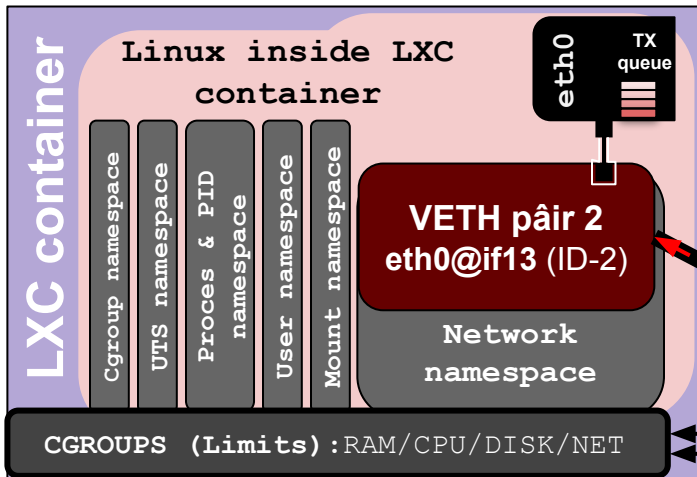
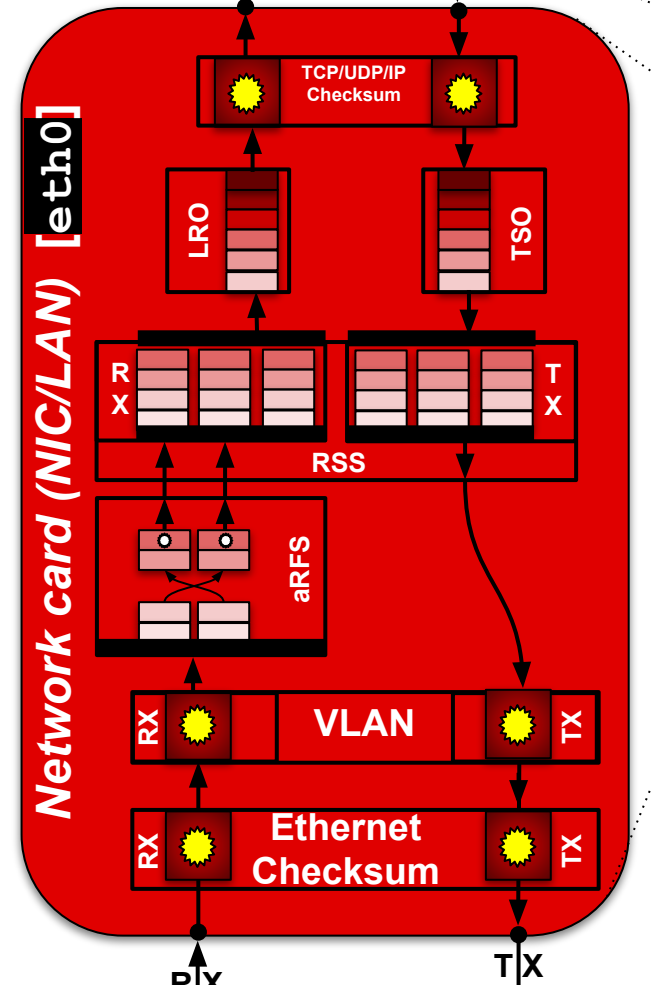
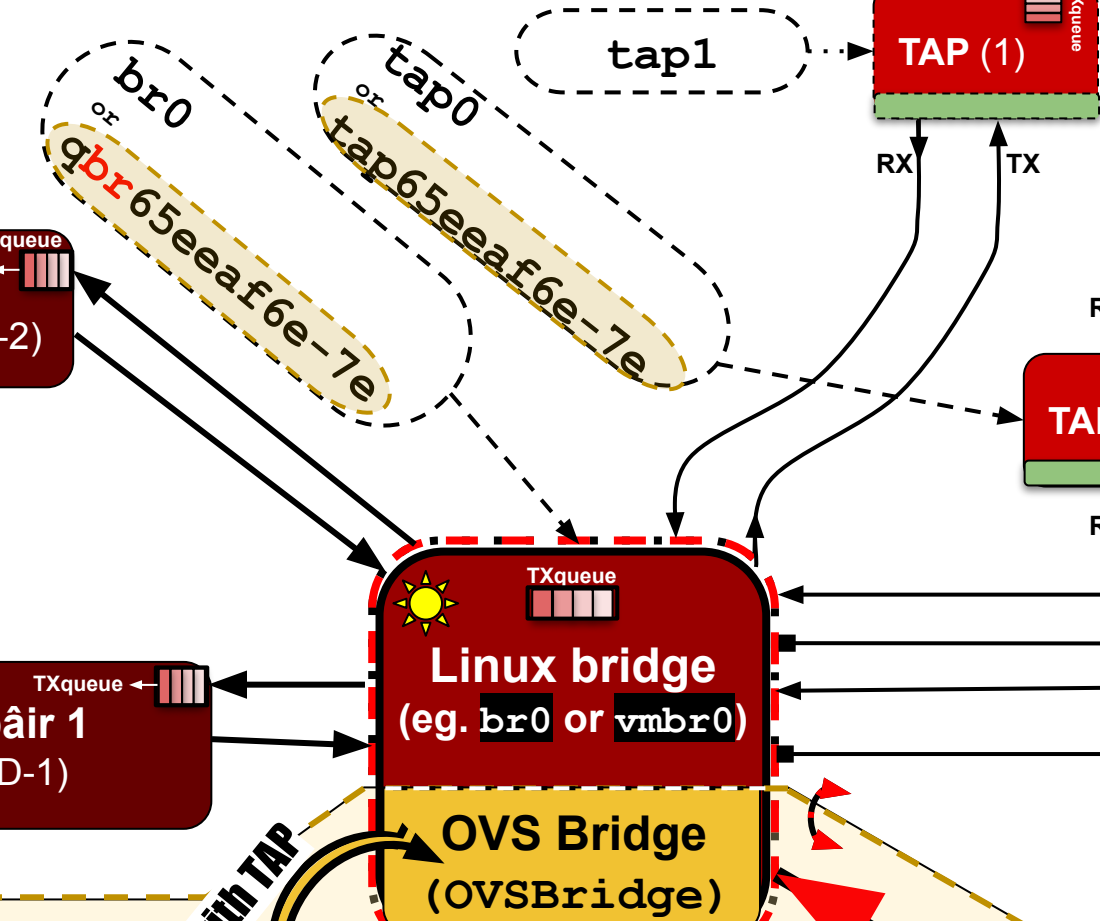
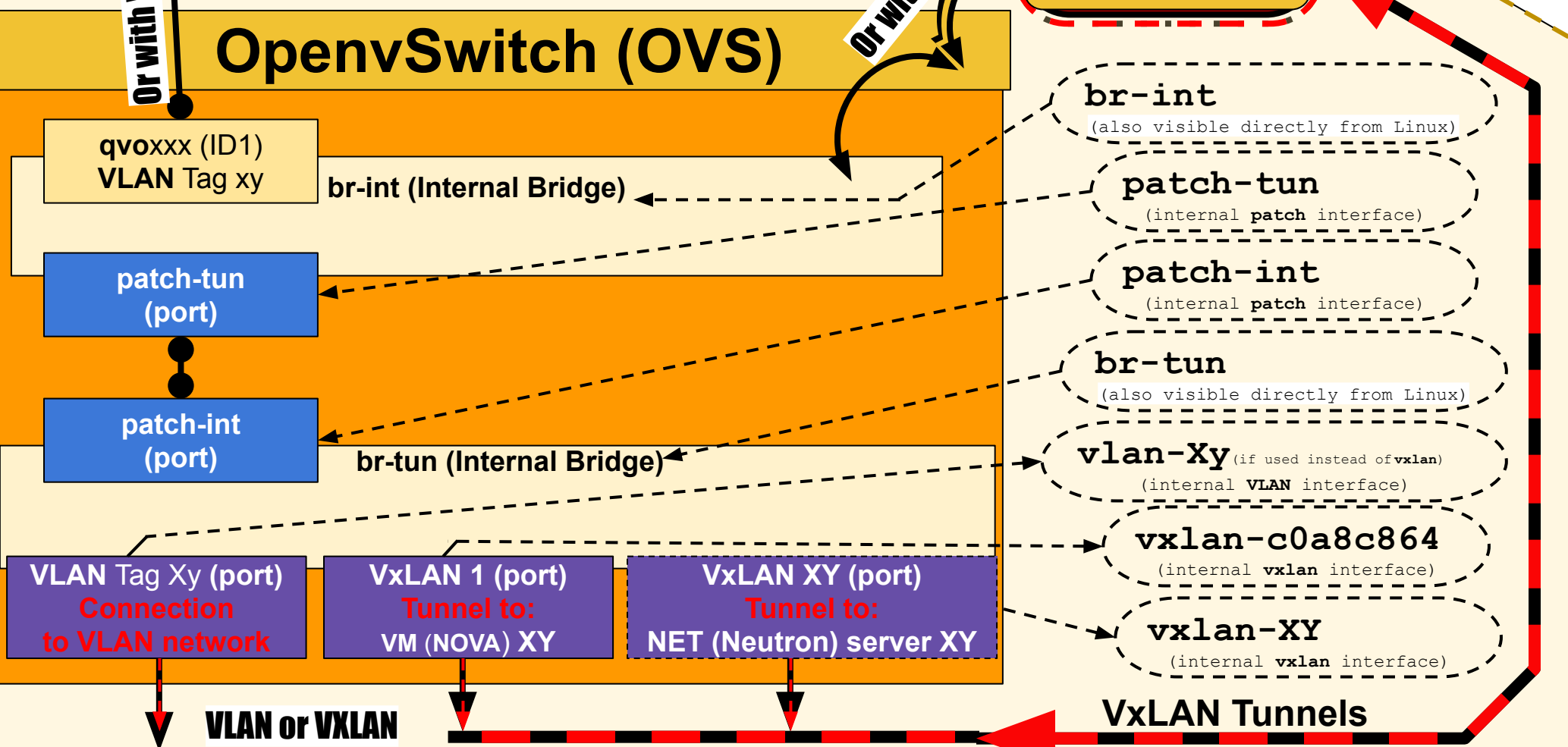


VM Virtual Machine



OpenStack Implementation



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`
- 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 - 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`

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`.

MRU 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

Socket Network Stack

- UDP
- TCP
 - Window scaling
 - Congestion Control
 - Timers
 - TCP queue & Limits
- IP
 - RFS
 - GSO
- Network scheduler (qdisc)
- Network scheduler (qdisc)

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 Timers are defined in: `net.ipv4.tcp_fin_timeout`

TIME_WAIT = 2 * FIN_WAIT_2

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

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

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

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 address (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. code1, 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 for 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`

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

NAPI Pooling

NAPI Pooling

Number of packet that can be stored to buffer before NAPI pooling. If we have enabled LRO and GRO that 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 μ s 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`
- 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 - 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`

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`.

MRU 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

Sysctl variables

`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

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`