MLAG on Linux - Lessons Learned

Scott Emery, Wilson Kok
Cumulus Networks Inc.

Agenda

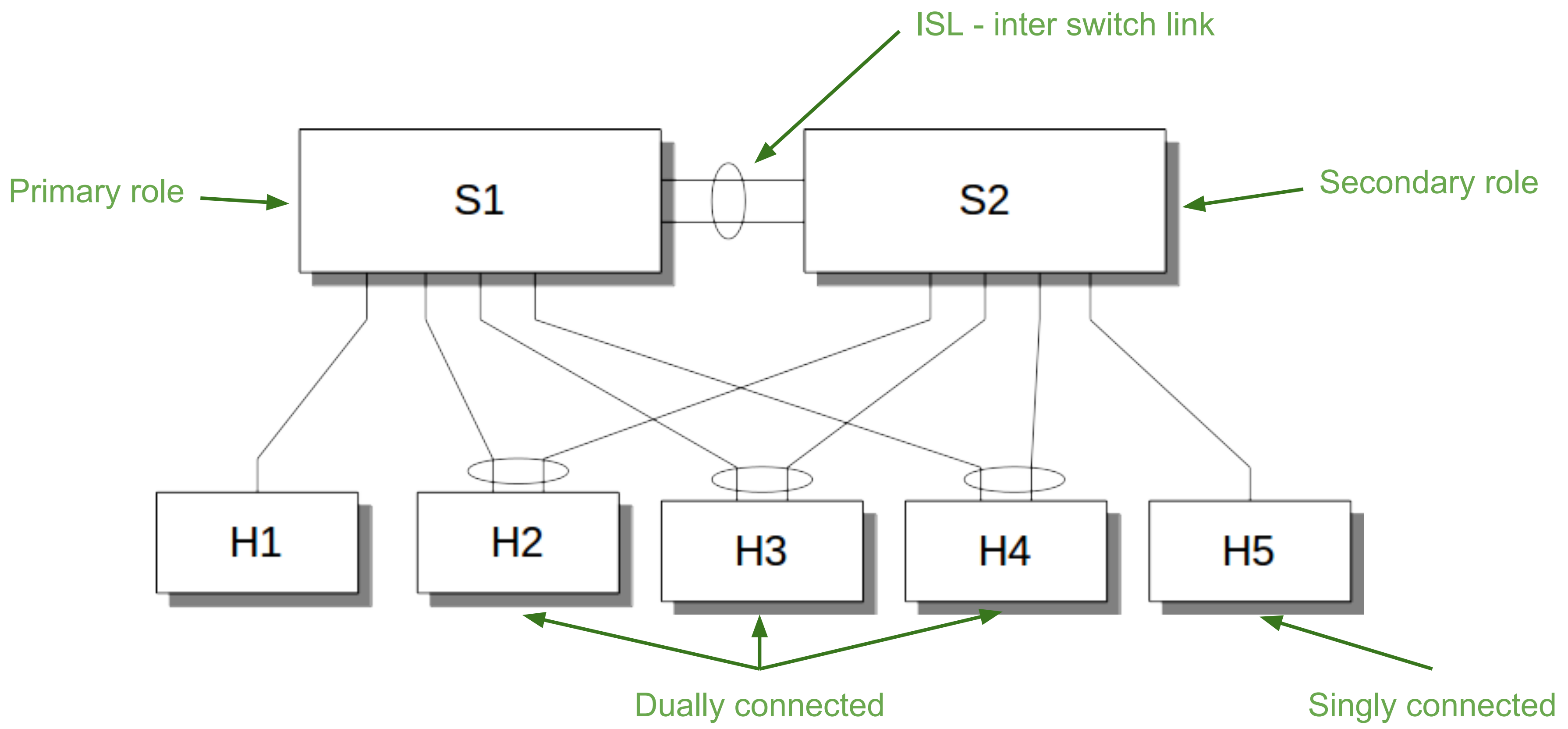
- MLAG introduction and use cases
- Lessons learned
- MLAG control plane model
- MLAG data plane
- Linux kernel requirements
- Other important changes and considerations

MLAG introduction

MLAG - a LAG across more than one node

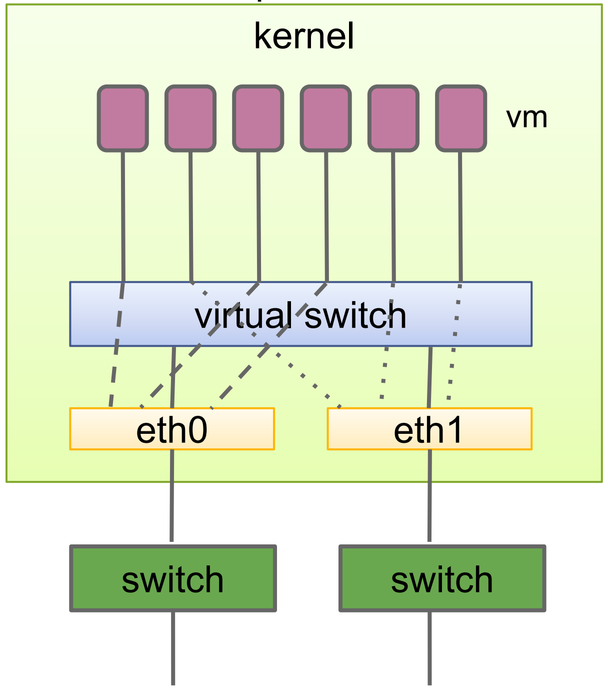
- multi-homing for redundancy
- active-active to utilize all links which otherwise may get blocked by Spanning Tree
- no modification of LAG partner

MLAG terminology

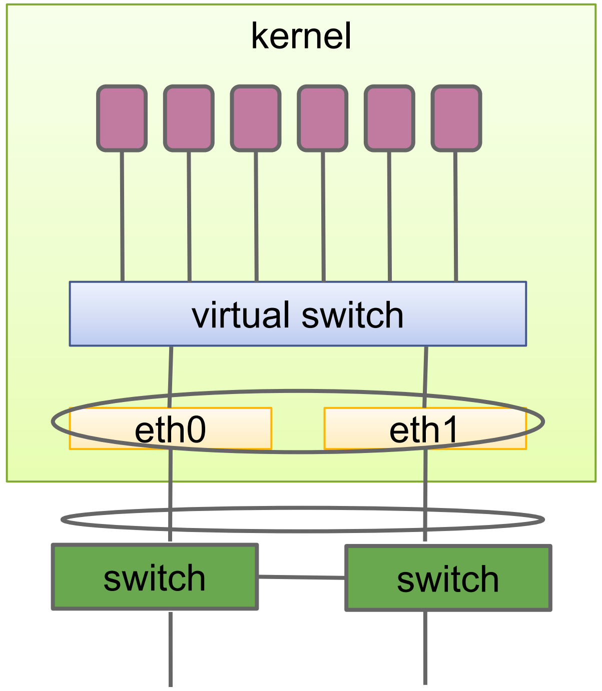


MLAG use case - hypervisor

no MLAG - striping by VM MACs
or other policies

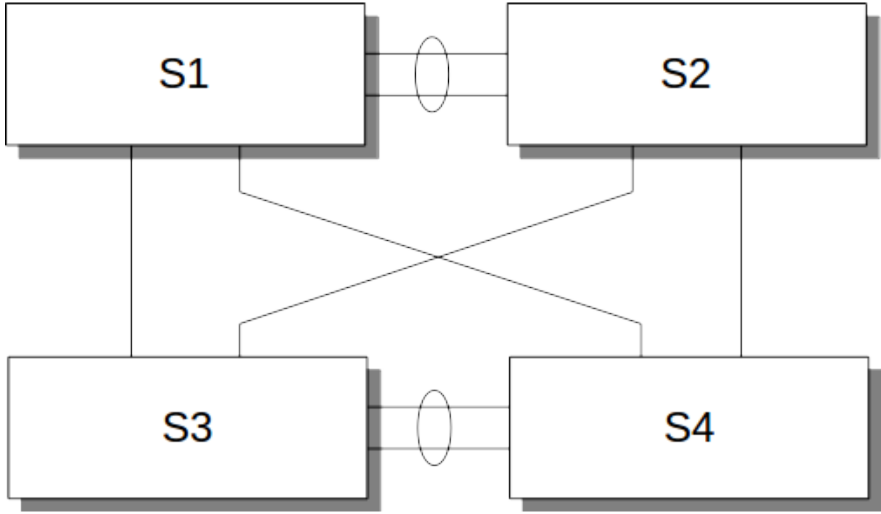


MLAG - it’s a bond



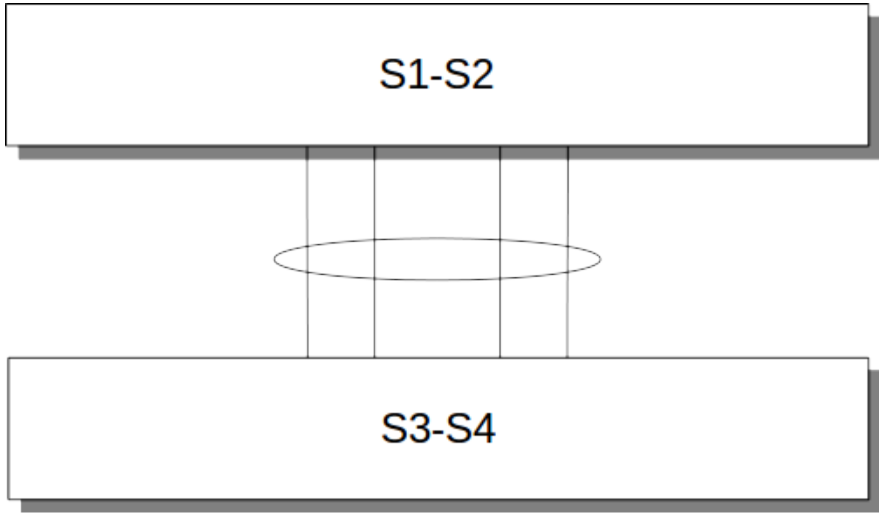
MLAG use case - L2 fabric

- no blocking links, full utilization of bandwidth
- load balancing and redundancy offered by LAG



MLAG use case - L2 fabric

- no blocking links, full utilization of bandwidth
- load balancing and redundancy offered by LAG



Lessons learned

- L2 can be dangerous! Fail open by default, no TTL, unknown means flood...
- MLAG - more ways to live dangerously
- Rigorous and conservative interface state management needed. Temporary loops or duplicates not acceptable

Lessons learned

- Fast convergence depends on a lot of things done right:
    - Proper daemon up/down sequences:
        - UP: STPd up > MLAGd up > interface enable
        - DOWN: interface disable > MLAGd down > STPd down
    - Avoid split brain as much as possible:
        - changing LACP system id flaps bonds
        - have multiple heart beat channels between MLAG daemons
- Failures, besides link and node down, do happen, should not melt network. e.g. daemon crash
    - Need to fail close, e.g. monit clean up

MLAG control plane model

- Linux kernel enforces default interface state on MLAG enabled interfaces
- User space MLAG daemon maintains MLAG configuration, controls the formation of MLAG and updates interface state and data path
- Analogous to the user space Spanning Tree model

MLAG data plane

- L2 must never have loops, redundant paths are blocked
- But want to utilize all links, cannot block

Answer…..

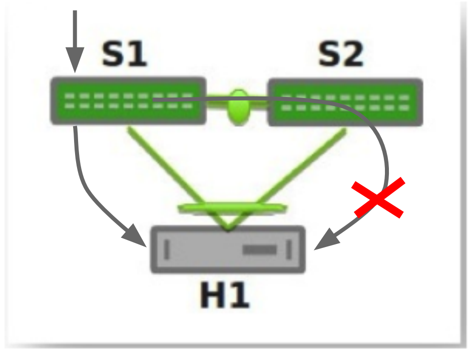
- Make the links appear logically the same for the protocols that are supposed to protect you!

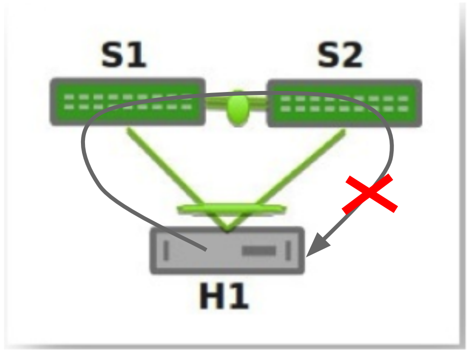
MLAG data plane rules

- same packet is not delivered to a node more than once
- packet sourced from a dually connected node is not delivered back to the same node

This means packets crossing the ISL and destined to:

- dually-connected links => drop
- singly-connected links => forward





Minimum Linux kernel requirements

- ability to set LACP system ID on bond independent of bond mac address
- mlag_enable attribute on bond
- mechanism to keep member interface carrier down independent of admin state
    - IFF_PROTO_DOWN
- duplicate filtering of packets crossing the ISL

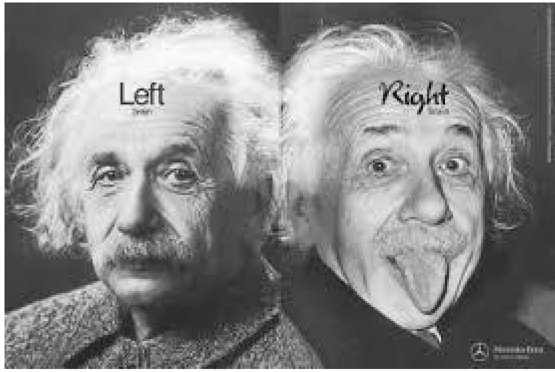
Interface bring up

- user enables an mlag (bond with mlag_enable = 1)
    - bonding driver keeps the bond and all its slaves down
- MLAG daemon puts bond in dormant interface mode to begin
- when MLAG daemon peering is complete
    - sets mlag LACP system id on bond (802.3ad mode)
    - brings slaves up
    - LACP can run, no data traffic
    - LACP converges, bond moves from oper down to oper dormant
- MLAG daemon verifies MLAG membership, installs egress filter, then sets bond to oper up

Split brain handling

- MLAG daemon pair cannot talk to each other
  - ISL down but MLAG daemons alive
- MLAG daemon with secondary role keeps all MLAGs in down state with IFF_PROTO_DOWN

- IFF_PROTO_DOWN indicates to kernel to not bring bond slaves carrier up until it is cleared



Duplicate Filtering

Packet ingress on ISL should only egress on singly connected links

- use ebtables: -i <ISL> -o <dually connected interface> -j DROP
- rule MUST be installed before dually connected interface is oper up
- rule MUST be uninstalled as soon as interface becomes singly connected

One rule per dually connected interface, not scalable, especially in the case of non VLAN-aware bridge model with many bridges and many VLANs. Better if:

- ebtables can filter on the parent interface, e.g. eth1 instead of eth1.100, eth1.101, eth1.102….
- or bridge driver can make use of the knowledge of which link is ISL and which are dual-connected

Possible other Linux kernel requirements

- interface attribute to indicate ISL
- knowledge of the ‘dual-connectedness’ of a link
- knowledge of mlag id of interfaces
- bridge filtering modifications based upon above

Other important changes and considerations

- Spanning Tree changes
- MAC address management
- IGMP group membership handling
- MLAG control traffic treatment

Spanning Tree Changes

- STP daemon connects to MLAG daemon and learns
  - which is ISL
  - singly/dually connected interfaces and their MLAG id
  - when MLAG peering is up or down
- STP needs to run as if the two switches are one. Multiple approaches possible:
  - master STP daemon runs the protocol and maintains full state sync with the slave STP daemon

  or

  - each STP daemon does independent calculation. Loosely coupled, distributed processing
- Loosely coupled model is simpler and more scalable

Spanning Tree - Loosely coupled model

- use common bridge id (MLAG system id) when generating BPDUs
- only MLAG primary switch sends BPDU on dually-connected links
- both MLAG switches send BPDU on singly-connected links
- BPDU received on root port is processed and also relayed across ISL, replace source MAC with MLAG id

MAC address management

Goals

- reduce unknown flood
- eliminate constant MAC moves between ISL and MLAG

Solution

- disable learning on ISL
- synchronize MAC addresses
    - install address learned on MLAG on one side to corresponding MLAG on the other side
    - install address learned on singly connected link on ISL on other side

IGMP Snooping

MLAG daemons synchronize between themselves:

- IGMP group membership for dually connected links
- mrouter port information
- reports/queries may need to be flooded, the same duplicate filtering rule applies

MLAG control traffic

control traffic share the ISL with data traffic, needs to be

- given higher priority
- independent of data traffic topology change - use a separate VLAN device on the ISL which is not bridged

While we’re at it...

- VLAN-aware bridge driver
  - great enhancement!
  - more work needed
    - scalability: vlan range*, per port per vlan local fdb*
    - usability: limited to single STP instance, per bridge igmp snooping control
- Bonding driver
  - a few issues with slave active state setting and MUX machine transitions*

(*patches submitted upstream)

Thank You!