1GB hugepages):

```
$ echo 'vm.nr_hugepages=2048' > /etc/sysctl.d/hugepages.conf
$ cat /etc/sysctl.d/hugepages.conf
```

or

```
$ echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
$ cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

then

```
$ sudo mount -t hugetlbfs none /dev/hugepages
```

With multiple NUMA nodes hugepage allocations are split between nodes, to make per-NUMA node allocations:

```
$ echo 1024 >
/sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
$ echo 1024 >
/sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
$ cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

To persist across a reboot add the following value to /etc/sysctl.conf:

```
vm.nr_hugepages=2048
```

To allocate 1G hugepages (can only be done during boot time):

In /etc/grub2-efi.cfg or /boot/grub2/grub.cfg or wherever, add to the Kernel GRUB_CMDLINE_LINUX value:

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

Then update Grub and reboot:

```
$ grub2-mkconfig -o /boot/grub2/grub.cfg
```

Or for UEFI systems:

```
$ sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg
$ reboot
```

Then check after reboot:

```
$ cat /proc/cmdline
```

To make the hugepages available to DPDK create a mount point:

```
$ mkdir /mnt/huge
$ mount -t hugetlbfs nodev /mnt/huge
```

Check:

```
$ grep Huge /proc/meminfo
```

To make the mount persistent across reboots add an entry to /etc/fstab (for 1GB pages, the page size must be specified as a mount option:):

```
nodev /mnt/huge_1GB hugetlbfs pagesize=1GB 0 0
```

Virtualisation/KVM/SR-IOV

IOMMU/VT-d/VT-x

A general rule at the time of writing is to try and use a recent Kernel (as recent as you can without compromising stability). NFV, SR-IOV, et al aren't highly mature technologies yet so many bug fixes are only present in newer Kernel versions. For example, some Intel cards had a limit of only 30 multicast addresses shared amongst all VFs until Kernel 4.4 when a patch introduced multicast promiscuous mode to overcome this.

Intel VT-x support. The CPU flag for VT-x capability is "vmx" which stands for Virtual Machine Extensions. These are CPU extensions which allow ring0 access to the CPU from inside the VM (whilst keeping the host protected). Check for it's presence with:
`$ grep -m 1 "vmx" /proc/cpuinfo`

Intel VT-d support: "VT-d" stands for "Intel Virtualization Technology for Directed I/O". An input/output memory management unit (IOMMU) allows guest virtual machines to directly use peripheral devices, such as Ethernet NICs, accelerated graphics cards, and hard-drive controllers, through DMA and interrupt remapping. This is sometimes called PCI passthrough. Intel call this VT-d support…In addition to the CPU support, both motherboard chipset and system firmware (BIOS or UEFI) need to fully support the IOMMU I/O virtualization functionality for it to be usable. Only the PCI or PCI Express devices supporting function level reset (FLR) can be virtualized this way, as it is required for reassigning various device functions between virtual machines.

DPDK uses a 1:1 mapping and does not support IOMMU. IOMMU allows for simpler VM physical address translation. The second role of IOMMU is to allow protection from unwanted memory access by an unsafe device that has DMA privileges. Unfortunately, the protection comes with an extremely high performance cost for high speed NICs.

Set IOMMU to pass-through (pt) which also disables IOMMU for the hypervisor and enable the Intel VT-d support in the Kernel by adding boot parameters"iommu=pt intel_iommu=on". The Kernel option "iommu=pt" disables IOMMU support for the hypervisor.

Edit /etc/default/grub, add "iommu=pt intel_iommu=on" to GRUB_CMDLINE_LINUX and update grub:
`$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg`
or
`$ sudo grub2-mkconfig -o /etc/grub2-efi.cfg`
Or for UEFI systems:
`$ sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg`

After a reboot, verify with:
`$ cat /proc/cmdline`

```
$ dmesg | grep -e DMAR -e IOMMU
[    0.000000] DMAR: IOMMU enabled
…
[    0.036269] DMAR: DRHD base: 0x000000fbffc000 flags: 0x1
…
[    0.647712] DMAR: Intel(R) Virtualization Technology for Directed I/O
```

SR-IOV

PCI-SIG Single Root I/O Virtualization (SR-IOV): The PCI-SIG or Peripheral Component Interconnect Special Interest Group is an electronics industry consortium responsible for specifying the Peripheral Component Interconnect, PCI-X, and PCI Express computer buses.

Note: These examples are from a system with an Intel X710 10G NIC that uses the i40e driver.

Check that the i40e module is loaded:

```
$ lsmod | grep i40e
```

Enable at least one VF on the PF (Linux 3.8 and newer don't use the "modprobe i40e max_vfs=N" format anymore):

```
$ sudo bash -c "echo 1 > /sys/class/net/eth0/device/sriov_numvfs"
$ cat /sys/class/net/eth0/device/sriov_numvfs
1
```

To persist across reboots add the following to /etc/rc.d/rc.local:

```
echo 1 > /sys/class/net/eth0/device/sriov_numvfs
```

Check now that the i40evf Kernel module is loaded in addition to the i40e module, if it wasn't already:

```
$ lsmod | grep i40e
```

Check that a VF was created:

```
$ dmesg | grep i40e
[ 1390.687700] i40evf 0000:09:02.0 eth0: NIC Link is Up 10000 Mbps Full Duplex

$ ip link
23: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT qlen 1000
```

Since these VFs are to be used by a VM in KVM and not by the host OS one must stop the host OS from "grabbing" the VFs when they are created. First unload the VF Kernel module:

```
$ sudo rmmod i40evf
```

Blacklist the VF Kernel module, by adding the following lines to /lib/modprobe.d/dist-blacklist.conf

```
# i40evf driver
blacklist i40evf
```

Find the VF in virsh output, it will have the PCI ID:

```
$  virsh --connect qemu:///system nodedev-list | grep 09
pci_0000_09_00_0
pci_0000_09_00_1
pci_0000_09_02_0
```

```
$ virsh --connect qemu:///system nodedev-dumpxml pci_0000_09_02_0
```

Add the following XML to VM definition in KVM to use the VF:

```
        <interface type='hostdev' managed='yes'>
        <source>
        <address type='pci' domain='0x0000' bus='0x09' slot='0x02' function='0x0'/>
        </source>
        </interface>
```

Or define a network pool using the PF in an XML file “passthrough.xml”:

```
<network>
  <name>passthrough_eth0</name>
  <forward mode='hostdev' managed='yes'>
   <pf dev='eth0'/>
 </forward>
</network>
```

And import that XML into libvirt:

```
$ virsh net-define /tmp/passthrough.xml
$ virsh net-autostart passthrough
$ virsh net-start passthrough.
```

Then when using a network pool, add the following to the guest instead for automatic VF assignment from the PF:

```
<interface type='network'>
 <source network=passthrough_eth0/>
 <!-- optional VLAN tag -->
   <vlan>
      <tag id='42'/>
    </vlan>
```

```
</interface>
```

Ensure that mac-spoofing is disabled for the VF:

```
$ ip link show dev eth0
$ ip link set eth0 vf 0 spoofchk off
```

Add this to /etc/rc.d/rc.local to make it persistent across reboots:

```
$ cat /etc/rc.local

PF="eth0"
VF_COUNT=4
echo $VF_COUNT > /sys/class/net/$PF/device/sriov_numvfs
for VF in $(seq 0 $(($VF_COUNT-1))); do ip link set $PF vf $VF spoofchk off; done

PF="ens2f1"
VF_COUNT=4
echo $VF_COUNT > /sys/class/net/$PF/device/sriov_numvfs
for VF in $(seq 0 $((VF_COUNT-1))); do ip link set $PF vf $VF spoofchk off; done
```

Qemu/Virtio

To pin the vCPUs of a running VM:

```
$ virsh vcpupin <vmname> <VCPU#> <PhyCPU#>
```

Appendix A

Cisco CSR1000v Notes

Note: It turns out Intel X710 NICs are supported until IOS-XE 16.7 (Polaris). Until then only Virtio, ixgbe and ixgbevf drivers are supported. This is confusing because this link (http://www.cisco.com/c/en/us/td/docs/routers/csr1000/release/notes/csr1000v_3Srn.html#pgfId-3394164) states “Beginning with Cisco IOS XE 3.12.1S...SR-IOV is supported for Citrix XenServer and KVM only” but one must look else where to find out that it is for older Intel 10GE NICs.

Release Notes (CSR1000v Denali):

DPDK for CSR: DPDK (Dataplane Development Kit) is integrated into the dataplane of the CSR 1000v, using poll-mode drivers.
http://www.cisco.com/c/en/us/td/docs/routers/csr1000/release/notes/xe-16/csr1000v-rel-notes-xe-16-3.html#99125

Release Notes (CSR1000v):

http://www.cisco.com/c/en/us/td/docs/routers/csr1000/release/notes/csr1000v_3Srn.html#pgfId-3467066
www.cisco.com/c/en/us/td/docs/routers/csr1000/software/configuration/b_CSR1000v_Configuration_Guide/b_CSR1000v_Configuration_Guide_chapter_00.html

Supported I/O Modes and Drivers

Beginning with Cisco IOS XE Releases 3.16S, the CSR supports several modes of communication between vNICs and the physical hardware:

- Para Virtual
- PCI Passthrough
- Single Root I/O Virtualization (SR-IOV)
- Cisco Virtual Machine Fabric Extender (VM-FEX)

For information, see the Cisco CSR 1000v Cloud Services Router Software Configuration Guide.

Cisco CSR 1000v vNIC Support for Cisco IOS XE 3.16S (with KVM)

NIC Types Supported:

- Virtio
- ixgbevf
- ixgbe (Intel 10Gb PCI Express NIC Driver)
- enic

Max. number of vNICs per VM instance: 26

vNIC Hot Add/Remove Support: Yes

SR-IOV: Yes (Requires the host hardware to support the Intel VT-d or AMD IOMMU specification. SR-IOV is not supported with Virtual LANs (VLANs)).

Driver Support for I/O Modes

Para Virtual:

- VMXNET3 (ESXi)
- Virtio (KVM)
- VIF-netfront (Xen)
- NetVSC (Hyper-V)

PCI Passthrough:

- ixgbe (for Intel 10 gig NIC)
- enic (for Cisco VIC)

SR-IOV:

- ixgbevf

VM-FEX (only applicable to Cisco VIC)

- ESXi DirectPath IO: VMXNET3
- PCI Passthrough: enic

I/O Mode Limitations

PCI passthrough (enic):

- Interoperability with another NIC: If enic is connected to other NIC (for example, Intel NIC) and then that NIC is used for other CSR VM (Para virtual or Passthrough), traffic will not pass through if enic is configured with VLAN.
- If a VLAN is configured, the other NIC receives a VLAN packet with VLAN id of 0.
- Jumbo packet support: In this release, jumbo packet (MTU > 1518) is not supported.
- CDP is not supported.
- HSRP standby cannot ping the HSRP group address

SR-IOV (ixgbevf):

- MTU change: (Intel limitation) First change the VF MTU on the host PF using the ip link set command. Then change the corresponding interface MTU on the VM. Otherwise, no traffic will pass. (Intel limitation)
- MAC address change: After changing the MAC address, it is necessary to change the MAC address of the VF on the host PF using the ip link set command. Otherwise, no traffic will pass. (Intel limitation.)

- Maximum VLANs: The maximum number of VLANs supported on PF is 64. Together, all VFs can have a total of 64 VLANs. (Intel limitation.)
- Maximum Multicast filtering: Intel VF supports registering a maximum of 30 multicast addresses. (Intel limitation.)
- Layer2 Learning: Intel SRIOV VF does not support promiscuous mode, so Layer 2 functionality, such as EVC, does not work. (Intel limitation.)

VM-FEX ESXi DirectPath IO (VMXNET3):

- VLAN is not supported in high-performance mode.

CSR1000v Requirements

Table 8. Minimum Server Resource Requirements per Cisco CSR 1000v Instance
http://www.cisco.com/c/en/us/products/collateral/routers/cloud-services-router-1000v-series/datasheet-c78-733443.html?cachemode=refresh

It might be possible to get a 10G APPX trial license?
http://www.cisco.com/c/en/us/td/docs/routers/csr1000/release/notes/csr1000v_3Srn.html#57415