TCP and UDP Tuning

This document describes some existing tunings for Linux based system to achieve high performance in 10G/100G interface.

Tuning for Linux

Some General Information

- Tuning for TX/RX buffer

```
# allow testing with buffers up to 128MB
sysctl -w net.core.rmem_max = 134217728
sysctl -w net.core.wmem_max = 134217728
# increase the length of the processor input queue
sysctl -w net.core.netdev_max_backlog = 250000
```

- Tuning for NIC

```
# increase txqueuelen for 10G NICS
ifconfig ethN txqueuelen 10000
```

- Zero Copy Socket
  - Uses system call sendfile, sendfile64 and splice.
  - Reduce memory copy from memory mapped file to soket buffer.
  - a golang implementation zsocket , which can benefit for sending file.
- **NUMA issue** : the core handling IRQs differs from the core processing the packet
  - Turn off irqbalance

```
# stop irqbalance service in CentOS
```

```
systemctl stop irqbalance.service
# prevent the service autostart on reboot
chkconfig --level 123456 irqbalance off
```

  - Setup IRQ affinity to a certain core (as in here)
    - performance Tunning tools can be download here

```
# set up affinity of interface to CPU node
/set_irq_affinity_bynode.sh 1 ethN
```

  - Bind your program to the same CPU socket with numactl

```
# bind the program to numa pool associted with cpu node 0
numactl -N 0 -m 0 program
# to know which core belong to NUMA socket node 0
cat /sys/devices/system/node/node0/cpulist
```

- CPU Governor: linux defaultly uses 'powersave' CPU covernor, change to 'performance'

```
# RHEL/CentOS systems:
cpupower frequency-set -g performance
# Debian/Ubuntu systems:
cpufreq-set -r -g performance
```

- **Traffic control**: Fair Queuing Scheduler and Packet Pacing
  Potentially reduces the congestion and oscillation of traffic rate due to inefficient congestion control.

  - Fair Queuing (noted that TSO should be disabled when using FQ), to enable per flow pacing.

```
# enable fair queueing (off by default) for interface ETH
tc qdisc add dev $ETH root fq
```

  - Pacing: Sends the traffic with a given rate

```
# enable packet pacing (to reduce rate variation) to Ngbit
tc qdisc add dev $ETH root fq maxrate Ngbit

# for older verson of kernel use HTB-based pacing
# clear out any existing tc rules
tc qdisc del dev eth0 root
# create a Hierarchical Token Bucket
tc qdisc add dev eth0 handle 1: root htb
# add a 'class' to our route queue with a rate of 900Mbps
tc class add dev eth0 parent 1: classid 1:1 htb rate 900mbit
# create a filter that restricts our tc queue and class to a spe
cific source subnet
tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip
 dst X.Y.Z.0/24 flowid 1:1
```

TCP Tuning

- TCP socket buffer: Small socket buffer would limit the throughput.

```
# increase Linux autotuning TCP buffer limit to 64MB
sysctl -w net.ipv4.tcp_rmem = 4096 87380 67108864
sysctl -w net.ipv4.tcp_wmem = 4096 65536 67108864
# increase the length of the processor input queue
sysctl -w net.core.netdev_max_backlog = 250000
# recommended default congestion control is htcp
sysctl -w net.ipv4.tcp_congestion_control=htcp
# recommended for hosts with jumbo frames enabled
sysctl -w net.ipv4.tcp_mtu_probing=1
```

- Congestion control algorithm: HTCP or BBR: Different congestion algorithms behave differently for start-up and recovery transmission, which affect the performance when congestion happens.
    - HTCP increase the aggressiveness of TCP on high bandwidth-delay product(BDP) paths
    - BBR (Bottleneck Bandwidth and RTT) probe the bandwidth and RTT to derive current delivery rate with pacing the packets with that rate.

```
# check available congestion control
sysctl net.ipv4.tcp_available_congestion_control
# add designed congestion control algorithm (htcp as an example)
modprobe tcp_htcp
# set congestion control algorithm (htcp as an example)
sysctl -w net.ipv4.tcp_congestion_control=htcp
```

- Tunning for TCP Socket options
  - TCP_NODELAY (disable Nagle's algorithm) send packet immediately, in golang enabled by default.
  - TCP_QUICKACK (enable quick ack, may not be permenent)

UDP Tuning

- Tuning for Socket buffer
  - Set for each UDP socket buffer (4M is usually enough)
    - SetReadBuffer(4*1024*1024)
    - SetWriteBuffer(4*1024*1024)
  - Tuning default socket buffer in Linux

```
# increase udp minimal read/write memory (13k as an example)
sysctl net.ipv4.udp_rmem_min=131072
sysctl net.ipv4.udp_wmem_min=131072
```

VM Tuning

- Xen Network Throughput and Performance Tuning