HW High Availability and Link Aggregation for Eth switch, NIC RDMA and NIC SRIOV using Linux team/bonding

Or Gerlitz, Tzahi Oved

Mellanox
Ra'anana, Israel
ogerlitz@mellanox.com tzahio@mellanox.com

Abstract

The Linux kernel supports drivers for fast (10, 40 and 100Gbs) witch and NIC networking hardware from multiple vendors. These drivers implement the kernel software model for switch (switchdev) and NIC (netdev, roce device) interfaces, where the model includes certain offloading of traffic into HW. In many environments, it's common to require high-availability and link aggregation (LAG) for networking services. In this paper we show how to apply the kernel software LAG model on device drivers that actually offload traffic from the CPU (switch) or the kernel network stack (NIC). This concept is achieved in practice with the Mellanox mlxsw and mlx4 drivers in the upstream Linux kernel.

Keywords

Team, Bonding, LAG, offloads, switchdev, mlxsw, roce, sriov, mlx4.

Introduction

The Linux networking stack supports High-Availability (HA) and Link Aggregation (LAG) through usage of the bonding and team drivers, where both create a software netdevice on top of two or more netdevs. These LAG devices are set as master "upper" devices acting over "lower" devices. The core networking stack uses a notifier mechanism to announce setup/tear-down of such relations (referred to as linking / un-linking).

We show how to take advantage of standard bonding/team and their associated notifiers to reflect software LAG into hardware and achieve enhanced functionality.

We present few use cases dealing with physical switch, NIC RDMA and NIC SR-IOV Virtual Functions (VFs). In all cases, the creation of a bond/team LAG above Switch/NIC port netdevices is propagated to the device driver using network notifiers.

In the physical switch case, the device driver can either program the device to create the hardware LAG, or forbid the operation in case hardware resources were exceeded or because it lacks support for certain LAG parameters. The creation of further upper devices on top the LAG is propagated to the lower port netdevices in the same way as if the upper device was created directly on top of them.

In the NIC RDMA case, the device driver would create RoCE (RDMA-over-Ethernet) software device with only one port such that RDMA connections offloaded from the networking stack over this device are subject to LAG.

In the NIC SRIOV case, the virtual machine VF driver sees a HW device with only one port. The HW setup done by the host PF device driver causes the overall VF Ethernet traffic, both conventional TCP/IP that goes through the VMs networking stack and offloaded RDMA to be subject to HA LAG.

The paper begins with describing the Linux kernel LAG model, elaborating on the network notifiers and when/how they are invoked when LAGs are created/destroyed. Next, we briefly review the switchdev model for supporting HW switch ASIC and the Mellanox mlxsw driver which implements the switchdev model. Following is the use case of HW LAG based on SW bond/team over mlxsw switchdev port instances. After that, we provide a sketch of the kernel RDMA (Remote DMA) device model, RoCE (RDMA over Ethernet) transport and describe the Mellanox mlx4 NIC ASIC driver which supports both conventional Ethernet net-device and RoCE functionality. Following that are the use-case of HW LAG for RoCE traffic which is based on SW LAG done on mlx4 ports and the use-case of HW LAG for mlx4 SRIOV VFs (Virtual-Functions) which is based on SA LAG over mlx4 Physical function (PF) ports.

Other parameters to configure are methods for link integrity monitoring, transmit hash function, etc.

Linux LAG model

The Linux bonding and team drivers allow one to aggregate multiple network interfaces into a single logical "teamed" or "bonded" software interface. The detailed functionality and behavior of the grouped interface follows the setup mode. Basically, different modes provide high-availability (hot standby) or load balancing services. Other parameters to configure are methods for link integrity monitoring, transmit hash function, etc. Two specific modes of interest in our context are active-backup and LACP (802.3ad).

Under the active-backup mode only one of the underlying devices in the bond is active. A different device becomes active if, and only if, the active one fails. The bond's MAC address is externally visible on only one port (network adapter) to avoid confusing the switch.

The IEEE 802.3ad dynamic link aggregation mode creates aggregation groups that share the same speed and duplex settings and utilizes all the devices in the active aggregator according to the specification. Device selection for outgoing traffic is done according to the transmit hash policy, which may be based on different layers of outgoing packets: link (L2), network (L3) or transport (L4) and combinations of them.

Linux network notifiers

The kernel network notifiers allow sending notification to subscribed consumers in the networking stack on a change which is about to take place, or that just happened. The notification contains events type and affected parties. Depending on the notification type and data, the consumers can ignore the change, refuse the change, or conduct certain action based on the notification.

In the kernel networking stack terms, bonding and team devices are referred to as upper devices and the teamed devices as lower devices. The bond/team is an upper device for the teamed interfaces, and the teamed interfaces are lower device for the team/bond.

When a LAG (bond/team) is set, the notifications which are being sent to the lower devices are: pre-change upper (NETDEV_PRECHANGEUPPER) and change-upper (NETDEV_CHANGEUPPER). If a teamed device returns error when they get the pre change upper notification, the operation of adding that device to the team fails.

Linux Ethernet switch device driver model (switchdev)

The Ethernet switch device driver model (switchdev) is an in-kernel driver model for switch devices which offloads the forwarding (data) plane from the host CPU.

During switchdev driver initialization, the driver will allocate and register a net-device structure for each enumerated physical switch port, called the port netdev. A port netdev is the software representation of the physical port and provides a conduit for control traffic to/from the controller (the kernel) and the network, as well as an anchor point for higher level software constructs such as bridge, team/bond, VLAN, tunnels, and L3 router instances. Using standard netdev tools (iproute2, ethtool, etc), the port netdev can also provide to the user access to the physical properties of the switch port such as physical link state and I/O statistics.

In the switchdev model, the user is setting up a certain standard kernel software configuration (for example a bridge) over the port netdevs. This results in a hardware configuration that is derived from the software one which implements the required functionality in HW. In the bridge example, after the Linux bridge is set over the switchdev ports, FDB learning, forwarding and aging is offloaded to the HW.

For proper operation, although the fast path is offloaded to the HW, the ASIC is expected to be able to trap packets belonging to control protocols and send them towards the CPU. For example, proper bridge learning of multicast L2 entries (MDB) depends on trapping IGMP packets and have them received into the kernel from the port netdevs.

Mellanox switchdev driver (mlxsw)

The mlxsw driver is a switchdev driver for 10/25/40/50 and 100Gb/s Mellanox Spectrum Ethernet Switch ASIC. By version 4.6 of the upstream kernel, the driver implements bridge forwarding offload for 802.1q and 802.1d bridges which also includes vlan filtering (implied), and HW ageing. It also supports port splitter functionality and offloading bond/team LAG to HW which relates to the subject of this paper.

HW team/bonding with the mlxsw switchdev driver

In the switchdev model, when a team/bond is set over netdev ports that represent switch ports, the required result is a HW LAG set in the switch ASIC with these ports being members of that LAG.

The state-machine which is needed for proper operation of the 802.3ad LAG is not expected to be HW offloaded. This includes sending/receiving LACP packets and maintaining the LAG according to the spec. For that end, LACP packets received from peers should be trapped and sent towards the CPU, and on the other hand, the kernel should be able to xmit LACP packets over the switchdev ports exposed by the HW driver.

The mlxsw driver and Spectrum ASIC co-operate to implement this LAG offloading. The driver subscribes to networking notifiers in order to be able to accept / reject and properly act when the ports netdevs are added (deleted) to (from) team or bonding instance.

The user creates the software team/bond and starts adding mlxsw switchdev ports there. The team/bonding driver code first sends a pre change-upper notification to subscribed consumers. In this step, the callback installed by the switchdev driver is invoked, the driver checks the team mode and refuses the operation (return error) if they can’t offload it (for example when the team mode is not LAG). Next, the team driver continues with the normal flow of setting up a team instance. Once done, the team driver sends the change-upper notification. At this point in time, the switchdev driver checks if it has already created HW LAG which corresponds to this SW LAG, if not, it's created. Finally, the port is added to the HW LAG.

Linux RDMA over Ethernet (RoCE) and RDMA-CM model

The upstream Linux RDMA stack supports multiple transports: RoCE, Infiniband and iWARP, with RoCE standing for RDMA over Converged Ethernet.

RoCE HW supports Infiniband RC (IB Reliable Connection) transport over Ethernet links. RoCE V2 (upstream since 4.5) uses IBTA transport headers over UDP and hence being routable across IP subnets. RoCE traffic uses the IPv4/6 addresses which are set over the regular Ethernet NIC port netdev, as of other IP, but non RoCE traffic.

RoCE applications use the RDMA-CM (Connection Manager) API for control path and verbs API for data path.

Similarly to the switchdev case, the control path works through the kernel networking stack and the data-path is offloaded from the stack. However, unlike the switchdev case, the HW ASIC here is a NIC serving applications which do run on the local CPU. Hence, the data-path is offloaded from the stack but not from the CPU. The applications gain access to dedicated HW queues exposed by the RoCE HW driver and manage their data-path communication over these queues.

Establishing a RoCE session is made of three steps, which are implemented by the RDMA-CM:

1. Address resolution – does local route lookup and call ARP/ND services to resolve the local/remote IP addresses to local device/port, vlan and remote MAC address.
2. Route resolution – does path lookup (relevant only to IB networks).
3. Connection establishment – invokes the RoCE CM (Connection Manager) to wire the offloaded connection HW end-points, e.g. using three way hand-shake as defined in the IBTA spec.

The data-path verbs API has the following elements:

1. Post receive buffer – hand receive buffers to the NIC
2. Send/RDMA – send message or perform RDMA operation
3. Poll– poll for completion of Send/RDMA or Receive operation
4. Asynchronous completion handling and fd (for user-space applications) semantics are supported

The data-path API has also "slow" entries which are used by consumers to register application memory for direct HW DMA access (includes protection and translation), create HW end-points (Queue-Pairs) and completion queues. RoCE HW drivers expose RDMA device (struct ib_device) to the kernel. Device instances expose all the above data-path operations and have association with the NIC port netdevice. This device serves kernel and user-space RoCE applications, and user-space Raw Ethernet applications such as DPDK.

HW Bonding with the Mellanox mlx4 RoCE driver

The upstream mlx4 driver implements Ethernet and RoCE functionality for the Mellanox ConnectX3 NIC ASIC. Each port of the HW NIC is exposed as both Ethernet netdevice through the mlx4_en driver and as RoCE device through the mlx4_ib driver. The Ethernet port netdevice serves for plain Ethernet kernel networking and control pass for the RDMA stack (ARP/ND address resolution) as explained above.

In a similar manner to the mlxsw case, when a SW LAG (bond) is set over two netdevices corresponding to two ports of the same ConnectX NIC, we would like to achieve LAG functionality for RDMA sessions offloaded from the network stack as that functionality applies to plain TCP/IP sessions which are not offloaded.

To do that, network notifiers are being registered by the mlx4 Ethernet driver, which act on the NETDEV_BONDING_INFO event. This is indeed a bit less elegant/generic versus the mlxsw LAG offload which works over both team and bonding. Once the driver realizes that a LAG of type active-backup or LACP is set over two mlx4 ports, the RoCE devices that correspond to these ports are destroyed and a new RoCE device which logically corresponds to the LAG is created. The new RoCE device has only one port which is virtual. RoCE sessions created over this device enjoy from HW LAG functionality as they were TCP sessions established over the SW LAG.

It's common for transmit hash policies used by SW LAG solutions to make sure packets belonging to the same TCP session would be transmitted over the same port of the LAG. Here we set for each RoCE session its base port over which the session will run as long as the Ethernet port is active in the LAG. If this port stops to be active, such sessions fail-over to run over the other port, and when it's active again, they fail-back to their base-port.

SRIOV primer

Traditional hypervisors expose emulated or para-virtual devices to guest virtual machines and multiplexes the I/O requests of the guests onto the real hardware. More recently, there has been an effort to offload those tasks to I/O devices themselves. SR-IOV is a specification by PCI-SIG that allows a single physical device to expose multiple virtual devices. Those virtual devices can be safely assigned to guest virtual machine giving them direct access to the hardware. Using hardware directly reduced the CPU load on the hypervisor and usually results in better performance and lower latency.

Under SRIOV the physical function (PF) is the primary access point for control and management which includes creation of virtual functions (VFs) and assignment of various parameters/policies to be either exported to the VF or enforced on the VF such as default MAC address, default vlan and priority, max/min rate for VF traffic, virtual link state and spoof check policy.

HW Bonding with the Mellanox mlx4 SRIOV PF driver

The Mellanox mlx4 driver supports SRIOV. In the ConnectX architecture, the VF device driver control path goes through the PF driver using a dedicated communication channel, where the fast data-path (post send/receive buffers, start DMA ops, post HW door-bells and poll for copletion), goes directly to the HW, as expected under SRIOV.

This proxying nature of the control path allows the PF driver to provision certain aspects to VF devices in a non 1:1 manner versus the HW properties. One of these elements is the number of ports for VF devices. Under a configuration directive, the PF would act such that the VF driver sees a virtual HW device with one port, even though the HW/PF device has two ports and two netdevice instances running over them.

When such single ported VF is exported to VM and the PF Ethernet interfaces are bonded, the VM stack and applications running over the VF enjoy from HW LAG functionality transparently.

Note that this applies to all VM traffic that goes through the VF device: plain VM kernel Ethernet TCP/IP, kernel and user-space RoCE and user-space Ethernet (DPDK).

Conclusion

The Linux kernel network stack supports setting SW LAG that provide high-availability and load-balancing for applications through either simple active-backup policy or sophisticated link aggregation methods. We have shown how to offload a standard SW LAG built over HW device driver ports into a HW LAG for three use cases: physical switch (switchdev) driver, NIC RoCE driver and NIC SRIOV VF driver. In all the cases, no further configuration has to be carried out in end-nodes or applications that run over the LAG devices.

Acknowledgements

I'd like to thank Ido Schimmel for his assistance, support and fun minutes while preparing the presentation and the paper. Thanks to Jack Morgenstein, Moni Shoua and Rony Efraim for the joint work we did on the design, development and upstream submissions for the mlx4 RoCE and VF LAG offload solution. Thanks to Jiri Pirko for his inspiration, push and dedication for upstream way of developing drivers to Mellanox switch/NIC hardware and specially for aligning the design of the upcoming HW SRIOV LAG in the mlx5 driver to follow proper NIC e-switch model.

Bibliography

1. Feldman S, Pirko J - Switchdev documentation – linux kernel source tree: Documentation/networking/switchdev.txt

2. Lesokhin I, Eran H, Gerlitz O - "Flow-based tunneling for SR-IOV using switchdev API", netdev 1.1

3. Pirko J, Ido S, Elad R – "mlxsw: spectrum: Add initial support for Spectrum ASIC" upstream kernel commit 56ade8fe3fe1

4. Pirko J - "Merge branch 'bonding-team-offload'" upstream kernel commit c5b8b34c3f415

5. Pirko J, "Hardware switches - the open-source approach" netdev 0.1, http://people. netfilter.org/pablo/netdev0.1/papers/Hardware-switches-the-open-source-approach.pdf

6. Pirko J, libteam: team netdevice library https://github.com/jpirko/libteam

7. Bonding driver documentation - linux kernel source tree: Documentation/networking/bonding.txt

8. Shoua M, Gerlitz O - "IB/mlx4: Load balance ports in port aggregation mode" upstream commit c6215745b66

9. Shoua M, Morgenstein J, Gerlitz O - "net/mlx4_core: Support the HA mode for SRIOV VFs too" upstream kernel commit e57968a10bc

Author(s) Biography(ies)

Or Gerlitz is a Linux kernel developer dealing with networking, RDMA and storage. He is the co-maintainer of the upstream iSCSI RDMA (iSER) initiator driver and highly involved in the Mellanox upstream NIC drivers (mlx4 and mlx5).

Tzahi Oved is software architect dealing with software interfaces to the Mellanox NIC ASIC product line.