(VM) Virtual Machine

VM para-virtual Net (*virtIO*) [eth1]

- TXqueue
- Device Driver [eth1]
- VirtIO FRONTEND
- Pseudo functionality: (depends on implementation) LRO, TSO, RX/TX Checksum, VLAN
- VirtIO BACKEND
- KVM Hypervisor

VM virtual Net Card (*e1000*) [eth0]

- TXqueue
- Device Driver [eth0]
- NAPI (Depends on driver) Lower Layer of Driver
- TSO, LRO, TCP/UDP/IP Checksum, RSS, aRFS, VLAN, Ethernet Checksum

Backlog buffer, Net options buffer, GRO, RPS, RFS, GSO, Network scheduler (qdisc)

IP

Network Stack: UDP; TCP (Window scaling, Congestion Control, Timers, TCP queue & Limits)

Socket

Socket buffer

Application & Programs

QEMU *EMULATOR*

TAP (1), TAP (0), tap1, br0 or qbr65eeaf6e-7e, tap0 or tap65eeaf6e-7e

Application/programm

Socket buffer

Socket

Network Stack: UDP; TCP (Window scaling, Congestion Control, Timers, TCP queue & Limits)

IP

RFS, RPS, GRO, GSO

Network scheduler (qdisc)

(RX) Backlog buffers

(FIFO) TXqueuelen buffer

Device Driver [eg. eth0]

NAPI (Depends on driver support)

Lower Layer of drivers

NAPI Pooling

(FIFO) Ring buffers

Network card (NIC/LAN) [eth0]: TCP/UDP/IP Checksum, LRO, TSO, RSS, aRFS, VLAN, Ethernet Checksum

Sysctl variables

r = receive (RX)
w = write (TX)

`net.core.rmem_max` `net.core.rmem_default`
`net.core.wmem_max` `net.core.wmem_default`

TCP	UDP
`net.ipv4.tcp_mem`	`net.ipv4.udp_mem`
`net.ipv4.tcp_rmem`	`net.ipv4.udp_rmem_min`
`net.ipv4.tcp_wmem`	`net.ipv4.udp_wmem_min`

Scaling TCP window (ON/OFF) set in: `net.ipv4.tcp_window_scaling`

Congestion control algorithm in use are defined in: `net.ipv4.tcp_congestion_control`

TCP T*imers* are defined in:
`FIN_WAIT_2 :` `net.ipv4.tcp_fin_timeout`
`TIME_WAIT= 2 x FIN_WAIT_2`

Using connection in **Time_wait** state: `net.ipv4.tcp_tw_reuse`

Limiting TCP connections in state of oppenning: `net.core.somaxconn`

TCP Syn queue memory
-maximum number of connections in state of oppenning, for specific (each) port: `net.ipv4.tcp_max_syn_backlog`

Net statistics on this level of network can be seen in files: `/proc/softirqs` , `/proc/net/softnet_stat`, `/proc/net/dev` and `/proc/net/netstat` and `/proc/net/snmp`
Also with the next commands: `netstat -i` , `netstat -s` and `ss` also with command : `nstat`

Port range available for all network connections: `net.ipv4.ip_local_port_range`
Enabling of *routing* between network interfaces: `net.ipv4.ip_forward`
Usage of non local IP adress (for example for ***VRRP*** protocol): `net.ipv4.ip_nonlocal_bind` , . . .

Another functionalities on this layer of network (entirely or partially implemented in software):

- **Receive Packet Steering (RPS) -** to check and change: `/sys/class/net/eth0/queues/`
- **Transmit Packet Steering (XPS) -** to check and change (flow number are `XX`): `/sys/class/net/eth0/queues/tx-XX/xps_cpus`
- **Generic receive offload (GRO) -** for enabling: `ethtool -K eth0 gro on`
- **Generic segmentation offload (GSO) -** for enabling: `ethtool -K eth0 gso on`

Selected scheduler for all network interfaces is set in: `net.core.default_qdisc`
For list of all schedulers (eg.`codel`,`pfifo_fast`,`fq`,...) start command:
`grep '^CONFIG_NET_SCH' /boot/config-$(uname -r)`
For configuring only specific network interface; eg. `eth0` **for selecting** `fq` **scheduler:**
`tc qdisc add dev eth0 root fq`

Memory buffer fo options which are saved with network packet (eg. net interface, IP header [TTL], ...) : `net.core.optmem_max`

(RX) Buffer for all network interfaces on the system `net.core.netdev_max_backlog`

`TXquelen` **memory/buffer**
Possible to change the size (`XX`):
`ip link set` *`eth0`* `txqueuelen XX`

NAPI Pooling

Number of packet that can be stored to buffer before NAPI pooling. If we have enabled LRO and GRO than this is related to aggregated packets: `net.core.dev_weight`

Number of packets that can be pooled at once, for all network interfaces on the system: `net.core.netdev_budget`

Time in *μs* allowed for *pooling* of packages: `net.core.netdev_budget_usecs`

For network statistics on this level (for `eth0`) , we can use commands like:
`ip -s link` & `sar -n EDEV` & `ethtool -S eth0` **for lowest layer of network.**
You may watch statistics directly from files: `/proc/net/dev` **or** `/proc/net/netstat` **or** `/proc/net/snmp`

You can increase/decrease buffers up to some hardware size (for RX=`YY` and for TX= `ZZ`) with the next command:
`ethtool -G eth0 rx YY tx ZZ`

Hardware accelerated (if any) functions of the network cards:

Receive Side Scaling (RSS) - for increasing RX & TX flows to XX & YY: `ethtool -L eth0 rx XX tx YY`
Hardverski Receive Flow Steering (aRFS) - for enabling: `ethtool -K eth0 ntuple on`
Large receive offload (LRO) - for enabling: `ethtool -K eth0 lro on`
TCP segmentation offload (TSO) - for enabling: `ethtool -K eth0 tso on`
Checksum offload - for enabling for receive (RX) and send (TX): `ethtool -K eth0 rx on tx on`
Scatter-gather - for enabling : `ethtool -K eth0 sg on`
Hardware support for VLAN (802.1Q) za TX i RX : `ethtool -K eth0 rxvlan on txvlan on`

Net card setting, like speed and duplex; for example: 1000Mbps (1Gbps), full duplex:
`ethtool -s eth0 speed 1000 duplex full`

Optimization of interrupts (*IRQ*) through the CPU affinity (for IRQ Nr. XY) are setting in:
`/proc/irq/XY/smp_affinity`

Linux inside LXC container

- eth0, TX queue
- Cgroup namespace, UTS namespace, Proces & PID namespace, User namespace, Mount namespace
- VETH pâir 2 eth0@if13 (ID-2)
- Network namespace
- CGROUPS (Limits): RAM/CPU/DISK/NET

LXC container

VETH pâir 2 veth2@if12 (ID-2) — TXqueue

qvo65eeaf6e-7e, qvb65eeaf6e-7e

VETH pâir 1 qvoxxx(ID-1) — VETH Tunnel — VETH pâir 1 qvbxxx(ID-1) — TXqueue

Linux bridge (eg. br0 or vmbr0) — TXqueue

OVS Bridge (OVSBridge)

Or with VETH, Or with TAP

OpenvSwitch (OVS)

OpenStack Implementation

- qvoxxx (ID1) VLAN Tag xy
- br-int (Internal Bridge)
- patch-tun (port)
- patch-int (port)
- br-tun (Internal Bridge)
- VLAN Tag Xy (port) Connection to VLAN network
- VxLAN 1 (port) Tunnel to: VM (NOVA) XY
- VxLAN XY (port) Tunnel to: NET (Neutron) server XY

br-int (also visible directly from Linux)
patch-tun (internal `patch` interface)
patch-int (internal `patch` interface)
br-tun (also visible directly from Linux)
vlan-Xy (if used instead of `vxlan`) (internal `VLAN` interface)
vxlan-c0a8c864 (internal `vxlan` interface)
vxlan-XY (internal `vxlan` interface)

VLAN or VXLAN

VxLAN Tunnels

Physical NET interface(s) (eg. `eth0`) are with `TAP` interface connected with: Linux bridge interface (eg. `br0`) or OVS bridge interface !
With `VxLAN` tunnels (OVS) are interconnecting servers between themselves, by standard implementation of OpenStack. With `VLAN` servers are connected to isolated LAN.
All physical and logical network interface can be connected to Linux ewall.
`MTU` exists and can be changed on any network interface(s) on Linux !.

tap1 — Net interface name under Linux
tapXY — Net interface name in OpenStack under Linux