(VM) Virtual Machine

TXqueue
Device driver (virtio_net) → [eth1]
VirtIO FRONTEND
Pseudo functionality (depends on implementation)
LRO, TSO, RX/TX Checksum offload, VLAN, Multi-Queue, ...
T X
R X
VM Paravirtual network card (virtio_net)
TX
RX
VirtIO BACKEND
KVM (Hypervisor)
TXqueue
Device driver (e1000e) → [eth0]
NAPI (Depends on driver)
Low layer of device driver
TSO
TX
Ethernet Checksum
TCP/UDP/IP Checksum
VM virtual (emulated) network card (eg. Intel® 82540EM)
scheduler (qdisc)
Network
NetLink
Netfilter +Bridge,Bond,...
Backlog buffer
Net options buffer
GRO
RPS
RFS
IP
GSO
UDP
TCP
Window scaling
Congestion Control
Timers
TCP queue & Limits
Network stack
Socket
SKB
Socket buffer
System calls (SCI) and Netlink sockets
Applications
QEMU EMULATOR

Aplications

System calls (SCI) & *Netlink* sockets

`netstat -tunap` , `ss -anm`
→ Rows: `Recv-Q` i `Send-Q`

SKB (sk_buff)
Socket buffer

Sysctl variables

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

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

rmem (SKB)
wmem (SKB)
Linux Socket
Network stack
UDP
TCP
Window scaling
Congestion Control
Timers
TCP queue & Limits
GSO
IP
RFS
RPS
GRO
(RX) Backlog buffers
netfilter
NetLink
eBPF
TC (traffic control)
INGRESS
EGRESS
Network scheduler (qdisc)
TX queue
Device driver ixgbe → eth0
NAPI (depends on device driver)
Low layer device driver
(RX) XDP
Work with interrupt req. (IRQ)
eXpress Data Path (if supported)
TXquelen buffer
NAPI Pooling
(HW FIFO) Ring buffers

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`

TCP Window scaling (on/off) set in: `net.ipv4.tcp_window_scaling`
Selected algorithm for congestion control set in: `net.ipv4.tcp_congestion_control`
Timers defined in:
`FIN_WAIT_2` : `net.ipv4.tcp_fin_timeout`
`TIME_WAIT= 2 x FIN_WAIT_2`
Connections in Time_wait state reuse: `net.ipv4.tcp_tw_reuse`
Limiting total number of conn. in TCP opening state: `net.core.somaxconn`
TCP Syn queue buffer (Max number of half-open connections per port): `net.ipv4.tcp_max_syn_backlog`

Network statistics on this layer: `/proc/softirqs`, `/proc/net/softnet_stat`, `/proc/net/dev` and `/proc/net/netstat` or `/proc/net/snmp`
Visible with the next commands: `netstat -i`, `netstat -s` and `ss` and also with : `nstat` and `sar -n DEV`

Port range for all connections (per unique IP address): `net.ipv4.ip_local_port_range`
Routing between network interfaces defined here: `net.ipv4.ip_forward`
Enabling of non local IP binding (eg. for *VRRP* protocol): `net.ipv4.ip_nonlocal_bind` , . . .

Receive Packet Steering (RPS) - for check and change: `/sys/class/net/eth0/queues/`
Transmit Packet Steering (XPS) - for check and change (queue number 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`
Kernel 6.3+ BIG TCP (GRO i GSO) `ip link eth0 gro_max_size XY gso_max_size XY`

Buffer for options saved for every packet (eg. interface name, headers, errors, ...) : `net.core.optmem_max`

(RX) Backlog buffer for all network interfaces: `net.core.netdev_max_backlog`

Netfilter for packet filtering (ebtables, arptables, iptables, nf_tables, conntrack, logging, ...)
Bridge component
VLAN, VxLAN, VETH, MACVLAN, ...
Bonding/teaming component (eg. LACP)
`ifenslave bond0 eth0 eth1`
Statistics are visible in: `/proc/net/bonding/bondXY`

Active scheduler for all network interfaces (*sysctl*):
`net.core.default_qdisc`
For specific interface; eg. `eth0` for `fq` scheduler:
`tc qdisc add dev eth0 root fq`
For QDISC buffer - tc option `memory_limit`

Network scheduler (QDISC) i QDISC medumemorija (`tc` opcija `memory_limit`)

TXquelen (FIFO) buffer
Possible to change size (`XX`):
`ip link set eth0 txqueuelen XX`

NAPI Pooling
Number of packets that can be buffered before NAPI retrieval. If we have LRO and GRO turned on then it refers to aggregated/merged packages: `net.core.dev_weight`
Total number of packets that can be fetched at once, common to all network interfaces on the system: `net.core.netdev_budget`
Time in *µs* for *pooling* packages: `net.core.netdev_budget_usecs`

For network statistics on thiy layer (for `eth0`) , we can use commands like: `ip -s link`, `sar -n EDEV` also `ethtool -S eth0` for low layer of network.
Or watching directly from files: `/proc/net/dev` or `/proc/net/netstat` or `/proc/net/snmp`

Interrupt requests (*IRQ*) per CPU affinity (for IRQ number `XY`) are set up in: `/proc/irq/XY/smp_affinity` or with *Irqbalance* service (`/etc/default/irqbalance`)

It is possible to increase/decrease this buffer up to a certain hardware limit (for RX=*YY* and for TX=*ZZ*) with command: `ethtool -G eth0 rx YY tx ZZ`

Hardware accelerated functionality of network cards (eg. `eth0`), (if exists):

Receive Side Scaling (RSS) - increasing ***RX***/***TX*** queue(multiqueue) to ***XX***/***YY***: `ethtool -L eth0 rx XX tx YY`
Hardware 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 - enabling for receive (RX) and transmit (TX): `ethtool -K eth0 rx on tx on`
Scatter-gather - for enabling : `ethtool -K eth0 sg on`
Hardware support for VLAN (802.1Q) for TX and RX : `ethtool -K eth0 rxvlan on txvlan on`
Network card speed and duplex setting; eg. 1000Mbps (1Gbps), full duplex: `ethtool -s eth0 speed 1000 duplex full`

Network Interface (eg. Intel® i82599)
TCP/UDP/IP Checksum
LRO
RSS
TSO
aRFS
VLAN, VxLAN, GRE, ESP, ...
Ethernet Checksum
RX
TX

- Physical network interfaces (eg. `eth0`) are connected with `TAP`, `VETH` and other logical interfaces with Linux bridge (eg. `br0`,`vmbr`) or *OVS bridge* interface!
- `VLAN`-s are used for network isolation, while `VxLAN` tunnels (in OVS) usually interconnecting servers, by standard OpenStack implementation.
- All physical and logical Linux network interfaces can be connected to Linux firewall since they are passing through the *Netfilter* component.
- `MTU` size exists and can be changed on any network interface in Linux!

`tap1` Linux `TAP` interface name
`tapXYZ` *OpenStack* `TAP` interface name

License:
CC BY SA
Hrvoje Horvat
v.1.45
OID:

(LXC) container

Linux inside (LXC) container
eth0
Cgroup namespace
UTS namespace
Proces & PID namespace
User namespace
Mount namespace
VETH pair 2
veth4@if13
(ID-2)
Network namespace
CGROUPS (Limits): RAM/CPU/DISK/NET

Linux correlation:
X@if12→Y@if13
`cat /sys/class/net/veth3/if{index,link}`

VETH pair 2
veth3@if12
(ID-2)

br0 / vmbr0 or for OpenStack: qbr65eeaf6e-7e
tap0 or for OpenStack: tap65eeaf6e-7e
tap1
TAP [tap1]
TAP [tap0]
Linux bridge or OVS bridge

qvo65eeaf6e-7e
qvb65eeaf6e-7e
VETH pair 1
qvoxxx or veth2@if5
(ID-1)
VETH Tunnel
VETH pair 1
qvbxxx or veth1@if4
(ID-1)
Linux bridge
eg. br0 or vmbr0
OVS Bridge
eg. br-int or br-tun, vmbr0...

or VETH
or direct

Open vSwitch (OVS)

Buffers
vmbr0 (OVS Bridge)
(visible directly from Linux)
qvoxxx (ID1)
VLAN Tag xy
br-int (OVS Bridge)
(visible directly from Linux)
patch-tun (port)
patch-int (port)
br-tun (OVS Bridge)
VLAN xy or VxLAN xy (Internal Port)
VLAN xy (port) Connection to: VM (NOVA) server Xy
VxLAN x (port) Tunnel to: VM (NOVA) server Xy
VxLAN y (port) Tunnel to: NET (Neutron) server Xy
br-int (visible directly from Linux)
patch-tun (internal patch interface)
patch-int (internal patch interface)
br-tun (visible directly from Linux)
VLAN-Xy (internal VLAN interface)
vxlan-c0a8c864 (internal VxLAN interface)
vxlan-Xy (internal VxLAN interface)
PROXMOX Implementation
i / iii
OpenStack implementation
VLAN or VxLAN or direct connection