Scaling Linux bridge forwarding database

Roopa Prabhu, Nikolay Aleksandrov

Cumulus Networks,
Mountain View, CA, USA,
roopa@cumulusnetworks.com, nikolay@cumulusnetworks.com

Abstract

Linux bridge is deployed on Hosts, Hypervisors, Container OS's and in most recent years on data center switches. It is complete in its feature set with forwarding, learning, proxy and snooping functions. It can bridge layer-2 domains between VM's, Containers, Racks, POD's and between data centers as seen with Ethernet-Virtual Private networks [1]. With Linux bridge deployments moving up the rack, it is now bridging larger layer-2 domains bringing in scale challenges. The bridge forwarding database can scale to thousands of entries on a data center switch with hardware acceleration support.

In this paper we discuss performance and operational challenges with large scale bridge forwarding database and solutions to address them. We will discuss solutions like FDB dst port failover for faster convergence and faster API for FDB updates from control plane.

Most solutions though discussed around the below deployment scenarios, are generic and can be applied to all bridge use-cases:

– Multi-chassis link aggregation scenarios where Linux bridge is part of the active-active switch redundancy solution

– Ethernet VPN solutions where Linux bridge forwarding database is extended to reach Layer-2 domains over a network overlay like VxLAN.

Keywords

Netlink, Linux bridge, iproute2, E-VPN, VTEP

Introduction

A bridge is a networking device that creates a single aggregate network by connecting multiple network segments. This function is called bridging. Bridges use a table called forwarding database (FDB) to forward frames between network segments. A FDB entry carries a mac, vlan and a destination port. By default a bridge builds its FDB via flood and learn. Bridge FDB can also be populated via a controller or a control plane protocol.

Linux bridge [2] is a software bridge providing bridging function in the Linux networking stack. It can bridge between multiple network devices or interfaces known as bridge ports. Any network device can be a bridge port. The Linux bridge can bridge, learn, snoop, initiate and terminate tunnels. There are two variants of the Linux bridge: default non-vlan filtering bridge and the other more recent vlan filtering bridge. In this paper, we will focus on the vlan filtering bridge.

The Linux bridge forwarding database supports Netlink based API (RTM_*NEIGH) to add/delete/update/refresh a bridge fdb entry [3]. The Linux bridge also supports an API to add hardware learnt entries to the bridge forwarding database [4]. NTF_USE flag can be used by control plane in user-space to refresh a bridge fdb entry [5].

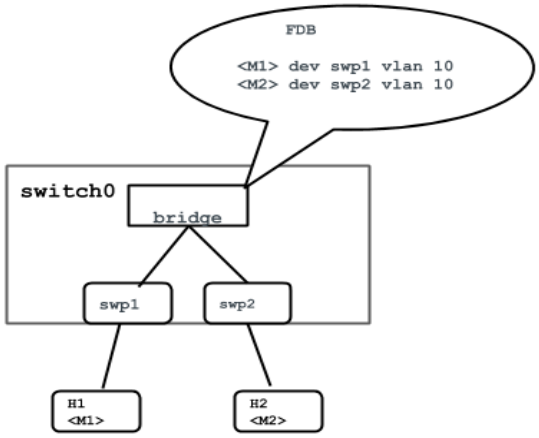

Fig1: Linux bridge on a switch bridging traffic between hosts

Fig1 shows hosts H1 and H2 connected to a switch running Linux bridge. Switch ports connected to hosts are in the bridge. Bridge FDB is indexed by <MAC, VLAN> and points to a destination (dst) port.

Linux Bridge and Network Virtualization

Network Virtualization

Network virtualization combines physical networks into a single virtual network. It comprises of the underlying physical network (underlay) and a virtual network (overlay) that is built on top of the underlay network. This paper focuses on the use of VxLan as the network virtualization technology. All references to Network overlay and network virtualization are VxLan. Network virtualization endpoints (NVE) initiate and terminate a network virtualization overlay. We will refer to VxLan tunnel end point as VTEP. Linux kernel has VxLan support and can be deployed as a

VxLan tunnel end point (VTEP). A VTEP can run completely in software or run on a switch chip with hardware acceleration. A first hop switch configured as a VTEP maps connected end devices (or hosts) to VxLan segments.

The Linux bridge maps traffic between the local endpoint Vlan to a VxLan segment. As shown in the fig below, this is achieved by two interfaces: a local interface connecting the end point device and a VxLan device enslaved in a bridge. The Vlan to VxLan mapping is achieved by configuring Vlans on the local port and on the VxLan device. To scale this to large number of VxLan endpoints one can use a VxLan single device with a vlan-vni map [1].

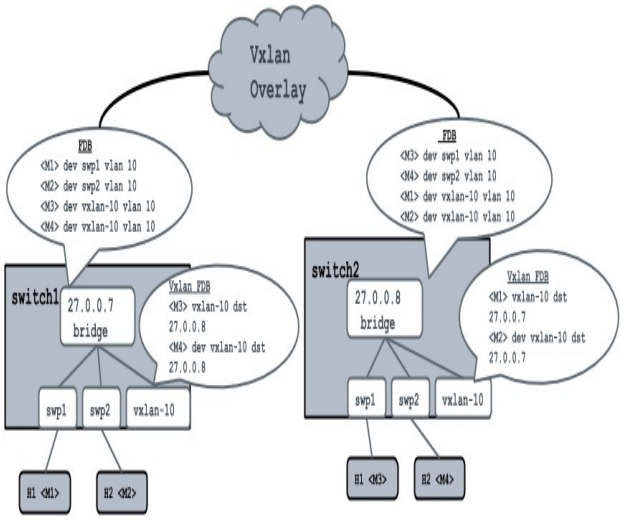

Fig 2. Bridging and Network virtualization

switch1 and switch2 are VTEPs connected by an overlay. vxLan0 is a bridge port. Vxlan FDB is an extension of the bridge FDB. Local macs in the bridge FDB point to non-VxLan ports. Remote macs reachable over the overlay point to VxLan ports. For every bridge mac pointing to the VxLan device, there is a corresponding mac in VxLan FDB with remote destination (remote VTEP) information. A all-zero default entry in the VxLan FDB can point to multiple VTEP's for broadcast-unknown-unicast-multicast (BUM) traffic. Such a FDB entry has <MAC, VNI> pointing to multiple remote destinations. Vxlan Driver replicates a BUM packet to each of these remote dsts.

Network virtualization is normally assisted by a control plane. It can be a centralized or distributed control plane. E-VPN [6] is the new and popular control plane for network virtualization in the data-center. The internet routing protocol BGP with a new "TYPE" is used as a control plane protocol. In EVPN, mac address reachability is learned and distributed by BGP control plane similar to routes.

On Linux, a BGP daemon with E-VPN support like FRR [7], learns local bridge FDB entries and distributes it to other E-VPN BGP Peers. Remote bridge FDB entries received via remote E-VPN BGP peers are installed in the kernel bridge FDB.

Bridge FDB scale

Linux bridge on a data center switch bridges layer 2 networks across racks or data-center pods or in some cases across data centers. With hardware acceleration, Linux bridge can see more than 150k FDB entries on some systems. These FDB entries in the most basic case are populated via flood and learn. In the network virtualization use case, bridging domains are further extended via overlays and distributed via controllers. We will see in future sections how multi-homing solutions multiply this scale by distributing FDB across multi-homed switches. We look at scaling bridge forwarding database with various parameters:

Learning at scale: Linux Bridge driver is capable of learning at scale. New updates to bridge FDB to use rhashtables [8] has allowed bridge FDB to scale close to a million entries. The external learn bridge FDB API [4] allows in-kernel hardware drivers to update bridge FDB with hardware learnt mac addresses at scale. Netlink batching mechanisms can be used for control plane MAC learning at scale.

Optimized broadcast-multicast-unknown unicast (BUM) Handling: A common problem with larger layer 2 bridging domain is flooding of (BUM) traffic at scale. The bridge driver provides multiple knobs to selectively turn on BUM handling on a per bridge port basis to optimize your network. In bridging with network virtualization case (eg VxLan), multicast tunnels can be used to reduce flood traffic across the overlays. Most network virtualization deployments use control plane or controller orchestrated mac distribution to reduce or eliminate flooding. Further, Linux bridge driver supports ARP and ND proxying to reduce flooding [9]. Future work will address moving bridge multicast database (MDB) [10] to a rhashtable and IGMP MLD proxy to reduce multicast protocol traffic across the overlay [18].

Network recovery and convergence on link failure: Bridge driver flushes dynamically learnt FDB entries on failed links. These flushed entries can be restored on link repairs by normal flood and learn procedures or by control-plane. At scale, this can result in larger downtime, churn and slower network convergence. Recent bridge backup port work [11] helps alleviate this issue. This work allows system administrator or control plane to program a backup bridge port for every bridge port at provisioning time. When such config is present, bridge driver will not flush bridge FDB entries and seamlessly re-direct traffic to the back-up path if the primary path is down. And restore traffic to the primary path when it comes back up. This completely eliminates control plane intervention during failure recovery leading to faster network convergence.

Network convergence on MAC moves: Mac moves are a result of an end-point moving to another host or to another port thereby changing the FDB entry destination for that mac. If left on its

own, the FDB entries age out and the network converges eventually. But, on data center switches these FDB aging timers can be more than 5mins leading to longer network outages by black-holing traffic. In the network virtualization case, due to the nature of stacked FDB between bridge and VxLan driver, there is a chance the stacked FDB's can go out of sync. We show this with an example: In fig 2, if a remote host H2 moves local to switch1, the bridge FDB entry on switch1 is updated to point to a local port. Bridge driver has detected this move due to a packet from H2. But the VxLan FDB entry can still exist pointing to a remote VTEP switch2 until it ages out. There are multiple solutions to make make these mac moves converge faster (a) Control plane detects the move (via bridge driver async FDB update netlink notifications) and intervenes to update the FDB to point to the right destination port faster. It can use the bridge FDB replace Netlink API to update the FDB entry. (b) Bridge driver can proactively call into the overlay (VxLan) port to update its FDB (delete stale FDB entry in the earlier example).

Bridging and Multihoming

Multihoming is the practice of connecting host or a network to more than one network (device) to increase reliability and performance. The network switches providing multihoming solution run a protocol between them to ensure redundancy and failure recovery for the connected end-points or networks. In this paper we will look at such switches as a "cluster of switches running Linux" providing redundancy and greater system throughput to multi-homed end-points or networks.

Multihoming with Dedicated Link

Multihoming with a dedicated link, also called Multi-Chassis Link Aggregation (MLAG), enables a server or switch with a two-port bond, such as a link aggregation group/LAG, EtherChannel, port group or trunk, to connect those ports to different switches and operate as if they are connected to a single logical switch.

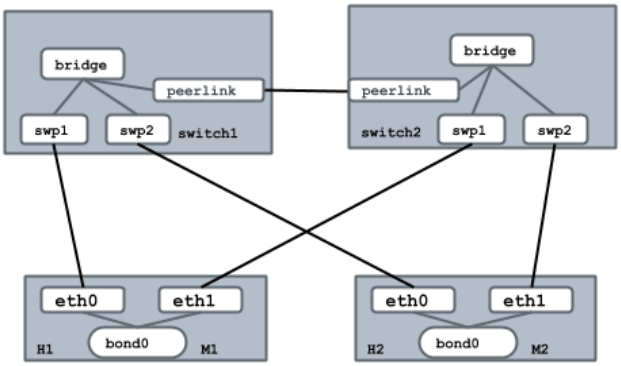

Fig 3: Multi-homing with dedicated peerlink

Fig 3 shows a typical multi-homed network with a dedicated link in between the switches. Host H1 and H2 are multi-homed to switches switch1 and switch2. The dedicated link (peerlink) in most cases is a bond for redundancy. Peerlink provides failover path on failure of a direct path to the end host. The multihoming protocol running on switch1 and switch2 keep the bridge FDB and MDB [10] in sync between the peers. This protocol also learns dual and singly connected end points from FDB. This involves keeping FDB alive and refreshed after they age out. Mac's learnt by the peer on a singly connected host are installed in the bridge FDB pointing to the peerlink.

Failure handling: In the event of a link failure, the MLAG control plane re-programs the bridge FDB to point MAC's on the failed link to peerlink. For faster network convergence, this re-programming of the FDB has to be as fast as possible. Dedicated peerlink interface in such a multihoming deployment is a static physical link between the peer switches and has to be configured at provisioning time. It serves as a backup link for all dually connected ports. For faster convergence of network traffic, bridge driver can implicitly re-direct traffic to the peerlink in the event of a primary link failure. The patch in [11] does just that. It adds a per bridge port flag (IFLA_BRPORT_BACKUP Netlink attribute) to indicate a backup port. The MLAG control plane can set this flag on dually connected links. This ensures minimal downtime on Link failures.

Multihoming with network overlay

With new BGP control plane for network virtualization [6], network overlay can be used to replace the dedicated link in traditional multihoming solution [13]. This scheme allows for multiple redundant paths for multi-homed hosts. Eliminating a dedicated peer switch link, ability to have more than one redundant path and a open standards based control plane have been the primary factors for popularity of such a new multi-homing solution.

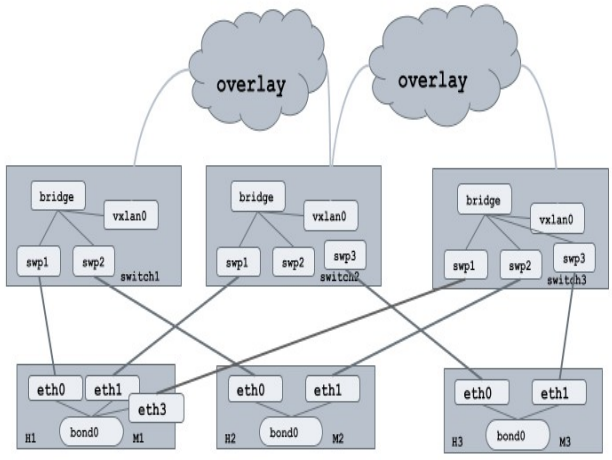

Fig 4: Multihoming with network overlay

Fig4 shows host H1 multi-homed to three switches switch1, switch2 and swicth3. Host H2 multi-homed to switch1 and switch3 and Host H3 multi-homed to switch2 and switch3. Network overlay ports (vxlan0) provide backup paths across the overlay in the event of a local path failure.

BGP EVPN control plane is capable of advertising a notion of an ethernet segment (ES) via ES routes. All redundant links to a multi-homed end-point belong to the same ES. This is key in determining multi homed links in the distributed BGP E-VPN multihoming control plane.

On Linux, as we have seen, the VxLan FDB is an extension to the bridge FDB. For E-VPN multihoming, the VxLan FDB entry for a end-point MAC points to list of remote virtualization end-points (VTEP's) providing multi-homing function to the end-point. In a active-active multi-homed solution, this list of remote VTEP's provide redundant paths and hence such a FDB entry must be treated as a Equal cost multi-path group (ECMP) similar to route ECMP groups.

Failure Handling: Failover to backup network ports (VxLan) on local bridge port failure works similar to the dedicated peerlink multihoming case as seen earlier in this paper [11]. In this scheme, VxLan bridge ports are configured as backup bridge ports by the E-VPN multihoming control plane. Upon a failure in connectivity to an attached ES, the E-VPN multihoming control plane withdraws its advertised ES routes from peer E-VPN instances. The E-VPN RFC [13] describes several other optimizations for faster convergence [14]. In scaled environments, the network convergence time is a function of number of mac advertisements that must be withdrawn on failures. E-VPN control plane optimizations like designated forwarder [15], split horizon checks [16] and aliasing [17] are used to prevent loops, reduce BUM flooding and reduce FDB updates.

To reflect these optimizations in the forwarding plane (eg VxLan FDB), we will need a scalable FDB update API to assist control plane in faster updates to the FDB entries pointing to remote network virtualization ports. In other words, we need a scalable FDB API to update overlay VxLan FDB remote destinations.

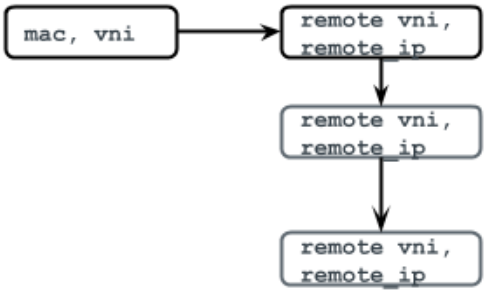

Fig 5. Current Vxlan FDB structure

As seen earlier in this paper, a VxLan FDB entry has <MAC, VNI> pointing to a list of remote destinations (VTEPs). Lets call the list of remote destinations “dst groups”.

Scalable FDB dst group update API: Goal is to have an API with faster FDB dst group update mechanism. This leads us to think of mac FDB entries as routes. FDB entries are macs pointing to dst groups. Dst groups can have a single dst, multiple dsts for replication or multiple dsts for ECMP. We propose a separate dst group database indexed by dst group id. FDB entries carry dst group ids.

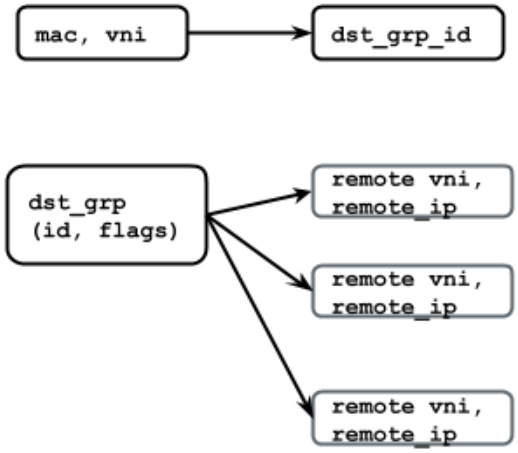

Fig 6: New VxLAN FDB structure

In the vxlan case, dst = rdst or remote dst.

Considerations for the new API: Extend AF_BRIDGE RTM_NEIGH* api with NDA_DST_GRP_ID attribute.

New Netlink API to create/delete/update dsts and dst groups: RTM_NEWDST, RTM_DELDST, RTM_GETDST

```
enum {
    NDA_DST_GROUP_UNSPEC,
    NDA_DST_GROUP_ID,
    NDA_DST_GROUP_FLAGS,
    NDA_DST_GROUP_ENTRY,
    __NDA_DST_GROUP_MAX,
};

#define                         NDA_DST_GROUP_MAX
(__NDA_DST_GROUP_MAX – 1)

enum {
     NDA_DST_UNSPEC,
     NDA_DST_IP,
```

```
    NDA_DST_VNI,
    NDA_DST_PORT,
    __NDA_DST_MAX,
}
#define NDA_DST_MAX (__NDA_DST_MAX – 1)

/* dst group flags */
#define NTF_DST_GROUP_REPLICATION 0x01
#define NTF_DST_GROUP_ECMP 0x02
```

We are also considering re-use of the new route nexthop API if feasible [18]

Acknowledgements

We would like to thank Wilson Kok, Anuradha Karuppiah, Vivek Venkataraman, and Balki Ramakrishnan for discussions, knowledge and requirements for building better Multihoming solutions on Linux.

Conclusions

In this paper we discussed scale challenges and solutions with Linux bridge in network virtualization and multi-homed environments. We showed how recent work has improved bridge learning and failure handling at scale. We categorized scale challenges based on learning, network convergence on failures mac moves, flooding and other factors at scale. We also suggested solutions to help build faster scalable multi-homing and network virtualization solutions with switches running Linux.

References

1. Linux bridge, Layer-2 gateways and ethernet VPNs: https://netdevconf.org/2.2/session.html?prabhu-linuxbridge-tutorial
2. Linux bridge driver: https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/tree/net/bridge
3. Linux bridge FDB API https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/tree/net/core/rtnetlink.c#n3571
4. Api to add externally learnt entries with NTF_EXT_LEARNED: https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/tree/net/bridge/br_fdb.c#n1086
5. Linux bridge NTF_USE flag to refresh bridge forwarding entries via control plane: https://patchwork.ozlabs.org/patch/124774/
6. A Network Virtualization Overlay Solution using EVPN: https://tools.ietf.org/html/draft-ietf-bess-evpn-overlay-12
7. FRR routing: https://frrouting.org/ Use rhashtables for bridge FDB: http://patchwork.ozlabs.org/patch/847503/
8. Use rhashtables for bridge FDB: http://patchwork.ozlabs.org/patch/847503/
9. bridge arp nd suppression: https://patchwork.ozlabs.org/cover/822906/
10. bridge multicast database: http://man7.org/linux/man-pages/man8/bridge.8.html
11. net: bridge: add support for backup port: https://patchwork.ozlabs.org/cover/947461/
12. MLAG https://en.wikipedia.org/wiki/MC-LAG
13. E-VPN Multihoming: https://tools.ietf.org/html/rfc8365#section-8
14. E-VPN Multihoming: Fast convergence: https://tools.ietf.org/html/rfc8365#section-8.1.2
15. E-VPN Designated forwarder: https://tools.ietf.org/html/rfc8365#section-8.1.5
16. E-VPN multihoming split horizon: https://tools.ietf.org/html/rfc8365#section-8.1.3
17. E-VPN Aliasing and Backup Path: https://tools.ietf.org/html/rfc8365#section-8.1.4
18. Route Nexthop groups: https://lwn.net/Articles/763950/
19. IGMP MLD proxy for E-VPN https://tools.ietf.org/html/draft-ietf-bess-evpn-igmp-mld-proxy-02

Author Biography

Roopa Prabhu leads the kernel networking group at Cumulus Networks. At Cumulus she works on networking in the Linux kernel and user-space, Network interface management and other system infrastructure areas. Her previous experience includes Linux clusters, ethernet drivers and Linux KVM virtualization platforms.

Nikolay initially was a FreeBSD kernel developer working on proprietary network projects, then joined Red Hat where he maintained a few Linux network drivers internally and started contributing to upstream. Now he is a part of Cumulus Networks, working on the Linux network stack and actively contributing to upstream.